



Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

March 2023

Notice: The Intel® 64 and IA-32 architectures may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are documented in the specification updates.

Document Number: 252046-071



Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

All product plans and roadmaps are subject to change without notice.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), <https://opensource.org/licenses/0BSD>. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Contents

Revision History	4
Preface	7
Summary Tables of Changes	8
Documentation Changes	9

Revision History

Revision	Description	Date
-001	<ul style="list-style-type: none"> Initial release 	November 2002
-002	<ul style="list-style-type: none"> Added 1-10 Documentation Changes. Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual 	December 2002
-003	<ul style="list-style-type: none"> Added 9 -17 Documentation Changes. Removed Documentation Change #6 - References to bits Gen and Len Deleted. Removed Documentation Change #4 - VIF Information Added to CLI Discussion 	February 2003
-004	<ul style="list-style-type: none"> Removed Documentation changes 1-17. Added Documentation changes 1-24. 	June 2003
-005	<ul style="list-style-type: none"> Removed Documentation Changes 1-24. Added Documentation Changes 1-15. 	September 2003
-006	<ul style="list-style-type: none"> Added Documentation Changes 16- 34. 	November 2003
-007	<ul style="list-style-type: none"> Updated Documentation changes 14, 16, 17, and 28. Added Documentation Changes 35-45. 	January 2004
-008	<ul style="list-style-type: none"> Removed Documentation Changes 1-45. Added Documentation Changes 1-5. 	March 2004
-009	<ul style="list-style-type: none"> Added Documentation Changes 7-27. 	May 2004
-010	<ul style="list-style-type: none"> Removed Documentation Changes 1-27. Added Documentation Changes 1. 	August 2004
-011	<ul style="list-style-type: none"> Added Documentation Changes 2-28. 	November 2004
-012	<ul style="list-style-type: none"> Removed Documentation Changes 1-28. Added Documentation Changes 1-16. 	March 2005
-013	<ul style="list-style-type: none"> Updated title. There are no Documentation Changes for this revision of the document. 	July 2005
-014	<ul style="list-style-type: none"> Added Documentation Changes 1-21. 	September 2005
-015	<ul style="list-style-type: none"> Removed Documentation Changes 1-21. Added Documentation Changes 1-20. 	March 9, 2006
-016	<ul style="list-style-type: none"> Added Documentation changes 21-23. 	March 27, 2006
-017	<ul style="list-style-type: none"> Removed Documentation Changes 1-23. Added Documentation Changes 1-36. 	September 2006
-018	<ul style="list-style-type: none"> Added Documentation Changes 37-42. 	October 2006
-019	<ul style="list-style-type: none"> Removed Documentation Changes 1-42. Added Documentation Changes 1-19. 	March 2007
-020	<ul style="list-style-type: none"> Added Documentation Changes 20-27. 	May 2007
-021	<ul style="list-style-type: none"> Removed Documentation Changes 1-27. Added Documentation Changes 1-6 	November 2007
-022	<ul style="list-style-type: none"> Removed Documentation Changes 1-6 Added Documentation Changes 1-6 	August 2008
-023	<ul style="list-style-type: none"> Removed Documentation Changes 1-6 Added Documentation Changes 1-21 	March 2009

Revision	Description	Date
-024	<ul style="list-style-type: none"> Removed Documentation Changes 1-21 Added Documentation Changes 1-16 	June 2009
-025	<ul style="list-style-type: none"> Removed Documentation Changes 1-16 Added Documentation Changes 1-18 	September 2009
-026	<ul style="list-style-type: none"> Removed Documentation Changes 1-18 Added Documentation Changes 1-15 	December 2009
-027	<ul style="list-style-type: none"> Removed Documentation Changes 1-15 Added Documentation Changes 1-24 	March 2010
-028	<ul style="list-style-type: none"> Removed Documentation Changes 1-24 Added Documentation Changes 1-29 	June 2010
-029	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	September 2010
-030	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	January 2011
-031	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	April 2011
-032	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-14 	May 2011
-033	<ul style="list-style-type: none"> Removed Documentation Changes 1-14 Added Documentation Changes 1-38 	October 2011
-034	<ul style="list-style-type: none"> Removed Documentation Changes 1-38 Added Documentation Changes 1-16 	December 2011
-035	<ul style="list-style-type: none"> Removed Documentation Changes 1-16 Added Documentation Changes 1-18 	March 2012
-036	<ul style="list-style-type: none"> Removed Documentation Changes 1-18 Added Documentation Changes 1-17 	May 2012
-037	<ul style="list-style-type: none"> Removed Documentation Changes 1-17 Added Documentation Changes 1-28 	August 2012
-038	<ul style="list-style-type: none"> Removed Documentation Changes 1-28 Add Documentation Changes 1-22 	January 2013
-039	<ul style="list-style-type: none"> Removed Documentation Changes 1-22 Add Documentation Changes 1-17 	June 2013
-040	<ul style="list-style-type: none"> Removed Documentation Changes 1-17 Add Documentation Changes 1-24 	September 2013
-041	<ul style="list-style-type: none"> Removed Documentation Changes 1-24 Add Documentation Changes 1-20 	February 2014
-042	<ul style="list-style-type: none"> Removed Documentation Changes 1-20 Add Documentation Changes 1-8 	February 2014
-043	<ul style="list-style-type: none"> Removed Documentation Changes 1-8 Add Documentation Changes 1-43 	June 2014
-044	<ul style="list-style-type: none"> Removed Documentation Changes 1-43 Add Documentation Changes 1-12 	September 2014
-045	<ul style="list-style-type: none"> Removed Documentation Changes 1-12 Add Documentation Changes 1-22 	January 2015
-046	<ul style="list-style-type: none"> Removed Documentation Changes 1-22 Add Documentation Changes 1-25 	April 2015
-047	<ul style="list-style-type: none"> Removed Documentation Changes 1-25 Add Documentation Changes 1-19 	June 2015

Revision	Description	Date
-048	<ul style="list-style-type: none"> Removed Documentation Changes 1-19 Add Documentation Changes 1-33 	September 2015
-049	<ul style="list-style-type: none"> Removed Documentation Changes 1-33 Add Documentation Changes 1-33 	December 2015
-050	<ul style="list-style-type: none"> Removed Documentation Changes 1-33 Add Documentation Changes 1-9 	April 2016
-051	<ul style="list-style-type: none"> Removed Documentation Changes 1-9 Add Documentation Changes 1-20 	June 2016
-052	<ul style="list-style-type: none"> Removed Documentation Changes 1-20 Add Documentation Changes 1-22 	September 2016
-053	<ul style="list-style-type: none"> Removed Documentation Changes 1-22 Add Documentation Changes 1-26 	December 2016
-054	<ul style="list-style-type: none"> Removed Documentation Changes 1-26 Add Documentation Changes 1-20 	March 2017
-055	<ul style="list-style-type: none"> Removed Documentation Changes 1-20 Add Documentation Changes 1-28 	July 2017
-056	<ul style="list-style-type: none"> Removed Documentation Changes 1-28 Add Documentation Changes 1-18 	October 2017
-057	<ul style="list-style-type: none"> Removed Documentation Changes 1-18 Add Documentation Changes 1-29 	December 2017
-058	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Add Documentation Changes 1-17 	March 2018
-059	<ul style="list-style-type: none"> Removed Documentation Changes 1-17 Add Documentation Changes 1-24 	May 2018
-060	<ul style="list-style-type: none"> Removed Documentation Changes 1-24 Add Documentation Changes 1-23 	November 2018
-061	<ul style="list-style-type: none"> Removed Documentation Changes 1-23 Add Documentation Changes 1-21 	January 2019
-062	<ul style="list-style-type: none"> Removed Documentation Changes 1-21 Add Documentation Changes 1-28 	May 2019
-063	<ul style="list-style-type: none"> Removed Documentation Changes 1-28 Add Documentation Changes 1-34 	October 2019
-064	<ul style="list-style-type: none"> Removed Documentation Changes 1-34 Add Documentation Changes 1-36 	May 2020
-065	<ul style="list-style-type: none"> Removed Documentation Changes 1-36 Add Documentation Changes 1-31 	November 2020
-066	<ul style="list-style-type: none"> Removed Documentation Changes 1-31 Add Documentation Changes 1-24 	April 2021
-067	<ul style="list-style-type: none"> Removed Documentation Changes 1-24 Add Documentation Changes 1-30 	June 2021
-068	<ul style="list-style-type: none"> Removed Documentation Changes 1-30 Add Documentation Changes 1-29 	December 2021
-069	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Add Documentation Changes 1-18 	April 2022
-070	<ul style="list-style-type: none"> Removed Documentation Changes 1-18 Add Documentation Changes 1-41 	December 2022
-071	<ul style="list-style-type: none"> Removed Documentation Changes 1-41 Add Documentation Changes 1-23 	March 2023



§



Preface

This document is an update to the specifications contained in the [Affected Documents](#) table below. This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

Affected Documents

Document Title	Document Number/ Location
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture</i>	253665
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L</i>	253666
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-U</i>	253667
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference, V</i>	326018
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference, W-Z</i>	334569
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1</i>	253668
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2</i>	253669
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3</i>	326019
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4</i>	332831
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model Specific Registers</i>	335592

Nomenclature

Documentation Changes include typos, errors, or omissions from the current published specifications. These will be incorporated in any new release of the specification.

Summary Tables of Changes

The following table indicates documentation changes which apply to the Intel® 64 and IA-32 architectures. This table uses the following notations:

Codes Used in Summary Tables

Change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 2, Volume 1
2	Updates to Chapter 3, Volume 2A
3	Updates to Chapter 4, Volume 2B
4	Updates to Chapter 5, Volume 2C
5	Updates to Chapter 6, Volume 2D
6	Updates to Chapter 9, Volume 3A
7	Updates to Chapter 10, Volume 3A
8	Updates to Chapter 15, Volume 3B
9	Updates to Chapter 17, Volume 3B
10	Updates to Chapter 18, Volume 3B
11	Updates to Chapter 20, Volume 3B
12	Updates to Chapter 25, Volume 3C
13	Updates to Chapter 26, Volume 3C
14	Updates to Chapter 27, Volume 3C
15	Updates to Chapter 28, Volume 3C
16	Updates to Chapter 32, Volume 3C
17	Updates to Chapter 35, Volume 3D
18	Updates to Chapter 36, Volume 3D
19	Updates to Chapter 38, Volume 3D
20	Updates to Chapter 40, Volume 3D
21	Updates to Appendix B, Volume 3D
22	Updates to Appendix C, Volume 3D
23	Updates to Chapter 2, Volume 4

Documentation Changes

Changes to the Intel® 64 and IA-32 Architectures Software Developer's Manual volumes follow, and are listed by chapter. Only chapters with changes are included in this document.

1. Updates to Chapter 2, Volume 1

Change bars and green text show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- Updated Section 2.4 name from "Proposed Removal of Intel® Instruction Set Architecture and Features from Upcoming Products" to "Planned Removal of Intel® Instruction Set Architecture and Features from Upcoming Products," and added an entry to Table 2-4, "Planned Intel® ISA and Features Removal List."

2.1 BRIEF HISTORY OF INTEL® 64 AND IA-32 ARCHITECTURE

The following sections provide a summary of the major technical evolutions from IA-32 to Intel 64 architecture: starting from the Intel 8086 processor to the latest Intel® Core® 2 Duo, Core 2 Quad and Intel Xeon processor 5300 and 7300 series. Object code created for processors released as early as 1978 still executes on the latest processors in the Intel 64 and IA-32 architecture families.

2.1.1 16-bit Processors and Segmentation (1978)

The IA-32 architecture family was preceded by 16-bit processors, the 8086 and 8088. The 8086 has 16-bit registers and a 16-bit external data bus, with 20-bit addressing giving a 1-MByte address space. The 8088 is similar to the 8086 except it has an 8-bit external data bus.

The 8086/8088 introduced segmentation to the IA-32 architecture. With segmentation, a 16-bit segment register contains a pointer to a memory segment of up to 64 KBytes. Using four segment registers at a time, 8086/8088 processors are able to address up to 256 KBytes without switching between segments. The 20-bit addresses that can be formed using a segment register and an additional 16-bit pointer provide a total address range of 1 MByte.

2.1.2 The Intel® 286 Processor (1982)

The Intel 286 processor introduced protected mode operation into the IA-32 architecture. Protected mode uses the segment register content as selectors or pointers into descriptor tables. Descriptors provide 24-bit base addresses with a physical memory size of up to 16 MBytes, support for virtual memory management on a segment swapping basis, and a number of protection mechanisms. These mechanisms include:

- Segment limit checking.
- Read-only and execute-only segment options.
- Four privilege levels.

2.1.3 The Intel386™ Processor (1985)

The Intel386 processor was the first 32-bit processor in the IA-32 architecture family. It introduced 32-bit registers for use both to hold operands and for addressing. The lower half of each 32-bit Intel386 register retains the properties of the 16-bit registers of earlier generations, permitting backward compatibility. The processor also provides a virtual-8086 mode that allows for even greater efficiency when executing programs created for 8086/8088 processors.

In addition, the Intel386 processor has support for:

- A 32-bit address bus that supports up to 4-GBytes of physical memory.
- A segmented-memory model and a flat memory model.
- Paging, with a fixed 4-KByte page size providing a method for virtual memory management.
- Support for parallel stages.

2.1.4 The Intel486™ Processor (1989)

The Intel486™ processor added more parallel execution capability by expanding the Intel386 processor's instruction decode and execution units into five pipelined stages. Each stage operates in parallel with the others on up to five instructions in different stages of execution.

In addition, the processor added:

- An 8-KByte on-chip first-level cache that increased the percent of instructions that could execute at the scalar rate of one per clock.
- An integrated x87 FPU.
- Power saving and system management capabilities.

2.1.5 The Intel® Pentium® Processor (1993)

The introduction of the Intel Pentium processor added a second execution pipeline to achieve superscalar performance (two pipelines, known as u and v, together can execute two instructions per clock). The on-chip first-level cache doubled, with 8 KBytes devoted to code and another 8 KBytes devoted to data. The data cache uses the MESI protocol to support more efficient write-back cache in addition to the write-through cache previously used by the Intel486 processor. Branch prediction with an on-chip branch table was added to increase performance in looping constructs.

In addition, the processor added:

- Extensions to make the virtual-8086 mode more efficient and allow for 4-MByte as well as 4-KByte pages.
- Internal data paths of 128 and 256 bits add speed to internal data transfers.
- Burstable external data bus was increased to 64 bits.
- An APIC to support systems with multiple processors.
- A dual processor mode to support glueless two processor systems.

A subsequent stepping of the Pentium family introduced Intel MMX technology (the Pentium Processor with MMX technology). Intel MMX technology uses the single-instruction, multiple-data (SIMD) execution model to perform parallel computations on packed integer data contained in 64-bit registers.

See Section 2.2.7, “SIMD Instructions.”

2.1.6 The P6 Family of Processors (1995-1999)

The P6 family of processors was based on a superscalar microarchitecture that set new performance standards; see also Section 2.2.1, “P6 Family Microarchitecture.” One of the goals in the design of the P6 family microarchitecture was to exceed the performance of the Pentium processor significantly while using the same 0.6-micrometer, four-layer, metal BICMOS manufacturing process. Members of this family include the following:

- The **Intel Pentium Pro processor** is three-way superscalar. Using parallel processing techniques, the processor is able on average to decode, dispatch, and complete execution of (retire) three instructions per clock cycle. The Pentium Pro introduced the dynamic execution (micro-data flow analysis, out-of-order execution, superior branch prediction, and speculative execution) in a superscalar implementation. The processor was further enhanced by its caches. It has the same two on-chip 8-KByte 1st-Level caches as the Pentium processor and an additional 256-KByte Level 2 cache in the same package as the processor.
- The **Intel Pentium II processor** added Intel MMX technology to the P6 family processors along with new packaging and several hardware enhancements. The processor core is packaged in the single edge contact cartridge (SECC). The Level 1 data and instruction caches were enlarged to 16 KBytes each, and Level 2 cache sizes of 256 KBytes, 512 KBytes, and 1 MByte are supported. A half-frequency backside bus connects the Level 2 cache to the processor. Multiple low-power states such as AutoHALT, Stop-Grant, Sleep, and Deep Sleep are supported to conserve power when idling.
- The **Pentium II Xeon processor** combined the premium characteristics of previous generations of Intel processors. This includes: 4-way, 8-way (and up) scalability and a 2 MByte 2nd-Level cache running on a full-frequency backside bus.
- The **Intel Celeron processor** family focused on the value PC market segment. Its introduction offers an integrated 128 KBytes of Level 2 cache and a plastic pin grid array (P.P.G.A.) form factor to lower system design cost.
- The **Intel Pentium III processor** introduced the Streaming SIMD Extensions (SSE) to the IA-32 architecture. SSE extensions expand the SIMD execution model introduced with the Intel MMX technology by providing a

new set of 128-bit registers and the ability to perform SIMD operations on packed single precision floating-point values. See Section 2.2.7, “SIMD Instructions.”

- The **Pentium III Xeon processor** extended the performance levels of the IA-32 processors with the enhancement of a full-speed, on-die, and Advanced Transfer Cache.

2.1.7 The Intel® Pentium® 4 Processor Family (2000-2006)

The Intel Pentium 4 processor family is based on Intel NetBurst microarchitecture; see Section 2.2.2, “Intel NetBurst® Microarchitecture.”

The Intel Pentium 4 processor introduced Streaming SIMD Extensions 2 (SSE2); see Section 2.2.7, “SIMD Instructions.” The Intel Pentium 4 processor 3.40 GHz, supporting Hyper-Threading Technology introduced Streaming SIMD Extensions 3 (SSE3); see Section 2.2.7, “SIMD Instructions.”

Intel 64 architecture was introduced in the Intel Pentium 4 Processor Extreme Edition supporting Hyper-Threading Technology and in the Intel Pentium 4 Processor 6xx and 5xx sequences.

Intel® Virtualization Technology (Intel® VT) was introduced in the Intel Pentium 4 processor 672 and 662.

2.1.8 The Intel® Xeon® Processor (2001- 2007)

Intel Xeon processors (with exception for dual-core Intel Xeon processor LV, Intel Xeon processor 5100 series) are based on the Intel NetBurst microarchitecture; see Section 2.2.2, “Intel NetBurst® Microarchitecture.” As a family, this group of IA-32 processors (more recently Intel 64 processors) is designed for use in multi-processor server systems and high-performance workstations.

The Intel Xeon processor MP introduced support for Intel® Hyper-Threading Technology; see Section 2.2.8, “Intel® Hyper-Threading Technology.”

The 64-bit Intel Xeon processor 3.60 GHz (with an 800 MHz System Bus) was used to introduce Intel 64 architecture. The Dual-Core Intel Xeon processor includes dual core technology. The Intel Xeon processor 70xx series includes Intel Virtualization Technology.

The Intel Xeon processor 5100 series introduces power-efficient, high performance Intel Core microarchitecture. This processor is based on Intel 64 architecture; it includes Intel Virtualization Technology and dual-core technology. The Intel Xeon processor 3000 series are also based on Intel Core microarchitecture. The Intel Xeon processor 5300 series introduces four processor cores in a physical package, they are also based on Intel Core microarchitecture.

2.1.9 The Intel® Pentium® M Processor (2003-2006)

The Intel Pentium M processor family is a high performance, low power mobile processor family with microarchitectural enhancements over previous generations of IA-32 Intel mobile processors. This family is designed for extending battery life and seamless integration with platform innovations that enable new usage models (such as extended mobility, ultra thin form-factors, and integrated wireless networking).

Its enhanced microarchitecture includes:

- Support for Intel Architecture with Dynamic Execution.
- A high performance, low-power core manufactured using Intel’s advanced process technology with copper interconnect.
- On-die, primary 32-KByte instruction cache and 32-KByte write-back data cache.
- On-die, second-level cache (up to 2 MByte) with Advanced Transfer Cache Architecture.
- Advanced Branch Prediction and Data Prefetch Logic.
- Support for MMX technology, Streaming SIMD instructions, and the SSE2 instruction set.
- A 400 or 533 MHz, Source-Synchronous Processor System Bus.
- Advanced power management using Enhanced Intel SpeedStep® technology.

2.1.10 The Intel® Pentium® Processor Extreme Edition (2005)

The Intel Pentium processor Extreme Edition introduced dual-core technology. This technology provides advanced hardware multi-threading support. The processor is based on Intel NetBurst microarchitecture and supports Intel SSE, SSE2, SSE3, Intel Hyper-Threading Technology, and Intel 64 architecture.

See also:

- Section 2.2.2, “Intel NetBurst® Microarchitecture.”
- Section 2.2.3, “Intel® Core™ Microarchitecture.”
- Section 2.2.7, “SIMD Instructions.”
- Section 2.2.8, “Intel® Hyper-Threading Technology.”
- Section 2.2.9, “Multi-Core Technology.”
- Section 2.2.10, “Intel® 64 Architecture.”

2.1.11 The Intel® Core™ Duo and Intel® Core™ Solo Processors (2006-2007)

The Intel Core Duo processor offers power-efficient, dual-core performance with a low-power design that extends battery life. This family and the single-core Intel Core Solo processor offer microarchitectural enhancements over Pentium M processor family.

Its enhanced microarchitecture includes:

- Intel® Smart Cache which allows for efficient data sharing between two processor cores.
- Improved decoding and SIMD execution.
- Intel® Dynamic Power Coordination and Enhanced Intel® Deeper Sleep to reduce power consumption.
- Intel® Advanced Thermal Manager which features digital thermal sensor interfaces.
- Support for power-optimized 667 MHz bus.

The dual-core Intel Xeon processor LV is based on the same microarchitecture as Intel Core Duo processor, and supports IA-32 architecture.

2.1.12 The Intel® Xeon® Processor 5100, 5300 Series and Intel® Core™ 2 Processor Family (2006)

The Intel Xeon processor 3000, 3200, 5100, 5300, and 7300 series, Intel Pentium Dual-Core, Intel Core 2 Extreme, Intel Core 2 Quad processors, and Intel Core 2 Duo processor family support Intel 64 architecture; they are based on the high-performance, power-efficient Intel® Core microarchitecture built on 65 nm process technology. The Intel Core microarchitecture includes the following innovative features:

- Intel® Wide Dynamic Execution to increase performance and execution throughput.
- Intel® Intelligent Power Capability to reduce power consumption.
- Intel® Advanced Smart Cache which allows for efficient data sharing between two processor cores.
- Intel® Smart Memory Access to increase data bandwidth and hide latency of memory accesses.
- Intel® Advanced Digital Media Boost which improves application performance using multiple generations of Streaming SIMD extensions.

The Intel Xeon processor 5300 series, Intel Core 2 Extreme processor QX6800 series, and Intel Core 2 Quad processors support Intel quad-core technology.

2.1.13 The Intel® Xeon® Processor 5200, 5400, 7400 Series and Intel® Core™ 2 Processor Family (2007)

The Intel Xeon processor 5200, 5400, and 7400 series, Intel Core 2 Quad processor Q9000 Series, Intel Core 2 Duo processor E8000 series support Intel 64 architecture; they are based on the Enhanced Intel® Core microarchitec-

ture using 45 nm process technology. The Enhanced Intel Core microarchitecture provides the following improved features:

- A radix-16 divider, faster OS primitives further increases the performance of Intel® Wide Dynamic Execution.
- Improves Intel® Advanced Smart Cache with Up to 50% larger level-two cache and up to 50% increase in way-set associativity.
- A 128-bit shuffler engine significantly improves the performance of Intel® Advanced Digital Media Boost and SSE4.

The Intel Xeon processor 5400 series and the Intel Core 2 Quad processor Q9000 Series support Intel quad-core technology. The Intel Xeon processor 7400 series offers up to six processor cores and an L3 cache up to 16 MBytes.

2.1.14 The Intel Atom® Processor Family (2008)

The first generation of Intel Atom® processors are built on 45 nm process technology. They are based on a new microarchitecture, Intel Atom® microarchitecture, which is optimized for ultra low power devices. The Intel Atom® microarchitecture features two in-order execution pipelines that minimize power consumption, increase battery life, and enable ultra-small form factors. The initial Intel Atom Processor family and subsequent generations including Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series provide the following features:

- Enhanced Intel® SpeedStep® Technology.
- Intel® Hyper-Threading Technology.
- Deep Power Down Technology with Dynamic Cache Sizing.
- Support for instruction set extensions up to and including Supplemental Streaming SIMD Extensions 3 (SSSE3).
- Support for Intel® Virtualization Technology.
- Support for Intel® 64 Architecture (excluding Intel Atom processor Z5xx Series).

2.1.15 The Intel Atom® Processor Family Based on Silvermont Microarchitecture (2013)

Intel Atom Processor C2xxx, E3xxx, S1xxx series are based on the Silvermont microarchitecture. Processors based on the Silvermont microarchitecture supports instruction set extensions up to and including SSE4.2, AESNI, and PCLMULQDQ.

2.1.16 The Intel® Core™ i7 Processor Family (2008)

The Intel Core i7 processor 900 series supports Intel 64 architecture, and is based on Nehalem microarchitecture using 45 nm process technology. The Intel Core i7 processor and Intel Xeon processor 5500 series include the following features:

- Intel® Turbo Boost Technology converts thermal headroom into higher performance.
- Intel® HyperThreading Technology in conjunction with Quadcore to provide four cores and eight threads.
- Dedicated power control unit to reduce active and idle power consumption.
- Integrated memory controller on the processor supporting three channel of DDR3 memory.
- 8 MB inclusive Intel® Smart Cache.
- Intel® QuickPath interconnect (QPI) providing point-to-point link to chipset.
- Support for SSE4.2 and SSE4.1 instruction sets.
- Second generation Intel Virtualization Technology.

2.1.17 The Intel® Xeon® Processor 7500 Series (2010)

The Intel Xeon processor 7500 and 6500 series are based on Nehalem microarchitecture using 45 nm process technology. These processors support the same features described in Section 2.1.16, plus the following features:

- Up to eight cores per physical processor package.
- Up to 24 MB inclusive Intel® Smart Cache.
- Provides Intel® Scalable Memory Interconnect (Intel® SMI) channels with Intel® 7500 Scalable Memory Buffer to connect to system memory.
- Advanced RAS supporting software recoverable machine check architecture.

2.1.18 2010 Intel® Core™ Processor Family (2010)

The 2010 Intel Core processor family spans Intel Core i7, i5, and i3 processors. These processors are based on Westmere microarchitecture using 32 nm process technology. The features can include:

- Deliver smart performance using Intel Hyper-Threading Technology plus Intel Turbo Boost Technology.
- Enhanced Intel Smart Cache and integrated memory controller.
- Intelligent power gating.
- Repartitioned platform with on-die integration of 45 nm integrated graphics.
- Range of instruction set support up to AESNI, PCLMULQDQ, SSE4.2 and SSE4.1.

2.1.19 The Intel® Xeon® Processor 5600 Series (2010)

The Intel Xeon processor 5600 series are based on Westmere microarchitecture using 32 nm process technology. They support the same features described in Section 2.1.16, plus the following features:

- Up to six cores per physical processor package.
- Up to 12 MB enhanced Intel® Smart Cache.
- Support for AESNI, PCLMULQDQ, SSE4.2 and SSE4.1 instruction sets.
- Flexible Intel Virtualization Technologies across processor and I/O.

2.1.20 The Second Generation Intel® Core™ Processor Family (2011)

The Second Generation Intel Core processor family spans Intel Core i7, i5, and i3 processors based on the Sandy Bridge microarchitecture. These processors are built from 32 nm process technology and have features including:

- Intel Turbo Boost Technology for Intel Core i5 and i7 processors.
- Intel Hyper-Threading Technology.
- Enhanced Intel Smart Cache and integrated memory controller.
- Processor graphics and built-in visual features like Intel® Quick Sync Video, Intel® Insider™, etc.
- Range of instruction set support up to AVX, AESNI, PCLMULQDQ, SSE4.2 and SSE4.1.

The Intel Xeon processor E3-1200 product family is also based on the Sandy Bridge microarchitecture.

The Intel Xeon processor E5-2400/1400 product families are based on the Sandy Bridge-EP microarchitecture.

The Intel Xeon processor E5-4600/2600/1600 product families are based on the Sandy Bridge-EP microarchitecture and provide support for multiple sockets.

2.1.21 The Third Generation Intel® Core™ Processor Family (2012)

The Third Generation Intel Core processor family spans Intel Core i7, i5, and i3 processors based on the Ivy Bridge microarchitecture. The Intel Xeon processor E7-8800/4800/2800 v2 product families and Intel Xeon processor E3-1200 v2 product family are also based on the Ivy Bridge microarchitecture.

The Intel Xeon processor E5-2400/1400 v2 product families are based on the Ivy Bridge-EP microarchitecture.

The Intel Xeon processor E5-4600/2600/1600 v2 product families are based on the Ivy Bridge-EP microarchitecture and provide support for multiple sockets.

2.1.22 The Fourth Generation Intel® Core™ Processor Family (2013)

The Fourth Generation Intel Core processor family spans Intel Core i7, i5, and i3 processors based on the Haswell microarchitecture. Intel Xeon processor E3-1200 v3 product family is also based on the Haswell microarchitecture.

2.2 MORE ON SPECIFIC ADVANCES

The following sections provide more information on major innovations.

2.2.1 P6 Family Microarchitecture

The Pentium Pro processor introduced a new microarchitecture commonly referred to as P6 processor microarchitecture. The P6 processor microarchitecture was later enhanced with an on-die, Level 2 cache, called Advanced Transfer Cache.

The microarchitecture is a three-way superscalar, pipelined architecture. Three-way superscalar means that by using parallel processing techniques, the processor is able on average to decode, dispatch, and complete execution of (retire) three instructions per clock cycle. To handle this level of instruction throughput, the P6 processor family uses a decoupled, 12-stage superpipeline that supports out-of-order instruction execution.

Figure 2-1 shows a conceptual view of the P6 processor microarchitecture pipeline with the Advanced Transfer Cache enhancement.

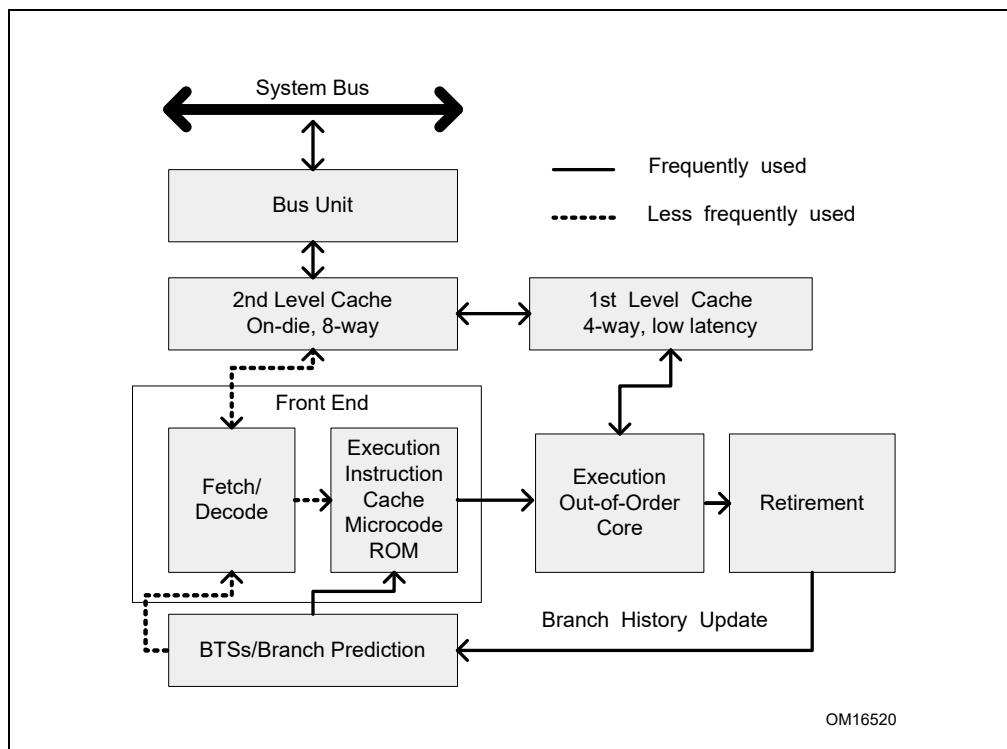


Figure 2-1. The P6 Processor Microarchitecture with Advanced Transfer Cache Enhancement

To ensure a steady supply of instructions and data for the instruction execution pipeline, the P6 processor microarchitecture incorporates two cache levels. The Level 1 cache provides an 8-KByte instruction cache and an 8-KByte data cache, both closely coupled to the pipeline. The Level 2 cache provides 256-KByte, 512-KByte, or 1-MByte static RAM that is coupled to the core processor through a full clock-speed 64-bit cache bus.

The centerpiece of the P6 processor microarchitecture is an out-of-order execution mechanism called dynamic execution. Dynamic execution incorporates three data-processing concepts:

- **Deep branch prediction** allows the processor to decode instructions beyond branches to keep the instruction pipeline full. The P6 processor family implements highly optimized branch prediction algorithms to predict the direction of the instruction.
- **Dynamic data flow analysis** requires real-time analysis of the flow of data through the processor to determine dependencies and to detect opportunities for out-of-order instruction execution. The out-of-order execution core can monitor many instructions and execute these instructions in the order that best optimizes the use of the processor's multiple execution units, while maintaining the data integrity.
- **Speculative execution** refers to the processor's ability to execute instructions that lie beyond a conditional branch that has not yet been resolved, and ultimately to commit the results in the order of the original instruction stream. To make speculative execution possible, the P6 processor microarchitecture decouples the dispatch and execution of instructions from the commitment of results. The processor's out-of-order execution core uses data-flow analysis to execute all available instructions in the instruction pool and store the results in temporary registers. The retirement unit then linearly searches the instruction pool for completed instructions that no longer have data dependencies with other instructions or unresolved branch predictions. When completed instructions are found, the retirement unit commits the results of these instructions to memory and/or the IA-32 registers (the processor's eight general-purpose registers and eight x87 FPU data registers) in the order they were originally issued and retires the instructions from the instruction pool.

2.2.2 Intel NetBurst® Microarchitecture

The Intel NetBurst microarchitecture provides:

- The Rapid Execution Engine.
 - Arithmetic Logic Units (ALUs) run at twice the processor frequency.
 - Basic integer operations can dispatch in 1/2 processor clock tick.
- Hyper-Pipelined Technology.
 - Deep pipeline to enable industry-leading clock rates for desktop PCs and servers.
 - Frequency headroom and scalability to continue leadership into the future.
- Advanced Dynamic Execution.
 - Deep, out-of-order, speculative execution engine.
 - Up to 126 instructions in flight.
 - Up to 48 loads and 24 stores in pipeline¹.
 - Enhanced branch prediction capability.
 - Reduces the misprediction penalty associated with deeper pipelines.
 - Advanced branch prediction algorithm.
 - 4K-entry branch target array.
- New cache subsystem.
 - First level caches.
 - Advanced Execution Trace Cache stores decoded instructions.
 - Execution Trace Cache removes decoder latency from main execution loops.
 - Execution Trace Cache integrates path of program execution flow into a single line.
 - Low latency data cache.
 - Second level cache.
 - Full-speed, unified 8-way Level 2 on-die Advance Transfer Cache.
 - Bandwidth and performance increases with processor frequency.

1. Intel 64 and IA-32 processors based on the Intel NetBurst microarchitecture at 90 nm process can handle more than 24 stores in flight.

- High-performance, quad-pumped bus interface to the Intel NetBurst microarchitecture system bus.
 - Supports quad-pumped, scalable bus clock to achieve up to 4X effective speed.
 - Capable of delivering up to 8.5 GBytes of bandwidth per second.
- Superscalar issue to enable parallelism.
- Expanded hardware registers with renaming to avoid register name space limitations.
- 64-byte cache line size (transfers data up to two lines per sector).

Figure 2-2 is an overview of the Intel NetBurst microarchitecture. This microarchitecture pipeline is made up of three sections: (1) the front end pipeline, (2) the out-of-order execution core, and (3) the retirement unit.

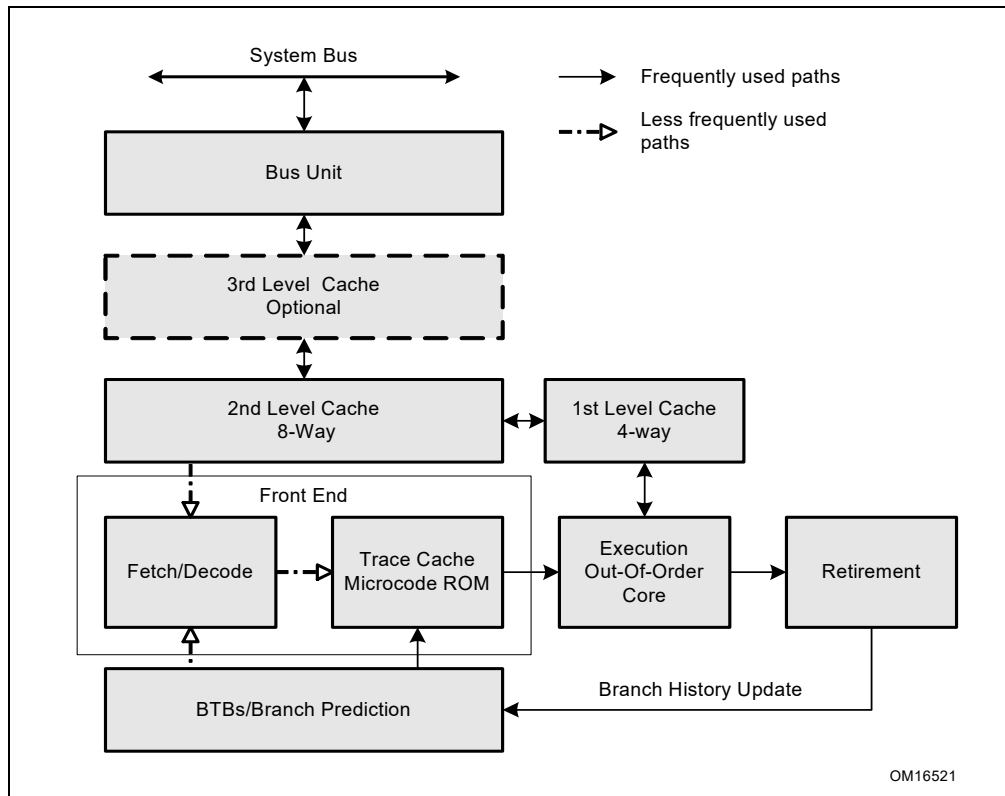


Figure 2-2. The Intel NetBurst® Microarchitecture

2.2.2.1 The Front End Pipeline

The front end supplies instructions in program order to the out-of-order execution core. It performs a number of functions:

- Prefetches instructions that are likely to be executed.
- Fetches instructions that have not already been prefetched.
- Decodes instructions into micro-operations.
- Generates microcode for complex instructions and special-purpose code.
- Delivers decoded instructions from the execution trace cache.
- Predicts branches using highly advanced algorithm.

The pipeline is designed to address common problems in high-speed, pipelined microprocessors. Two of these problems contribute to major sources of delays:

- Time to decode instructions fetched from the target.

- Wasted decode bandwidth due to branches or branch target in the middle of cache lines.

The operation of the pipeline's trace cache addresses these issues. Instructions are constantly being fetched and decoded by the translation engine (part of the fetch/decode logic) and built into sequences of micro-ops called traces. At any time, multiple traces (representing prefetched branches) are being stored in the trace cache. The trace cache is searched for the instruction that follows the active branch. If the instruction also appears as the first instruction in a pre-fetched branch, the fetch and decode of instructions from the memory hierarchy ceases and the pre-fetched branch becomes the new source of instructions (see Figure 2-2).

The trace cache and the translation engine have cooperating branch prediction hardware. Branch targets are predicted based on their linear addresses using branch target buffers (BTBs) and fetched as soon as possible.

2.2.2.2 Out-Of-Order Execution Core

The out-of-order execution core's ability to execute instructions out of order is a key factor in enabling parallelism. This feature enables the processor to reorder instructions so that if one micro-op is delayed, other micro-ops may proceed around it. The processor employs several buffers to smooth the flow of micro-ops.

The core is designed to facilitate parallel execution. It can dispatch up to six micro-ops per cycle (this exceeds trace cache and retirement micro-op bandwidth). Most pipelines can start executing a new micro-op every cycle, so several instructions can be in flight at a time for each pipeline. A number of arithmetic logical unit (ALU) instructions can start at two per cycle; many floating-point instructions can start once every two cycles.

2.2.2.3 Retirement Unit

The retirement unit receives the results of the executed micro-ops from the out-of-order execution core and processes the results so that the architectural state updates according to the original program order.

When a micro-op completes and writes its result, it is retired. Up to three micro-ops may be retired per cycle. The Reorder Buffer (ROB) is the unit in the processor which buffers completed micro-ops, updates the architectural state in order, and manages the ordering of exceptions. The retirement section also keeps track of branches and sends updated branch target information to the BTB. The BTB then purges pre-fetched traces that are no longer needed.

2.2.3 Intel® Core™ Microarchitecture

Intel Core microarchitecture introduces the following features that enable high performance and power-efficient performance for single-threaded as well as multi-threaded workloads:

- **Intel® Wide Dynamic Execution** enable each processor core to fetch, dispatch, execute in high bandwidths to support retirement of up to four instructions per cycle.
 - Fourteen-stage efficient pipeline.
 - Three arithmetic logical units.
 - Four decoders to decode up to five instruction per cycle.
 - Macro-fusion and micro-fusion to improve front-end throughput.
 - Peak issue rate of dispatching up to six micro-ops per cycle.
 - Peak retirement bandwidth of up to 4 micro-ops per cycle.
 - Advanced branch prediction.
 - Stack pointer tracker to improve efficiency of executing function/procedure entries and exits.
- **Intel® Advanced Smart Cache** delivers higher bandwidth from the second level cache to the core, and optimal performance and flexibility for single-threaded and multi-threaded applications.
 - Large second level cache up to 4 MB and 16-way associativity.
 - Optimized for multicore and single-threaded execution environments.
 - 256 bit internal data path to improve bandwidth from L2 to first-level data cache.

- **Intel® Smart Memory Access** prefetches data from memory in response to data access patterns and reduces cache-miss exposure of out-of-order execution.
 - Hardware prefetchers to reduce effective latency of second-level cache misses.
 - Hardware prefetchers to reduce effective latency of first-level data cache misses.
 - Memory disambiguation to improve efficiency of speculative execution engine.
- **Intel® Advanced Digital Media Boost** improves most 128-bit SIMD instruction with single-cycle throughput and floating-point operations.
 - Single-cycle throughput of most 128-bit SIMD instructions.
 - Up to eight floating-point operation per cycle.
 - Three issue ports available to dispatching SIMD instructions for execution.

Intel Core 2 Extreme, Intel Core 2 Duo processors and Intel Xeon processor 5100 series implement two processor cores based on the Intel Core microarchitecture, the functionality of the subsystems in each core are depicted in Figure 2-3.

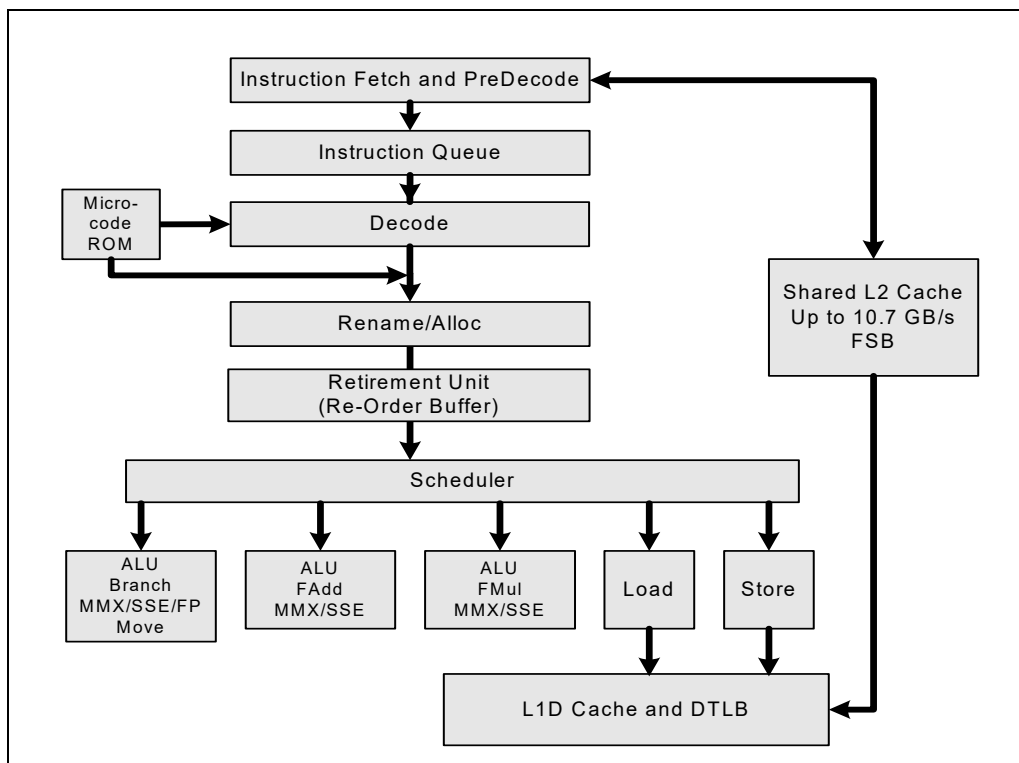


Figure 2-3. The Intel® Core™ Microarchitecture Pipeline Functionality

2.2.3.1 The Front End

The front end of Intel Core microarchitecture provides several enhancements to feed the Intel Wide Dynamic Execution engine:

- Instruction fetch unit prefetches instructions into an instruction queue to maintain steady supply of instruction to the decode units.
- Four-wide decode unit can decode 4 instructions per cycle or 5 instructions per cycle with Macrofusion.
- Macrofusion fuses common sequence of two instructions as one decoded instruction (micro-ops) to increase decoding throughput.
- Microfusion fuses common sequence of two micro-ops as one micro-ops to improve retirement throughput.

- Instruction queue provides caching of short loops to improve efficiency.
- Stack pointer tracker improves efficiency of executing procedure/function entries and exits.
- Branch prediction unit employs dedicated hardware to handle different types of branches for improved branch prediction.
- Advanced branch prediction algorithm directs instruction fetch unit to fetch instructions likely in the architectural code path for decoding.

2.2.3.2 Execution Core

The execution core of the Intel Core microarchitecture is superscalar and can process instructions out of order to increase the overall rate of instructions executed per cycle (IPC). The execution core employs the following feature to improve execution throughput and efficiency:

- Up to six micro-ops can be dispatched to execute per cycle.
- Up to four instructions can be retired per cycle.
- Three full arithmetic logical units.
- SIMD instructions can be dispatched through three issue ports.
- Most SIMD instructions have 1-cycle throughput (including 128-bit SIMD instructions).
- Up to eight floating-point operation per cycle.
- Many long-latency computation operation are pipelined in hardware to increase overall throughput.
- Reduced exposure to data access delays using Intel Smart Memory Access.

2.2.4 Intel Atom® Microarchitecture

Intel Atom microarchitecture maximizes power-efficient performance for single-threaded and multi-threaded workloads by providing:

- **Advanced Micro-Ops Execution**
 - Single-micro-op instruction execution from decode to retirement, including instructions with register-only, load, and store semantics.
 - Sixteen-stage, in-order pipeline optimized for throughput and reduced power consumption.
 - Dual pipelines to enable decode, issue, execution, and retirement of two instructions per cycle.
 - Advanced stack pointer to improve efficiency of executing function entry/returns.
- **Intel® Smart Cache**
 - Second level cache is 512 KB and 8-way associativity.
 - Optimized for multi-threaded and single-threaded execution environments
 - 256 bit internal data path between L2 and L1 data cache improves high bandwidth.
- **Efficient Memory Access**
 - Efficient hardware prefetchers to L1 and L2, speculatively loading data likely to be requested by processor to reduce cache miss impact.
- **Intel® Digital Media Boost**
 - Two issue ports for dispatching SIMD instructions to execution units.
 - Single-cycle throughput for most 128-bit integer SIMD instructions.
 - Up to six floating-point operations per cycle.
 - Up to two 128-bit SIMD integer operations per cycle.
 - Safe Instruction Recognition (SIR) to allow long-latency floating-point operations to retire out of order with respect to integer instructions.

2.2.5 Nehalem Microarchitecture

Nehalem microarchitecture provides the foundation for many features of Intel Core i7 processors. It builds on the success of 45 nm Intel Core microarchitecture and provides the following feature enhancements:

- **Enhanced processor core**
 - Improved branch prediction and recovery from misprediction.
 - Enhanced loop streaming to improve front end performance and reduce power consumption.
 - Deeper buffering in out-of-order engine to extract parallelism.
 - Enhanced execution units to provide acceleration in CRC, string/text processing and data shuffling.
- **Smart Memory Access**
 - Integrated memory controller provides low-latency access to system memory and scalable memory bandwidth.
 - New cache hierarchy organization with shared, inclusive L3 to reduce snoop traffic.
 - Two level TLBs and increased TLB size.
 - Fast unaligned memory access.
- **HyperThreading Technology**
 - Provides two hardware threads (logical processors) per core.
 - Takes advantage of 4-wide execution engine, large L3, and massive memory bandwidth.
- **Dedicated Power management Innovations**
 - Integrated microcontroller with optimized embedded firmware to manage power consumption.
 - Embedded real-time sensors for temperature, current, and power.
 - Integrated power gate to turn off/on per-core power consumption
 - Versatility to reduce power consumption of memory, link subsystems.

2.2.6 Sandy Bridge Microarchitecture

Sandy Bridge microarchitecture builds on the successes of Intel® Core™ microarchitecture and Nehalem microarchitecture. It offers the following features:

- Intel Advanced Vector Extensions (Intel AVX).
 - 256-bit floating-point instruction set extensions to the 128-bit Intel Streaming SIMD Extensions, providing up to 2X performance benefits relative to 128-bit code.
 - Non-destructive destination encoding offers more flexible coding techniques.
 - Supports flexible migration and co-existence between 256-bit AVX code, 128-bit AVX code and legacy 128-bit SSE code.
- Enhanced front-end and execution engine.
 - New decoded Icache component that improves front-end bandwidth and reduces branch misprediction penalty.
 - Advanced branch prediction.
 - Additional macro-fusion support.
 - Larger dynamic execution window.
 - Multi-precision integer arithmetic enhancements (ADC/SBB, MUL/IMUL).
 - LEA bandwidth improvement.
 - Reduction of general execution stalls (read ports, writeback conflicts, bypass latency, partial stalls).
 - Fast floating-point exception handling.

- XSAVE/XRSTORE performance improvements and XSAVEOPT new instruction.
- Cache hierarchy improvements for wider data path.
 - Doubling of bandwidth enabled by two symmetric ports for memory operation.
 - Simultaneous handling of more in-flight loads and stores enabled by increased buffers.
 - Internal bandwidth of two loads and one store each cycle.
 - Improved prefetching.
 - High bandwidth low latency LLC architecture.
 - High bandwidth ring architecture of on-die interconnect.

For additional information on Intel® Advanced Vector Extensions (AVX), see Section 5.13, “Intel® Advanced Vector Extensions (Intel® AVX)” and Chapter 14, “Programming with Intel® AVX, FMA, and Intel® AVX2” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

2.2.7 SIMD Instructions

Beginning with the Pentium II and Pentium with Intel MMX technology processor families, six extensions have been introduced into the Intel 64 and IA-32 architectures to perform single-instruction multiple-data (SIMD) operations. These extensions include the MMX technology, SSE extensions, SSE2 extensions, SSE3 extensions, Supplemental Streaming SIMD Extensions 3, and SSE4. Each of these extensions provides a group of instructions that perform SIMD operations on packed integer and/or packed floating-point data elements.

SIMD integer operations can use the 64-bit MMX or the 128-bit XMM registers. SIMD floating-point operations use 128-bit XMM registers. Figure 2-4 shows a summary of the various SIMD extensions (MMX technology, Intel SSE, Intel SSE2, Intel SSE3, SSSE3, and Intel SSE4), the data types they operate on, and how the data types are packed into MMX and XMM registers.

The Intel MMX technology was introduced in the Pentium II and Pentium with MMX technology processor families. MMX instructions perform SIMD operations on packed byte, word, or doubleword integers located in MMX registers. These instructions are useful in applications that operate on integer arrays and streams of integer data that lend themselves to SIMD processing.

Intel SSE was introduced in the Pentium III processor family. Intel SSE instructions operate on packed single precision floating-point values contained in XMM registers and on packed integers contained in MMX registers. Several Intel SSE instructions provide state management, cache control, and memory ordering operations. Other Intel SSE instructions are targeted at applications that operate on arrays of single precision floating-point data elements (3-D geometry, 3-D rendering, and video encoding and decoding applications).

Intel SSE2 was introduced in the Pentium 4 and Intel Xeon processors. Intel SSE2 instructions operate on packed double precision floating-point values contained in XMM registers and on packed integers contained in MMX and XMM registers. Intel SSE2 integer instructions extend IA-32 SIMD operations by adding new 128-bit SIMD integer operations and by expanding existing 64-bit SIMD integer operations to 128-bit XMM capability. Intel SSE2 instructions also provide new cache control and memory ordering operations.

Intel SSE3 was introduced with the Pentium 4 processor supporting Hyper-Threading Technology (built on 90 nm process technology). Intel SSE3 offers 13 instructions that accelerate performance of Streaming SIMD Extensions technology, Streaming SIMD Extensions 2 technology, and x87-FP math capabilities.

SSSE3 was introduced with the Intel Xeon processor 5100 series and Intel Core 2 processor family. SSSE3 offer 32 instructions to accelerate processing of SIMD integer data.

Intel SSE4 offers 54 instructions. 47 of them are referred to as Intel SSE4.1 instructions. Intel SSE4.1 was introduced with the Intel Xeon processor 5400 series and Intel Core 2 Extreme processor QX9650. The other seven Intel SSE4 instructions are referred to as Intel SSE4.2 instructions.

Intel AES-NI and PCLMULQDQ introduced seven new instructions. Six of them are primitives for accelerating algorithms based on AES encryption/decryption standard, and are referred to as Intel AES-NI.

The PCLMULQDQ instruction accelerates general-purpose block encryption, which can perform carry-less multiplication for two binary numbers up to 64-bit wide.

Intel 64 architecture allows four generations of 128-bit SIMD extensions to access up to 16 XMM registers. IA-32 architecture provides eight XMM registers.

Intel® Advanced Vector Extensions offers comprehensive architectural enhancements over previous generations of Streaming SIMD Extensions. Intel AVX introduces the following architectural enhancements:

- Support for 256-bit wide vectors and SIMD register set.
- 256-bit floating-point instruction set enhancement with up to 2X performance gain relative to 128-bit Streaming SIMD extensions.
- Instruction syntax support for generalized three-operand syntax to improve instruction programming flexibility and efficient encoding of new instruction extensions.
- Enhancement of legacy 128-bit SIMD instruction extensions to support three operand syntax and to simplify compiler vectorization of high-level language expressions.
- Support flexible deployment of 256-bit AVX code, 128-bit AVX code, legacy 128-bit code and scalar code.

In addition to performance considerations, programmers should also be cognizant of the implications of VEX-encoded AVX instructions with the expectations of system software components that manage the processor state components enabled by XCR0. For additional information see Section 2.3.10.1, "Vector Length Transition and Programming Considerations" in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

See also:

- Section 5.4, "MMX Instructions," and Chapter 9, "Programming with Intel® MMX™ Technology."
- Section 5.5, "Intel® SSE Instructions," and Chapter 10, "Programming with Intel® Streaming SIMD Extensions (Intel® SSE)."
- Section 5.6, "Intel® SSE2 Instructions," and Chapter 11, "Programming with Intel® Streaming SIMD Extensions 2 (Intel® SSE2)."
- Section 5.7, "Intel® SSE3 Instructions," Section 5.8, "Supplemental Streaming SIMD Extensions 3 (SSSE3) Instructions," Section 5.9, "Intel® SSE4 Instructions," and Chapter 12, "Programming with Intel® SSE3, SSSE3, Intel® SSE4, and Intel® AES-NI."

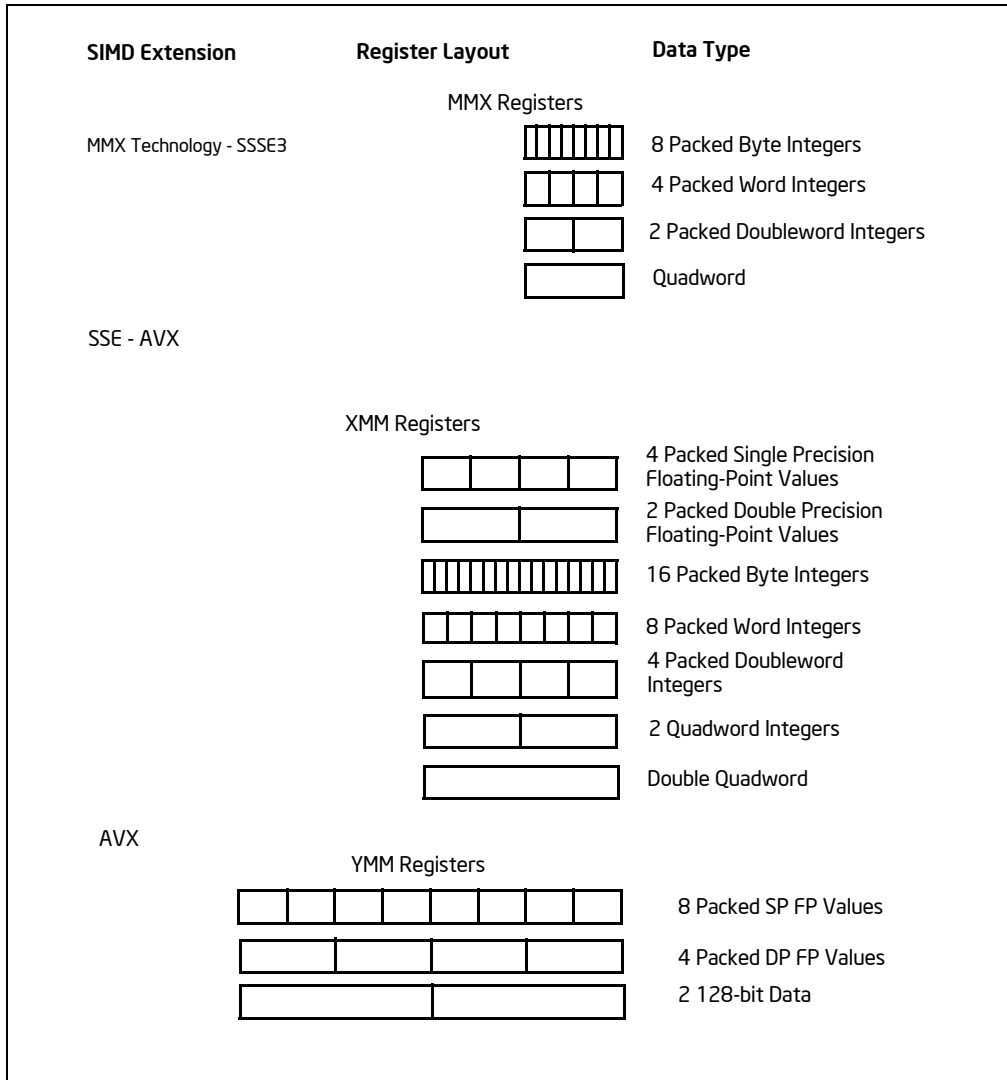


Figure 2-4. SIMD Extensions, Register Layouts, and Data Types

2.2.8 Intel® Hyper-Threading Technology

Intel Hyper-Threading Technology (Intel HT Technology) was developed to improve the performance of IA-32 processors when executing multi-threaded operating system and application code or single-threaded applications under multi-tasking environments. The technology enables a single physical processor to execute two or more separate code streams (threads) concurrently using shared execution resources.

Intel HT Technology is one form of hardware multi-threading capability in IA-32 processor families. It differs from multi-processor capability using separate physically distinct packages with each physical processor package mated with a physical socket. Intel HT Technology provides hardware multi-threading capability with a single physical package by using shared execution resources in a processor core.

Architecturally, an IA-32 processor that supports Intel HT Technology consists of two or more logical processors, each of which has its own IA-32 architectural state. Each logical processor consists of a full set of IA-32 data registers, segment registers, control registers, debug registers, and most of the MSRs. Each also has its own advanced programmable interrupt controller (APIC).

Figure 2-5 shows a comparison of a processor that supports Intel HT Technology (implemented with two logical processors) and a traditional dual processor system.

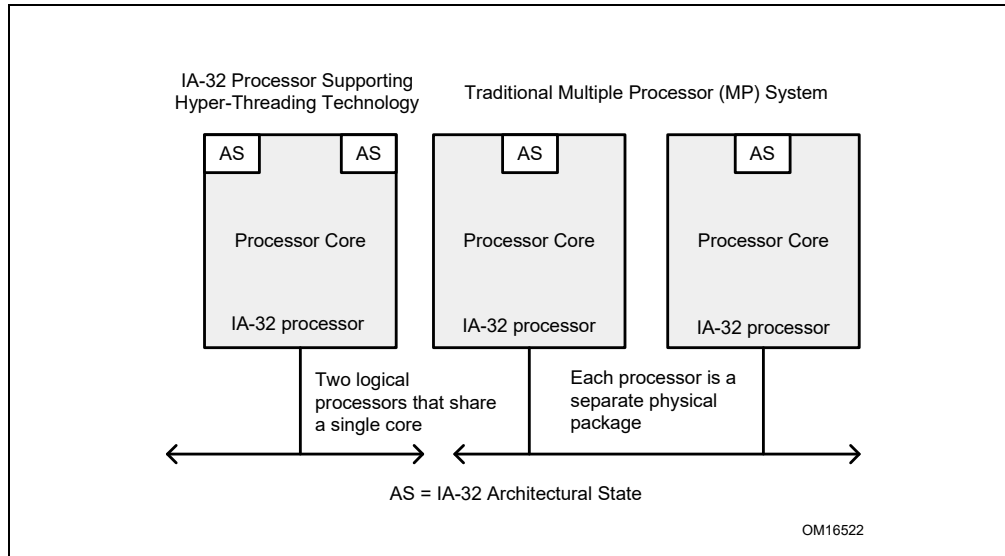


Figure 2-5. Comparison of an IA-32 Processor Supporting Intel® Hyper-Threading Technology and a Traditional Dual Processor System

Unlike a traditional MP system configuration that uses two or more separate physical IA-32 processors, the logical processors in an IA-32 processor supporting Intel HT Technology share the core resources of the physical processor. This includes the execution engine and the system bus interface. After power up and initialization, each logical processor can be independently directed to execute a specified thread, interrupted, or halted.

Intel HT Technology leverages the process and thread-level parallelism found in contemporary operating systems and high-performance applications by providing two or more logical processors on a single chip. This configuration allows two or more threads¹ to be executed simultaneously on each a physical processor. Each logical processor executes instructions from an application thread using the resources in the processor core. The core executes these threads concurrently, using out-of-order instruction scheduling to maximize the use of execution units during each clock cycle.

2.2.8.1 Some Implementation Notes

All Intel HT Technology configurations require:

- A processor that supports Intel HT Technology.
- A chipset and BIOS that utilize the technology.
- Operating system optimizations.

See http://www.intel.com/products/ht/hyperthreading_more.htm for information.

At the firmware (BIOS) level, the basic procedures to initialize the logical processors in a processor supporting Intel HT Technology are the same as those for a traditional DP or MP platform. The mechanisms that are described in the *Multiprocessor Specification, Version 1.4* to power-up and initialize physical processors in an MP system also apply to logical processors in a processor that supports Intel HT Technology.

An operating system designed to run on a traditional DP or MP platform may use CPUID to determine the presence of hardware multi-threading support feature and the number of logical processors they provide.

Although existing operating system and application code should run correctly on a processor that supports Intel HT Technology, some code modifications are recommended to get the optimum benefit. These modifications are discussed in Chapter 7, "Multiple-Processor Management," Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

1. In the remainder of this document, the term "thread" will be used as a general term for the terms "process" and "thread."

2.2.9 Multi-Core Technology

Multi-core technology is another form of hardware multi-threading capability in IA-32 processor families. Multi-core technology enhances hardware multi-threading capability by providing two or more execution cores in a physical package.

The Intel Pentium processor Extreme Edition is the first member in the IA-32 processor family to introduce multi-core technology. The processor provides hardware multi-threading support with both two processor cores and Intel Hyper-Threading Technology. This means that the Intel Pentium processor Extreme Edition provides four logical processors in a physical package (two logical processors for each processor core). The Dual-Core Intel Xeon processor features multi-core, Intel Hyper-Threading Technology and supports multi-processor platforms.

The Intel Pentium D processor also features multi-core technology. This processor provides hardware multi-threading support with two processor cores but does not offer Intel Hyper-Threading Technology. This means that the Intel Pentium D processor provides two logical processors in a physical package, with each logical processor owning the complete execution resources of a processor core.

The Intel Core 2 processor family, Intel Xeon processor 3000 series, Intel Xeon processor 5100 series, and Intel Core Duo processor offer power-efficient multi-core technology. The processor contains two cores that share a smart second level cache. The Level 2 cache enables efficient data sharing between two cores to reduce memory traffic to the system bus.

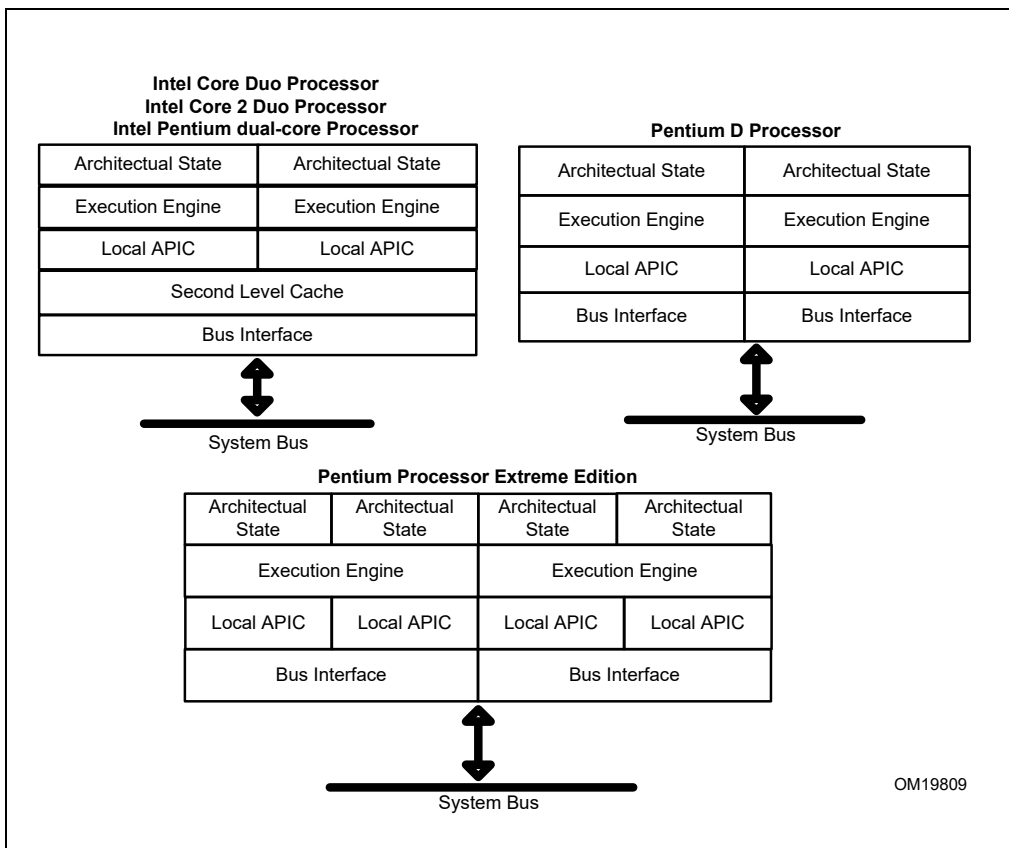


Figure 2-6. Intel 64 and IA-32 Processors that Support Dual-Core

The Pentium® dual-core processor is based on the same technology as the Intel Core 2 Duo processor family. The Intel Xeon processor 7300, 5300, and 3200 series, Intel Core 2 Extreme Quad-Core processor, and Intel Core 2 Quad processors support Intel quad-core technology. The Quad-core Intel Xeon processors and the Quad-Core Intel Core 2 processor family are also in Figure 2-7.

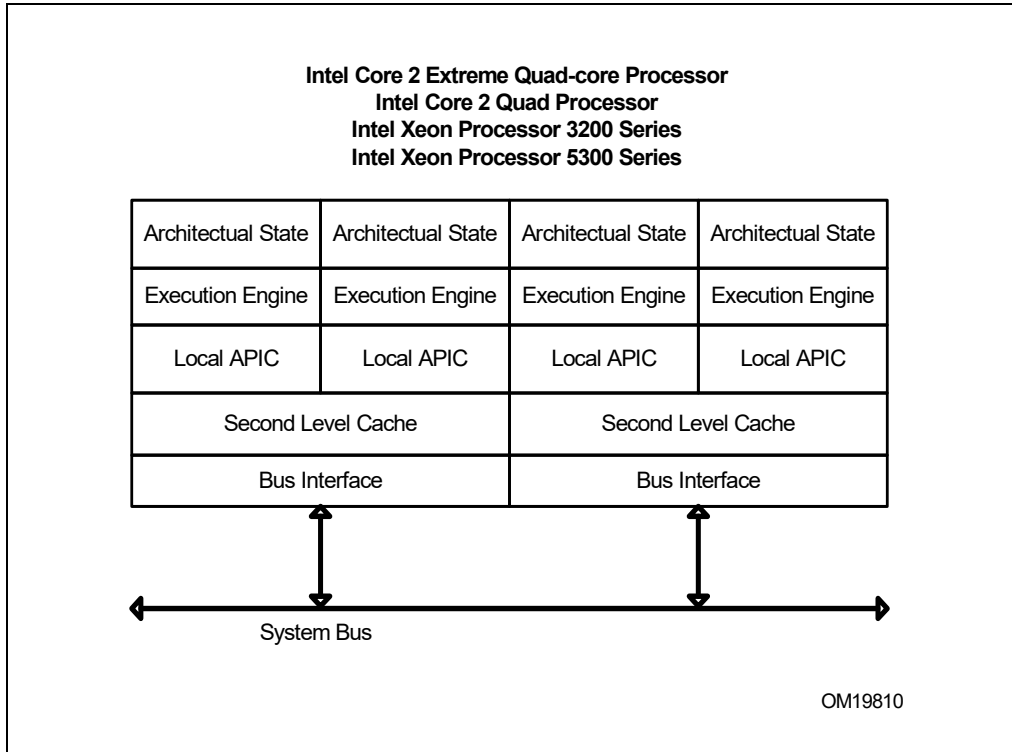


Figure 2-7. Intel® 64 Processors that Support Quad-Core

Intel Core i7 processors support Intel quad-core technology, Intel HyperThreading Technology, provides Intel QuickPath interconnect link to the chipset and have integrated memory controller supporting three channel to DDR3 memory.

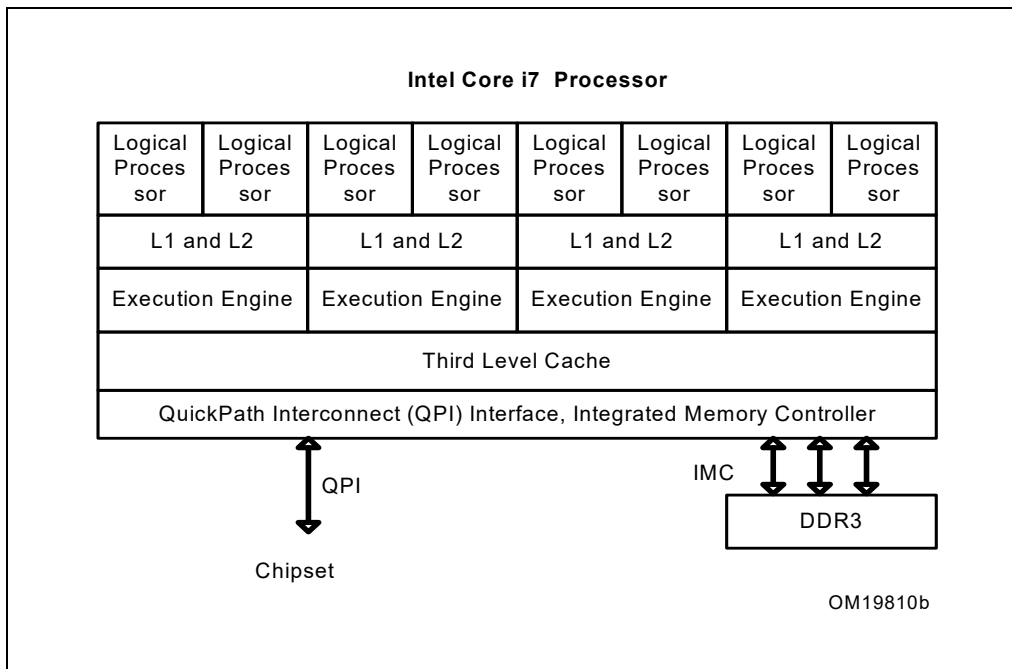


Figure 2-8. Intel® Core™ i7 Processor

2.2.10 Intel® 64 Architecture

Intel 64 architecture increases the linear address space for software to 64 bits and supports physical address space up to 52 bits. The technology also introduces a new operating mode referred to as IA-32e mode.

IA-32e mode operates in one of two sub-modes: (1) compatibility mode enables a 64-bit operating system to run most legacy 32-bit software unmodified, (2) 64-bit mode enables a 64-bit operating system to run applications written to access 64-bit address space.

In the 64-bit mode, applications may access:

- 64-bit flat linear addressing.
- 8 additional general-purpose registers (GPRs).
- 8 additional registers for streaming SIMD extensions (Intel SSE, SSE2, and SSE3, and SSSE3).
- 64-bit-wide GPRs and instruction pointers.
- uniform byte-register addressing.
- fast interrupt-prioritization mechanism.
- a new instruction-pointer relative-addressing mode.

An Intel 64 architecture processor supports existing IA-32 software because it is able to run all non-64-bit legacy modes supported by IA-32 architecture. Most existing IA-32 applications also run in compatibility mode.

2.2.11 Intel® Virtualization Technology (Intel® VT)

Intel® Virtualization Technology for Intel 64 and IA-32 architectures provide extensions that support virtualization. The extensions are referred to as Virtual Machine Extensions (VMX). An Intel 64 or IA-32 platform with VMX can function as multiple virtual systems (or virtual machines). Each virtual machine can run operating systems and applications in separate partitions.

VMX also provides programming interface for a new layer of system software (called the Virtual Machine Monitor (VMM)) used to manage the operation of virtual machines. Information on VMX and on the programming of VMMs is in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

Intel Core i7 processor provides the following enhancements to Intel Virtualization Technology:

- Virtual processor ID (VPID) to reduce the cost of VMM managing transitions.
- Extended page table (EPT) to reduce the number of transitions for VMM to manage memory virtualization.
- Reduced latency of VM transitions.

2.3 INTEL® 64 AND IA-32 PROCESSOR GENERATIONS

In the mid-1960s, Intel co-founder and Chairman Emeritus Gordon Moore had this observation: "... the number of transistors that would be incorporated on a silicon die would double every 18 months for the next several years." Over the past three and half decades, this prediction known as "Moore's Law" has continued to hold true.

The computing power and the complexity (or roughly, the number of transistors per processor) of Intel architecture processors has grown in close relation to Moore's law. By taking advantage of new process technology and new microarchitecture designs, each new generation of IA-32 processors has demonstrated frequency-scaling headroom and new performance levels over the previous generation processors.

The key features of the Intel Pentium 4 processor, Intel Xeon processor, Intel Xeon processor MP, Pentium III processor, and Pentium III Xeon processor with advanced transfer cache are shown in Table 2-1. Older generation IA-32 processors, which do not employ on-die Level 2 cache, are shown in Table 2-2.

Table 2-1. Key Features of Most Recent IA-32 Processors

Intel Processor	Date Introduced	Micro-architecture	Top-Bin Clock Frequency at Introduction	Transistors	Register Sizes ¹	System Bus Bandwidth	Max. Extern. Addr. Space	On-Die Caches ²
Intel Pentium M Processor 755 ³	2004	Intel Pentium M Processor	2.00 GHz	140 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	4 GB	L1: 64 KB L2: 2 MB
Intel Core Duo Processor T2600 ³	2006	Improved Intel Pentium M Processor Microarchitecture; Dual Core; Intel Smart Cache, Advanced Thermal Manager	2.16 GHz	152M	GP: 32 FPU: 80 MMX: 64 XMM: 128	5.3 GB/s	4 GB	L1: 64 KB L2: 2 MB (2MB Total)
Intel Atom Processor Z5xx series	2008	Intel Atom Microarchitecture; Intel Virtualization Technology.	1.86 GHz - 800 MHz	47M	GP: 32 FPU: 80 MMX: 64 XMM: 128	Up to 4.2 GB/s	4 GB	L1: 56 KB ⁴ L2: 512KB

NOTES:

1. The register size and external data bus size are given in bits.
2. First level cache is denoted using the abbreviation L1, 2nd level cache is denoted as L2. The size of L1 includes the first-level data cache and the instruction cache where applicable, but does not include the trace cache.
3. Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families.
See http://www.intel.com/products/processor_number for details.
4. In Intel Atom Processor, the size of L1 instruction cache is 32 KBytes, L1 data cache is 24 KBytes.

Table 2-2. Key Features of Most Recent Intel® 64 Processors

Intel Processor	Date Introduced	Micro-architecture	Highest Processor Base Frequency at Introduction	Transistors	Register Sizes	System Bus/QPI Link Speed	Max. Extern. Addr. Space	On-Die Caches
64-bit Intel Xeon Processor with 800 MHz System Bus	2004	Intel NetBurst Microarchitecture; Intel Hyper-Threading Technology; Intel 64 Architecture	3.60 GHz	125 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 1 MB L2
64-bit Intel Xeon Processor MP with 8MB L3	2005	Intel NetBurst Microarchitecture; Intel Hyper-Threading Technology; Intel 64 Architecture	3.33 GHz	675M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	5.3 GB/s ¹	1024 GB (1 TB)	12K μ op Execution Trace Cache; 16 KB L1; 1 MB L2, 8 MB L3

Table 2-2. Key Features of Most Recent Intel® 64 Processors (Contd.)

Intel Processor	Date Introduced	Micro-architecture	Highest Processor Base Frequency at Introduction	Transistors	Register Sizes	System Bus/QPI Link Speed	Max. Extern. Addr. Space	On-Die Caches
Intel Pentium 4 Processor Extreme Edition Supporting Hyper-Threading Technology	2005	Intel NetBurst Microarchitecture; Intel Hyper-Threading Technology; Intel 64 Architecture	3.73 GHz	164 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 2 MB L2
Intel Pentium Processor Extreme Edition 840	2005	Intel NetBurst Microarchitecture; Intel Hyper-Threading Technology; Intel 64 Architecture; Dual-core ²	3.20 GHz	230 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 1MB L2 (2MB Total)
Dual-Core Intel Xeon Processor 7041	2005	Intel NetBurst Microarchitecture; Intel Hyper-Threading Technology; Intel 64 Architecture; Dual-core ³	3.00 GHz	321M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 2MB L2 (4MB Total)
Intel Pentium 4 Processor 672	2005	Intel NetBurst Microarchitecture; Intel Hyper-Threading Technology; Intel 64 Architecture; Intel Virtualization Technology.	3.80 GHz	164 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 2MB L2
Intel Pentium Processor Extreme Edition 955	2006	Intel NetBurst Microarchitecture; Intel 64 Architecture; Dual Core; Intel Virtualization Technology.	3.46 GHz	376M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 2MB L2 (4MB Total)
Intel Core 2 Extreme Processor X6800	2006	Intel Core Microarchitecture; Dual Core; Intel 64 Architecture; Intel Virtualization Technology.	2.93 GHz	291M	GP: 32,64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	64 GB	L1: 64 KB L2: 4MB (4MB Total)

Table 2-2. Key Features of Most Recent Intel® 64 Processors (Contd.)

Intel Processor	Date Introduced	Micro-architecture	Highest Processor Base Frequency at Introduction	Transistors	Register Sizes	System Bus/QPI Link Speed	Max. Extern. Addr. Space	On-Die Caches
Intel Xeon Processor 5160	2006	Intel Core Microarchitecture; Dual Core; Intel 64 Architecture; Intel Virtualization Technology.	3.00 GHz	291M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	10.6 GB/s	64 GB	L1: 64 KB L2: 4MB (4MB Total)
Intel Xeon Processor 7140	2006	Intel NetBurst Microarchitecture; Dual Core; Intel 64 Architecture; Intel Virtualization Technology.	3.40 GHz	1.3 B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	12.8 GB/s	64 GB	L1: 64 KB L2: 1MB (2MB Total) L3: 16 MB (16MB Total)
Intel Core 2 Extreme Processor QX6700	2006	Intel Core Microarchitecture; Quad Core; Intel 64 Architecture; Intel Virtualization Technology.	2.66 GHz	582M	GP: 32,64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	64 GB	L1: 64 KB L2: 4MB (4MB Total)
Quad-core Intel Xeon Processor 5355	2006	Intel Core Microarchitecture; Quad Core; Intel 64 Architecture; Intel Virtualization Technology.	2.66 GHz	582 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	10.6 GB/s	256 GB	L1: 64 KB L2: 4MB (8 MB Total)
Intel Core 2 Duo Processor E6850	2007	Intel Core Microarchitecture; Dual Core; Intel 64 Architecture; Intel Virtualization Technology; Intel Trusted Execution Technology	3.00 GHz	291 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	10.6 GB/s	64 GB	L1: 64 KB L2: 4MB (4MB Total)
Intel Xeon Processor 7350	2007	Intel Core Microarchitecture; Quad Core; Intel 64 Architecture; Intel Virtualization Technology.	2.93 GHz	582 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	1024 GB	L1: 64 KB L2: 4MB (8MB Total)

Table 2-2. Key Features of Most Recent Intel® 64 Processors (Contd.)

Intel Processor	Date Introduced	Micro-architecture	Highest Processor Base Frequency at Introduction	Transistors	Register Sizes	System Bus/QPI Link Speed	Max. Extern. Addr. Space	On-Die Caches
Intel Xeon Processor 5472	2007	Enhanced Intel Core Microarchitecture; Quad Core; Intel 64 Architecture; Intel Virtualization Technology.	3.00 GHz	820 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	12.8 GB/s	256 GB	L1: 64 KB L2: 6MB (12MB Total)
Intel Atom Processor	2008	Intel Atom Microarchitecture; Intel 64 Architecture; Intel Virtualization Technology.	2.0 - 1.60 GHz	47 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	Up to 4.2 GB/s	Up to 64GB	L1: 56 KB ⁴ L2: 512KB
Intel Xeon Processor 7460	2008	Enhanced Intel Core Microarchitecture; Six Cores; Intel 64 Architecture; Intel Virtualization Technology.	2.67 GHz	1.9 B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	1024 GB	L1: 64 KB L2: 3MB (9MB Total) L3: 16MB
Intel Atom Processor 330	2008	Intel Atom Microarchitecture; Intel 64 Architecture; Dual core; Intel Virtualization Technology.	1.60 GHz	94 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	Up to 4.2 GB/s	Up to 64GB	L1: 56 KB ⁵ L2: 512KB (1MB Total)
Intel Core i7-965 Processor Extreme Edition	2008	Nehalem microarchitecture; Quadcore; HyperThreading Technology; Intel QPI; Intel 64 Architecture; Intel Virtualization Technology.	3.20 GHz	731 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; Memory: 25 GB/s	64 GB	L1: 64 KB L2: 256KB L3: 8MB

Table 2-2. Key Features of Most Recent Intel® 64 Processors (Contd.)

Intel Processor	Date Introduced	Micro-architecture	Highest Processor Base Frequency at Introduction	Transistors	Register Sizes	System Bus/QPI Link Speed	Max. Extern. Addr. Space	On-Die Caches
Intel Core i7-620M Processor	2010	Intel Turbo Boost Technology, Westmere microarchitecture; Dual-core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology., Integrated graphics	2.66 GHz	383 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128		64 GB	L1: 64 KB L2: 256KB L3: 4MB
Intel Xeon-Processor 5680	2010	Intel Turbo Boost Technology, Westmere microarchitecture; Six core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology.	3.33 GHz	1.1B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; 32 GB/s	1 TB	L1: 64 KB L2: 256KB L3: 12MB
Intel Xeon-Processor 7560	2010	Intel Turbo Boost Technology, Nehalem microarchitecture; Eight core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology.	2.26 GHz	2.3B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; Memory: 76 GB/s	16 TB	L1: 64 KB L2: 256KB L3: 24MB
Intel Core i7-2600K Processor	2011	Intel Turbo Boost Technology, Sandy Bridge microarchitecture; Four core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology., Processor graphics, Quicksync Video	3.40 GHz	995M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128 YMM: 256	DMI: 5 GT/s; Memory: 21 GB/s	64 GB	L1: 64 KB L2: 256KB L3: 8MB

Table 2-2. Key Features of Most Recent Intel® 64 Processors (Contd.)

Intel Processor	Date Introduced	Micro-architecture	Highest Processor Base Frequency at Introduction	Transistors	Register Sizes	System Bus/QPI Link Speed	Max. Extern. Addr. Space	On-Die Caches
Intel Xeon-Processor E3-1280	2011	Intel Turbo Boost Technology, Sandy Bridge microarchitecture; Four core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology.	3.50 GHz		GP: 32, 64 FPU: 80 MMX: 64 XMM: 128 YMM: 256	DMI: 5 GT/s; Memory: 21 GB/s	1 TB	L1: 64 KB L2: 256KB L3: 8MB
Intel Xeon-Processor E7-8870	2011	Intel Turbo Boost Technology, Westmere microarchitecture; Ten core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology.	2.40 GHz	2.2B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; Memory: 102 GB/s	16 TB	L1: 64 KB L2: 256KB L3: 30MB

NOTES:

1. The 64-bit Intel Xeon Processor MP with an 8-MByte L3 supports a multi-processor platform with a dual system bus; this creates a platform bandwidth with 10.6 GBytes.
2. In Intel Pentium Processor Extreme Edition 840, the size of on-die cache is listed for each core. The total size of L2 in the physical package is 2 MBytes.
3. In Dual-Core Intel Xeon Processor 7041, the size of on-die cache is listed for each core. The total size of L2 in the physical package is 4 MBytes.
4. In Intel Atom Processor, the size of L1 instruction cache is 32 KBytes, L1 data cache is 24 KBytes.
5. In Intel Atom Processor, the size of L1 instruction cache is 32 KBytes, L1 data cache is 24 KBytes.

Table 2-3. Key Features of Previous Generations of IA-32 Processors

Intel Processor	Date Introduced	Max. Clock Frequency/ Technology at Introduction	Transistors	Register Sizes ¹	Ext. Data Bus Size ²	Max. Extern. Addr. Space	Caches
8086	1978	8 MHz	29 K	16 GP	16	1 MB	None
Intel 286	1982	12.5 MHz	134 K	16 GP	16	16 MB	Note 3
Intel386 DX Processor	1985	20 MHz	275 K	32 GP	32	4 GB	Note 3
Intel486 DX Processor	1989	25 MHz	1.2 M	32 GP 80 FPU	32	4 GB	L1: 8 KB
Pentium Processor	1993	60 MHz	3.1 M	32 GP 80 FPU	64	4 GB	L1:16 KB
Pentium Pro Processor	1995	200 MHz	5.5 M	32 GP 80 FPU	64	64 GB	L1: 16 KB L2: 256 KB or 512 KB
Pentium II Processor	1997	266 MHz	7 M	32 GP 80 FPU 64 MMX	64	64 GB	L1: 32 KB L2: 256 KB or 512 KB
Pentium III Processor	1999	500 MHz	8.2 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32 KB L2: 512 KB
Pentium III and Pentium III Xeon Processors	1999	700 MHz	28 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32 KB L2: 256 KB
Pentium 4 Processor	2000	1.50 GHz, Intel NetBurst Microarchitecture	42 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	12K μ op Execution Trace Cache; L1: 8KB L2: 256 KB
Intel Xeon Processor	2001	1.70 GHz, Intel NetBurst Microarchitecture	42 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	12K μ op Execution Trace Cache; L1: 8KB L2: 512KB
Intel Xeon Processor	2002	2.20 GHz, Intel NetBurst Microarchitecture, HyperThreading Technology	55 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	12K μ op Execution Trace Cache; L1: 8KB L2: 512KB
Pentium M Processor	2003	1.60 GHz, Intel NetBurst Microarchitecture	77 M	32 GP 80 FPU 64 MMX 128 XMM	64	4 GB	L1: 64KB L2: 1 MB

Table 2-3. Key Features of Previous Generations of IA-32 Processors (Contd.)

Intel Pentium 4 Processor Supporting Hyper-Threading Technology at 90 nm process	2004	3.40 GHz, Intel NetBurst Microarchitecture, HyperThreading Technology	125 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	12K μ op Execution Trace Cache; L1: 16KB L2: 1 MB
--	------	---	-------	--------------------------------------	----	-------	---

NOTE:

1. The register size and external data bus size are given in bits. Note also that each 32-bit general-purpose (GP) registers can be addressed as an 8- or a 16-bit data registers in all of the processors.
2. Internal data paths are 2 to 4 times wider than the external data bus for each processor.

2.4 PLANNED REMOVAL OF INTEL® INSTRUCTION SET ARCHITECTURE AND FEATURES FROM UPCOMING PRODUCTS

This section lists Intel Instruction Set Architecture (ISA) and features that Intel plans to remove from select products starting from a specific year.

Table 2-4. Planned Intel® ISA and Features Removal List

Intel ISA/Feature	Year of Removal
xAPIC mode	2025 onwards

2.5 INTEL® INSTRUCTION SET ARCHITECTURE AND FEATURES REMOVED

This section lists Intel ISA and features that Intel has already removed for select upcoming products. All sections relevant to the removed features will be identified as such and may be moved to an archived section in future Intel® 64 and IA-32 Architectures Software Developer's Manual releases.

Table 2-5. Intel® ISA and Features Removal List

Intel ISA/Feature	Year of Removal
Intel® Memory Protection Extensions (Intel® MPX)	2019 onwards
MSR_TEST_CTRL, bit 31 (MSR address 33H)	2019 onwards
Hardware Lock Elision (HLE)	2019 onwards

2. Updates to Chapter 3, Volume 2A

Change bars and green text show changes to Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L*.

Changes to this chapter:

- Updated the Flags Affected section for the BZHI instruction.
- Updated the CPUID instruction to include updates to wording in leaf 0BH, leaf 1AH, and leaf 1FH, as well as add support for bus-lock detection and the ENCLU instruction leaf EDECCSSA. Updated DisplayFamily rule to remove shift that is unnecessary. Updated third paragraph under "INPUT EAX = 1BH: Returns PCONFIG Information."
- Updated the ENQCMD instruction to emphasize that the user bit is always 0 regardless of the current mode.
- Updated a table in each of the FPREM and FPREM1 instructions to update cells that erroneously indicated a #Z exception; those cases raise an #IA exception.

BZHI—Zero High Bits Starting with Specified Bit Position

Opcode/Instruction	Op/En	64/32-bit Mode	CPUID Feature Flag	Description
VEX.LZ.OF38.W0 F5 /r BZHI r32a, r/m32, r32b	RMV	V/V	BMI2	Zero bits in r/m32 starting with the position in r32b, write result to r32a.
VEX.LZ.OF38.W1 F5 /r BZHI r64a, r/m64, r64b	RMV	V/N.E.	BMI2	Zero bits in r/m64 starting with the position in r64b, write result to r64a.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMV	ModRM:reg (w)	ModRM:r/m (r)	VEX.vvvv (r)	N/A

Description

BZHI copies the bits of the first source operand (the second operand) into the destination operand (the first operand) and clears the higher bits in the destination according to the INDEX value specified by the second source operand (the third operand). The INDEX is specified by bits 7:0 of the second source operand. The INDEX value is saturated at the value of OperandSize - 1. CF is set, if the number contained in the 8 low bits of the third operand is greater than OperandSize - 1.

This instruction is not supported in real mode and virtual-8086 mode. The operand size is always 32 bits if not in 64-bit mode. In 64-bit mode operand size 64 requires VEX.W1. VEX.W1 is ignored in non-64-bit modes. An attempt to execute this instruction with VEX.L not equal to 0 will cause #UD.

Operation

```
N := SRC2[7:0]
DEST := SRC1
IF (N < OperandSize)
    DEST[OperandSize-1:N] := 0
FI
IF (N > OperandSize - 1)
    CF := 1
ELSE
    CF := 0
FI
```

Flags Affected

ZF and SF flags are updated based on the result. CF flag is set as specified in the Operation section. OF flag is cleared. AF and PF flags are undefined.

Intel C/C++ Compiler Intrinsic Equivalent

```
BZHI unsigned __int32 _bzhi_u32(unsigned __int32 src, unsigned __int32 index);
BZHI unsigned __int64 _bzhi_u64(unsigned __int64 src, unsigned __int32 index);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Table 2-29, "Type 13 Class Exception Conditions."

CPUID—CPU Identification

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F A2	CPUID	Z0	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, as determined by input entered in EAX (in some cases, ECX as well).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. This instruction operates the same in non-64-bit modes and 64-bit mode.

CPUID returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers.¹ The instruction's output is dependent on the contents of the EAX register upon execution (in some cases, ECX as well). For example, the following pseudocode loads EAX with 00H and causes CPUID to return a Maximum Return Value and the Vendor Identification String in the appropriate registers:

```
MOV EAX, 00H
CPUID
```

Table 3-8 shows information returned, depending on the initial value loaded into the EAX register.

Two types of information are returned: basic and extended function information. If a value entered for CPUID.EAX is higher than the maximum input value for basic or extended function for that processor then the data for the highest basic information leaf is returned. For example, using some Intel processors, the following is true:

```
CPUID.EAX = 05H (* Returns MONITOR/MWAIT leaf. *)
CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)
CPUID.EAX = 0BH (* Returns Extended Topology Enumeration leaf. *)2
CPUID.EAX = 1FH (* Returns V2 Extended Topology Enumeration leaf. *)2
CPUID.EAX = 80000008H (* Returns linear/physical address size data. *)
CPUID.EAX = 8000000AH (* INVALID: Returns same information as CPUID.EAX = 0BH. *)
```

If a value entered for CPUID.EAX is less than or equal to the maximum input value and the leaf is not supported on that processor then 0 is returned in all the registers.

When CPUID returns the highest basic leaf information as a result of an invalid input EAX value, any dependence on input ECX value in the basic leaf is honored.

CPUID can be executed at any privilege level to serialize instruction execution. Serializing instruction execution guarantees that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed.

See also:

"Serializing Instructions" in Chapter 9, "Multiple-Processor Management," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

"Caching Translation Information" in Chapter 4, "Paging," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

1. On Intel 64 processors, CPUID clears the high 32 bits of the RAX/RBX/RCX/RDX registers in all modes.
2. CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of CPUID leaf 1FH before using leaf 0BH.

Table 3-8. Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
<i>Basic CPUID Information</i>		
0H	EAX	Maximum Input Value for Basic CPUID Information.
	EBX	"Genu"
	ECX	"ntel"
	EDX	"inel"
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-6).
	EBX	Bits 07-00: Brand Index. Bits 15-08: CLFLUSH line size (Value * 8 = cache line size in bytes; used also by CLFLUSHOPT). Bits 23-16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31-24: Initial APIC ID**.
	ECX	Feature Information (see Figure 3-7 and Table 3-10).
	EDX	Feature Information (see Figure 3-8 and Table 3-11).
		NOTES: * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.1.EDX.HTT[bit 28]= 1. ** The 8-bit initial APIC ID in EBX[31:24] is replaced by the 32-bit x2APIC ID, available in Leaf 0BH and Leaf 1FH.
02H	EAX	Cache and TLB Information (see Table 3-12).
	EBX	Cache and TLB Information.
	ECX	Cache and TLB Information.
	EDX	Cache and TLB Information.
03H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 00-31 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32-63 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature.
CPUID leaves above 2 and below 80000000H are visible only when IA32_MISC_ENABLE[bit 22] has its default value of 0.		
<i>Deterministic Cache Parameters Leaf</i>		
04H		NOTES: Leaf 04H output depends on the initial value in ECX.* See also: "INPUT EAX = 04H: Returns Deterministic Cache Parameters for Each Level" on page 250.
	EAX	Bits 04-00: Cache Type Field. 0 = Null - No more caches. 1 = Data Cache. 2 = Instruction Cache. 3 = Unified Cache. 4-31 = Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		<p>Bits 07-05: Cache Level (starts at 1). Bit 08: Self Initializing cache level (does not need SW initialization). Bit 09: Fully Associative cache.</p> <p>Bits 13-10: Reserved. Bits 25-14: Maximum number of addressable IDs for logical processors sharing this cache**, ***. Bits 31-26: Maximum number of addressable IDs for processor cores in the physical package**, ****, *****.</p> <p>EBX Bits 11-00: L = System Coherency Line Size**. Bits 21-12: P = Physical Line partitions**. Bits 31-22: W = Ways of associativity**.</p> <p>ECX Bits 31-00: S = Number of Sets**.</p> <p>EDX Bit 00: Write-Back Invalidate/Invalidate. 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 01: Cache Inclusiveness. 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 02: Complex Cache Indexing. 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31-03: Reserved = 0.</p> <p>NOTES:</p> <p>* If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EAX[4:0] as 0.</p> <p>** Add one to the return value to get the result.</p> <p>***The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache.</p> <p>**** The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID.</p> <p>***** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.</p>
	<i>MONITOR/MWAIT Leaf</i>	
05H	EAX	<p>Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity). Bits 31-16: Reserved = 0.</p> <p>EBX Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity). Bits 31-16: Reserved = 0.</p> <p>ECX Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported. Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled. Bits 31-02: Reserved.</p> <p>EDX Bits 03-00: Number of C0* sub C-states supported using MWAIT. Bits 07-04: Number of C1* sub C-states supported using MWAIT. Bits 11-08: Number of C2* sub C-states supported using MWAIT. Bits 15-12: Number of C3* sub C-states supported using MWAIT. Bits 19-16: Number of C4* sub C-states supported using MWAIT. Bits 23-20: Number of C5* sub C-states supported using MWAIT. Bits 27-24: Number of C6* sub C-states supported using MWAIT. Bits 31-28: Number of C7* sub C-states supported using MWAIT.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	<p>NOTE: * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.</p>	
<i>Thermal and Power Management Leaf</i>		
06H	EAX	<p>Bit 00: Digital temperature sensor is supported if set. Bit 01: Intel Turbo Boost Technology available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved. Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bit 07: HWP. HWP base registers (IA32_PM_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS) are supported if set. Bit 08: HWP_Notification. IA32_HWP_INTERRUPT MSR is supported if set. Bit 09: HWP_Activity_Window. IA32_HWP_REQUEST[bits 41:32] is supported if set. Bit 10: HWP_Energy_Performance_Preference. IA32_HWP_REQUEST[bits 31:24] is supported if set. Bit 11: HWP_Package_Level_Request. IA32_HWP_REQUEST_PKG MSR is supported if set. Bit 12: Reserved. Bit 13: HDC. HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, IA32_THREAD_STALL MSRs are supported if set. Bit 14: Intel® Turbo Boost Max Technology 3.0 available. Bit 15: HWP Capabilities. Highest Performance change is supported if set. Bit 16: HWP PECL override is supported if set. Bit 17: Flexible HWP is supported if set. Bit 18: Fast access mode for the IA32_HWP_REQUEST MSR is supported if set. Bit 19: HW_FEEDBACK. IA32_HW_FEEDBACK_PTR MSR, IA32_HW_FEEDBACK_CONFIG MSR, IA32_PACKAGE_THERM_STATUS MSR bit 26, and IA32_PACKAGE_THERM_INTERRUPT MSR bit 25 are supported if set. Bit 20: Ignoring Idle Logical Processor HWP request is supported if set. Bits 22-21: Reserved. Bit 23: Intel® Thread Director supported if set. IA32_HW_FEEDBACK_CHAR and IA32_HW_FEEDBACK_THREAD_CONFIG MSRs are supported if set. Bit 24: IA32_THERM_INTERRUPT MSR bit 25 is supported if set. Bits 31-25: Reserved.</p>
	EBX	<p>Bits 03-00: Number of Interrupt Thresholds in Digital Thermal Sensor. Bits 31-04: Reserved.</p>
	ECX	<p>Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency. Bits 02-01: Reserved = 0. Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1BOH). Bits 07-04: Reserved = 0. Bits 15-08: Number of Intel® Thread Director classes supported by the processor. Information for that many classes is written into the Intel Thread Director Table by the hardware. Bits 31-16: Reserved = 0.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	<p>EDX</p> <p>Bits 07-00: Bitmap of supported hardware feedback interface capabilities. 0 = When set to 1, indicates support for performance capability reporting. 1 = When set to 1, indicates support for energy efficiency capability reporting. 2-7 = Reserved</p> <p>Bits 11-08: Enumerates the size of the hardware feedback interface structure in number of 4 KB pages; add one to the return value to get the result.</p> <p>Bits 31-16: Index (starting at 0) of this logical processor's row in the hardware feedback interface structure. Note that on some parts the index may be same for multiple logical processors. On some parts the indices may not be contiguous, i.e., there may be unused rows in the hardware feedback interface structure.</p> <p>NOTE: Bits 0 and 1 will always be set together.</p>
<i>Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)</i>	
07H	<p style="text-align: center;">Sub-leaf 0 (Input ECX = 0). *</p> <p>EAX</p> <p>Bits 31-00: Reports the maximum input value for supported leaf 7 sub-leaves.</p> <p>EBX</p> <p>Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: SGX. Supports Intel® Software Guard Extensions (Intel® SGX Extensions) if 1. Bit 03: BMI1. Bit 04: HLE. Bit 05: AVX2. Supports Intel® Advanced Vector Extensions 2 (Intel® AVX2) if 1. Bit 06: FDP_EXCPTN_ONLY. x87 FPU Data Pointer updated only on x87 exceptions if 1. Bit 07: SMEP. Supports Supervisor-Mode Execution Prevention if 1. Bit 08: BMI2. Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID. If 1, supports INVPCID instruction for system software that manages process-context identifiers. Bit 11: RTM. Bit 12: RDT-M. Supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability if 1. Bit 13: Deprecates FPU CS and FPU DS values if 1. Bit 14: MPX. Supports Intel® Memory Protection Extensions if 1. Bit 15: RDT-A. Supports Intel® Resource Director Technology (Intel® RDT) Allocation capability if 1. Bit 16: AVX512F. Bit 17: AVX512DQ. Bit 18: RDSEED. Bit 19: ADX. Bit 20: SMAP. Supports Supervisor-Mode Access Prevention (and the CLAC/STAC instructions) if 1. Bit 21: AVX512_IFMA. Bit 22: Reserved. Bit 23: CLFLUSHOPT. Bit 24: CLWB. Bit 25: Intel Processor Trace. Bit 26: AVX512PF. (Intel® Xeon Phi™ only.) Bit 27: AVX512ER. (Intel® Xeon Phi™ only.) Bit 28: AVX512CD. Bit 29: SHA. supports Intel® Secure Hash Algorithm Extensions (Intel® SHA Extensions) if 1. Bit 30: AVX512BW. Bit 31: AVX512VL.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
ECX	<p>Bit 00: PREFETCHWT1. (Intel® Xeon Phi™ only.)</p> <p>Bit 01: AVX512_VBMI.</p> <p>Bit 02: UMIP. Supports user-mode instruction prevention if 1.</p> <p>Bit 03: PKU. Supports protection keys for user-mode pages if 1.</p> <p>Bit 04: OSPKE. If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions).</p> <p>Bit 05: WAITPKG.</p> <p>Bit 06: AVX512_VBMI2.</p> <p>Bit 07: CET_SS. Supports CET shadow stack features if 1. Processors that set this bit define bits 1:0 of the IA32_U_CET and IA32_S_CET MSRs. Enumerates support for the following MSRs: IA32_INTERRUPT_SPP_TABLE_ADDR, IA32_PL3_SSP, IA32_PL2_SSP, IA32_PL1_SSP, and IA32_PL0_SSP.</p> <p>Bit 08: GFNI.</p> <p>Bit 09: VAES.</p> <p>Bit 10: VPCLMULQDQ.</p> <p>Bit 11: AVX512_VNNI.</p> <p>Bit 12: AVX512_BITALG.</p> <p>Bits 13: TME_EN. If 1, the following MSRs are supported: IA32_TME_CAPABILITY, IA32_TME_ACTIVATE, IA32_TME_EXCLUDE_MASK, and IA32_TME_EXCLUDE_BASE.</p> <p>Bit 14: AVX512_VPOPCNTDQ.</p> <p>Bit 15: Reserved.</p> <p>Bit 16: LA57. Supports 57-bit linear addresses and five-level paging if 1.</p> <p>Bits 21-17: The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode.</p> <p>Bit 22: RDPID and IA32_TSC_AUX are available if 1.</p> <p>Bit 23: KL. Supports Key Locker if 1.</p> <p>Bit 24: BUS_LOCK_DETECT. If 1, indicates support for OS bus-lock detection.</p> <p>Bit 25: CLDEMOTE. Supports cache line demote if 1.</p> <p>Bit 26: Reserved.</p> <p>Bit 27: MOVDIRI. Supports MOVDIRI if 1.</p> <p>Bit 28: MOVDIR64B. Supports MOVDIR64B if 1.</p> <p>Bit 29: ENQCMD. Supports Enqueue Stores if 1.</p> <p>Bit 30: SGX_LC. Supports SGX Launch Configuration if 1.</p> <p>Bit 31: PKS. Supports protection keys for supervisor-mode pages if 1.</p>
EDX	<p>Bit 00: Reserved.</p> <p>Bit 01: SGX-KEYS. If 1, Attestation Services for Intel® SGX is supported.</p> <p>Bit 02: AVX512_4VNNIW. (Intel® Xeon Phi™ only.)</p> <p>Bit 03: AVX512_4FMAPS. (Intel® Xeon Phi™ only.)</p> <p>Bit 04: Fast Short REP MOV.</p> <p>Bit 05: UINTR. If 1, the processor supports user interrupts.</p> <p>Bits 07-06: Reserved.</p> <p>Bit 08: AVX512_VP2INTERSECT.</p> <p>Bit 09: SRBDS_CTRL. If 1, enumerates support for the IA32_MCU_OPT_CTRL MSR and indicates its bit 0 (RNGDS_MITG_DIS) is also supported.</p> <p>Bit 10: MD_CLEAR supported.</p> <p>Bit 11: RTM_ALWAYS_ABORT. If set, any execution of XBEGIN immediately aborts and transitions to the specified fallback address.</p> <p>Bit 12: Reserved.</p> <p>Bit 13: If 1, RTM_FORCE_ABORT supported. Processors that set this bit support the IA32_TSX_FORCE_ABORT MSR. They allow software to set IA32_TSX_FORCE_ABORT[0] (RTM_FORCE_ABORT).</p> <p>Bit 14: SERIALIZE.</p> <p>Bit 15: Hybrid. If 1, the processor is identified as a hybrid part. If CPUID.0.MAXLEAF ≥ 1AH and CPUID.1A.EAX ≠ 0, then the Native Model ID Enumeration Leaf 1AH exists.</p> <p>Bit 16: TSXLDTRK. If 1, the processor supports Intel TSX suspend/resume of load address tracking.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	<p>Bit 17: Reserved. Bit 18: PCONFIG. Supports PCONFIG if 1. Bit 19: Architectural LBRs. If 1, indicates support for architectural LBRs. Bit 20: CET_IBT. Supports CET indirect branch tracking features if 1. Processors that set this bit define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs. Bit 21: Reserved. Bit 22: AMX-BF16. If 1, the processor supports tile computational operations on bfloat16 numbers. Bit 23: AVX512_FP16. Bit 24: AMX-TILE. If 1, the processor supports tile architecture. Bits 25: AMX-INT8. If 1, the processor supports tile computational operations on 8-bit integers. Bit 26: Enumerates support for indirect branch restricted speculation (IBRS) and the indirect branch predictor barrier (IBPB). Processors that set this bit support the IA32_SPEC_CTRL MSR and the IA32_PRED_CMD MSR. They allow software to set IA32_SPEC_CTRL[0] (IBRS) and IA32_PRED_CMD[0] (IBPB). Bit 27: Enumerates support for single thread indirect branch predictors (STIBP). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[1] (STIBP). Bit 28: Enumerates support for L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_FLUSH). Bit 29: Enumerates support for the IA32_ARCH_CAPABILITIES MSR. Bit 30: Enumerates support for the IA32_CORE_CAPABILITIES MSR.</p> <p>IA32_CORE_CAPABILITIES is an architectural MSR that enumerates model-specific features. A bit being set in this MSR indicates that a model specific feature is supported; software must still consult CPUID family/model/stepping to determine the behavior of the enumerated feature as features enumerated in IA32_CORE_CAPABILITIES may have different behavior on different processor models. <i>Some of these features may have behavior that is consistent across processor models (and for which consultation of CPUID family/model/stepping is not necessary); such features are identified explicitly where they are documented in this manual.</i></p> <p>Bit 31: Enumerates support for Speculative Store Bypass Disable (SSBD). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[2] (SSBD).</p> <p>NOTE: * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.</p>
	<i>Structured Extended Feature Enumeration Sub-leaf (EAX = 07H, ECX = 1)</i>
07H	<p>NOTES: Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.</p> <p>EAX</p> <p>This field reports 0 if the sub-leaf index, 1, is invalid.</p> <p>Bits 03-00: Reserved. Bit 04: AVX-VNNI. AVX (VEX-encoded) versions of the Vector Neural Network Instructions. Bit 05: AVX512_BF16. Vector Neural Network Instructions supporting BFLOAT16 inputs and conversion instructions from IEEE single precision. Bits 09-06: Reserved. Bit 10: If 1, supports fast zero-length REP MOVSB. Bit 11: If 1, supports fast short REP STOSB. Bit 12: If 1, supports fast short REP CMPSB, REP SCASB. Bits 21-13: Reserved. Bit 22: HRESET. If 1, supports history reset via the HRESET instruction and the IA32_HRESET_ENABLE MSR. When set, indicates that the Processor History Reset Leaf (EAX = 20H) is valid. Bits 31-23: Reserved.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EBX	<p>This field reports 0 if the sub-leaf index, 1, is invalid.</p> <p>Bit 00: Enumerates the presence of the IA32_PPIN and IA32_PPIN_CTL MSRs. If 1, these MSRs are supported.</p> <p>Bits 31-01: Reserved.</p>
	ECX	<p>This field reports 0 if the sub-leaf index, 1, is invalid; otherwise it is reserved.</p>
	EDX	<p>This field reports 0 if the sub-leaf index, 1, is invalid.</p> <p>Bits 17-00: Reserved.</p> <p>Bit 18: CET_SSS. If 1, indicates that an operating system can enable supervisor shadow stacks as long as it ensures that certain supervisor shadow-stack pushes will not cause page faults (see Section 17.2.3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). When emulating the CPUID instruction, a virtual-machine monitor should return this bit as 0 if those pushes can cause VM exits.</p> <p>Bits 31-19: Reserved.</p>
<i>Structured Extended Feature Enumeration Sub-leaf (EAX = 07H, ECX = 2)</i>		
07H		<p>NOTES: Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.</p> <p>EAX This field reports 0 if the sub-leaf index, 2, is invalid; otherwise it is reserved.</p> <p>EBX This field reports 0 if the sub-leaf index, 2, is invalid; otherwise it is reserved.</p> <p>ECX This field reports 0 if the sub-leaf index, 2, is invalid; otherwise it is reserved.</p> <p>EDX This field reports 0 if the sub-leaf index, 2, is invalid.</p> <p>Bit 00: PSFD. If 1, indicates bit 7 of the IA32_SPEC_CTRL MSR is supported. Bit 7 of this MSR disables Fast Store Forwarding Predictor without disabling Speculative Store Bypass.</p> <p>Bit 01: IPRED_CTRL. If 1, indicates bits 3 and 4 of the IA32_SPEC_CTRL MSR are supported. Bit 3 of this MSR enables IPRED_DIS control for CPL3. Bit 4 of this MSR enables IPRED_DIS control for CPL0/1/2.</p> <p>Bit 02: RRSBA_CTRL. If 1, indicates bits 5 and 6 of the IA32_SPEC_CTRL MSR are supported. Bit 5 of this MSR disables RRSBA behavior for CPL3. Bit 6 of this MSR disables RRSBA behavior for CPL0/1/2.</p> <p>Bit 03: DDPD_U. If 1, indicates bit 8 of the IA32_SPEC_CTRL MSR is supported. Bit 8 of this MSR disables Data Dependent Prefetcher.</p> <p>Bit 04: BHI_CTRL. If 1, indicates bit 10 of the IA32_SPEC_CTRL MSR is supported. Bit 10 of this MSR enables BHI_DIS_S behavior.</p> <p>Bit 05: MCDT_NO. Processors that enumerate this bit as 1 do not exhibit MXCSR Configuration Dependent Timing (MCDT) behavior and do not need to be mitigated to avoid data-dependent behavior for certain instructions.</p> <p>Bits 31-06: Reserved.</p>
<i>Direct Cache Access Information Leaf</i>		
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
<i>Architectural Performance Monitoring Leaf</i>		
0AH	EAX	<p>Bits 07-00: Version ID of architectural performance monitoring.</p> <p>Bits 15-08: Number of general-purpose performance monitoring counter per logical processor.</p> <p>Bits 23-16: Bit width of general-purpose, performance monitoring counter.</p> <p>Bits 31-24: Length of EBX bit vector to enumerate architectural performance monitoring events. Architectural event x is supported if EBX[x]=0 && EAX[31:24]>x.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	<p>EBX Bit 00: Core cycle event not available if 1 or if EAX[31:24]<1. Bit 01: Instruction retired event not available if 1 or if EAX[31:24]<2. Bit 02: Reference cycles event not available if 1 or if EAX[31:24]<3. Bit 03: Last-level cache reference event not available if 1 or if EAX[31:24]<4. Bit 04: Last-level cache misses event not available if 1 or if EAX[31:24]<5. Bit 05: Branch instruction retired event not available if 1 or if EAX[31:24]<6. Bit 06: Branch mispredict retired event not available if 1 or if EAX[31:24]<7. Bit 07: Top-down slots event not available if 1 or if EAX[31:24]<8. Bits 31-08: Reserved = 0.</p> <p>ECX Bits 31-00: Supported fixed counters bit mask. Fixed-function performance counter ‘i’ is supported if bit ‘i’ is 1 (first counter index starts at zero). It is recommended to use the following logic to determine if a Fixed Counter is supported: FxCtr[i]_is_supported := ECX[i] (EDX[4:0] > i);</p> <p>EDX Bits 04-00: Number of contiguous fixed-function performance counters starting from 0 (if Version ID > 1). Bits 12-05: Bit width of fixed-function performance counters (if Version ID > 1). Bits 14-13: Reserved = 0. Bit 15: AnyThread deprecation. Bits 31-16: Reserved = 0.</p>
<i>Extended Topology Enumeration Leaf</i>	
OBH	<p>NOTES:</p> <p><i>CPUID leaf 1FH is a preferred superset to leaf OBH. Intel recommends first checking for the existence of Leaf 1FH before using leaf OBH.</i></p> <p>The sub-leaves of CPUID leaf OBH describe an ordered hierarchy of logical processors starting from the smallest-scoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largest-scoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket).</p> <p>The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated.</p> <p>For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor.</p> <p>If sub-leaf index “N” returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than “N” shall also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.</p> <p>EAX Bits 04-00: The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly. Bits 31-05: Reserved.</p> <p>EBX Bits 15-00: The number of logical processors across all instances of this domain within the next higher-scoped domain. (For example, in a processor socket/package comprising “M” dies of “N” cores each, where each core has “L” logical processors, the “die” domain sub-leaf value of this field would be M*N*L.) This number reflects configuration as shipped by Intel. Note, software must not use this field to enumerate processor topology*. Bits 31-16: Reserved.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor										
	<p>ECX</p> <p>Bits 07-00: The input ECX sub-leaf index.</p> <p>Bits 15-08: Domain Type. This field provides an identification value which indicates the domain as shown below. Although domains are ordered, their assigned identification values are not and software should not depend on it.</p> <table border="0" data-bbox="440 436 1385 562"> <thead> <tr> <th style="text-align: left;"><u>Hierarchy</u></th> <th style="text-align: left;"><u>Domain</u></th> <th style="text-align: left;"><u>Domain Type Identification Value</u></th> </tr> </thead> <tbody> <tr> <td>Lowest</td> <td>Logical Processor</td> <td>1</td> </tr> <tr> <td>Highest</td> <td>Core</td> <td>2</td> </tr> </tbody> </table> <p>(Note that enumeration values of 0 and 3-255 are reserved.)</p> <p>Bits 31-16: Reserved.</p> <p>EDX</p> <p>Bits 31-00: x2APIC ID of the current logical processor.</p> <p>NOTES:</p> <p>* Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.</p>	<u>Hierarchy</u>	<u>Domain</u>	<u>Domain Type Identification Value</u>	Lowest	Logical Processor	1	Highest	Core	2	
<u>Hierarchy</u>	<u>Domain</u>	<u>Domain Type Identification Value</u>									
Lowest	Logical Processor	1									
Highest	Core	2									
<i>Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)</i>											
<p>0DH</p> <p>EAX</p> <p>EBX</p> <p>ECX</p> <p>EDX</p>		<p>NOTES:</p> <p>Leaf 0DH main leaf (ECX = 0).</p> <p>Bits 31-00: Reports the supported bits of the lower 32 bits of XCR0. XCR0[n] can be set to 1 only if EAX[n] is 1.</p> <p>Bit 00: x87 state.</p> <p>Bit 01: SSE state.</p> <p>Bit 02: AVX state.</p> <p>Bits 04-03: MPX state.</p> <p>Bits 07-05: AVX-512 state.</p> <p>Bit 08: Used for IA32_XSS.</p> <p>Bit 09: PKRU state.</p> <p>Bits 16-10: Used for IA32_XSS.</p> <p>Bit 17: TILECFG state.</p> <p>Bit 18: TILEDATA state.</p> <p>Bits 31-19: Reserved.</p> <p>Bits 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.</p> <p>Bit 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e., all the valid bit fields in XCR0.</p> <p>Bit 31-00: Reports the supported bits of the upper 32 bits of XCR0. XCR0[n+32] can be set to 1 only if EDX[n] is 1.</p> <p>Bits 31-00: Reserved.</p>									
<i>Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)</i>											
<p>0DH</p> <p>EAX</p>		<p>Bit 00: XSAVEOPT is available.</p> <p>Bit 01: Supports XSAVEC and the compacted form of XRSTOR if set.</p> <p>Bit 02: Supports XGETBV with ECX = 1 if set.</p> <p>Bit 03: Supports XSAVES/XRSTORS and IA32_XSS if set.</p> <p>Bit 04: Supports extended feature disable (XFD) if set.</p> <p>Bits 31-05: Reserved.</p>									

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	<p>EBX Bits 31-00: The size in bytes of the XSAVE area containing all states enabled by XCRO IA32_XSS.</p> <p>NOTES: If EAX[3] is enumerated as 0 and EAX[1] is enumerated as 1, EBX enumerates the size of the XSAVE area containing all states enabled by XCRO. If EAX[1] and EAX[3] are both enumerated as 0, EBX enumerates zero.</p> <p>ECX Bits 31-00: Reports the supported bits of the lower 32 bits of the IA32_XSS MSR. IA32_XSS[n] can be set to 1 only if ECX[n] is 1. Bits 07-00: Used for XCRO. Bit 08: PT state. Bit 09: Used for XCRO. Bit 10: PASID state. Bit 11: CET user state. Bit 12: CET supervisor state. Bit 13: HDC state. Bit 14: UINTR state. Bit 15: LBR state (only for the architectural LBR feature). Bit 16: HWP state. Bits 18-17: Used for XCRO. Bits 31-19: Reserved.</p> <p>EDX Bits 31-00: Reports the supported bits of the upper 32 bits of the IA32_XSS MSR. IA32_XSS[n+32] can be set to 1 only if EDX[n] is 1. Bits 31-00: Reserved.</p>
<i>Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)</i>	
0DH	<p>NOTES: Leaf 0DH output depends on the initial value in ECX. Each sub-leaf index (starting at position 2) is supported if it corresponds to a supported bit in either the XCRO register or the IA32_XSS MSR. * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n (0 ≤ n ≤ 31) is invalid if sub-leaf 0 returns 0 in EAX[n] and sub-leaf 1 returns 0 in ECX[n]. Sub-leaf n (32 ≤ n ≤ 63) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32].</p> <p>EAX Bits 31-00: The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index, n.</p> <p>EBX Bits 31-00: The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, n, does not map to a valid bit in the XCRO register*.</p> <p>ECX Bit 00 is set if the bit n (corresponding to the sub-leaf index) is supported in the IA32_XSS MSR; it is clear if bit n is instead supported in XCRO. Bit 01 is set if, when the compacted format of an XSAVE area is used, this extended state component located on the next 64-byte boundary following the preceding state component (otherwise, it is located immediately following the preceding state component). Bits 31-02 are reserved. This field reports 0 if the sub-leaf index, n, is invalid*.</p> <p>EDX This field reports 0 if the sub-leaf index, n, is invalid*; otherwise it is reserved.</p>
<i>Intel Resource Director Technology (Intel RDT) Monitoring Enumeration Sub-leaf (EAX = 0FH, ECX = 0)</i>	
0FH	<p>NOTES: Leaf 0FH output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.</p> <p>EAX Reserved.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	EBX Bits 31-00: Maximum range (zero-based) of RMID within this physical processor of all types. ECX Reserved. EDX Bit 00: Reserved. Bit 01: Supports L3 Cache Intel RDT Monitoring if 1. Bits 31-02: Reserved.
<i>L3 Cache Intel RDT Monitoring Capability Enumeration Sub-leaf (EAX = 0FH, ECX = 1)</i>	
0FH	NOTES: Leaf 0FH output depends on the initial value in ECX. EAX Reserved. EBX Bits 31-00: Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics. ECX Maximum range (zero-based) of RMID of this resource type. EDX Bit 00: Supports L3 occupancy monitoring if 1. Bit 01: Supports L3 Total Bandwidth monitoring if 1. Bit 02: Supports L3 Local Bandwidth monitoring if 1. Bits 31-03: Reserved.
<i>Intel Resource Director Technology (Intel RDT) Allocation Enumeration Sub-leaf (EAX = 10H, ECX = 0)</i>	
10H	NOTES: Leaf 10H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX. EAX Reserved. EBX Bit 00: Reserved. Bit 01: Supports L3 Cache Allocation Technology if 1. Bit 02: Supports L2 Cache Allocation Technology if 1. Bit 03: Supports Memory Bandwidth Allocation if 1. Bits 31-04: Reserved. ECX Reserved. EDX Reserved.
<i>L3 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID = 1)</i>	
10H	NOTES: Leaf 10H output depends on the initial value in ECX. EAX Bits 04-00: Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31-05: Reserved. EBX Bits 31-00: Bit-granular map of isolation/contention of allocation units. ECX Bits 01-00: Reserved. Bit 02: Code and Data Prioritization Technology supported if 1. Bits 31-03: Reserved. EDX Bits 15-00: Highest COS number supported for this ResID. Bits 31-16: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
<i>L2 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID =2)</i>	
10H	<p>NOTES: Leaf 10H output depends on the initial value in ECX.</p> <p>EAX Bits 04-00: Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31-05: Reserved.</p> <p>EBX Bits 31-00: Bit-granular map of isolation/contention of allocation units.</p> <p>ECX Bits 31-00: Reserved.</p> <p>EDX Bits 15-00: Highest COS number supported for this ResID. Bits 31-16: Reserved.</p>
<i>Memory Bandwidth Allocation Enumeration Sub-leaf (EAX = 10H, ECX = ResID =3)</i>	
10H	<p>NOTES: Leaf 10H output depends on the initial value in ECX.</p> <p>EAX Bits 11-00: Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result. Bits 31-12: Reserved.</p> <p>EBX Bits 31-00: Reserved.</p> <p>ECX Bits 01-00: Reserved. Bit 02: Reports whether the response of the delay values is linear. Bits 31-03: Reserved.</p> <p>EDX Bits 15-00: Highest COS number supported for this ResID. Bits 31-16: Reserved.</p>
<i>Intel SGX Capability Enumeration Leaf, sub-leaf 0 (EAX = 12H, ECX = 0)</i>	
12H	<p>NOTES: Leaf 12H sub-leaf 0 (ECX = 0) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.</p> <p>EAX Bit 00: SGX1. If 1, Indicates Intel SGX supports the collection of SGX1 leaf functions. Bit 01: SGX2. If 1, Indicates Intel SGX supports the collection of SGX2 leaf functions. Bits 04-02: Reserved. Bit 05: If 1, indicates Intel SGX supports ENCLV instruction leaves EINCVIRTUAL, EDECVIRTUAL, and ESETCONTEXT. Bit 06: If 1, indicates Intel SGX supports ENCLS instruction leaves ETRACKC, ERDINFO, ELDBC, and ELDUC. Bit 07: If 1, indicates Intel SGX supports ENCLU instruction leaf EVERIFYREPORT2. Bits 09-08: Reserved. Bit 10: If 1, indicates Intel SGX supports ENCLS instruction leaf EUPDATESVN. Bit 11: If 1, indicates Intel SGX supports ENCLU instruction leaf EDECCSSA. Bits 31-12: Reserved.</p> <p>EBX Bits 31-00: MISCSELECT. Bit vector of supported extended SGX features.</p> <p>ECX Bits 31-00: Reserved.</p> <p>EDX Bits 07-00: MaxEnclaveSize_Not64. The maximum supported enclave size in non-64-bit mode is 2^(EDX[7:0]). Bits 15-08: MaxEnclaveSize_64. The maximum supported enclave size in 64-bit mode is 2^(EDX[15:8]). Bits 31-16: Reserved.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
<i>Intel SGX Attributes Enumeration Leaf, sub-leaf 1 (EAX = 12H, ECX = 1)</i>	
12H	<p>NOTES: Leaf 12H sub-leaf 1 (ECX = 1) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.</p> <p>EAX Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE. EBX Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE. ECX Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE. EDX Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.</p>
<i>Intel SGX EPC Enumeration Leaf, sub-leaves (EAX = 12H, ECX = 2 or higher)</i>	
12H	<p>NOTES: Leaf 12H sub-leaf 2 or higher (ECX >= 2) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1. For sub-leaves (ECX = 2 or higher), definition of EDX,ECX,EBX,EAX[31:4] depends on the sub-leaf type listed below.</p> <p>EAX Bit 03-00: Sub-leaf Type 0000b: Indicates this sub-leaf is invalid. 0001b: This sub-leaf enumerates an EPC section. EBX:EAX and EDX:ECX provide information on the Enclave Page Cache (EPC) section. All other type encodings are reserved.</p> <p>Type 0000b. This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.</p> <p>Type 0001b. This sub-leaf enumerates an EPC sections with EDX:ECX, EBX:EAX defined as follows. EAX[11:04]: Reserved (enumerate 0). EAX[31:12]: Bits 31:12 of the physical address of the base of the EPC section. EBX[19:00]: Bits 51:32 of the physical address of the base of the EPC section. EBX[31:20]: Reserved. ECX[03:00]: EPC section property encoding defined as follows: If ECX[3:0] = 0000b, then all bits of the EDX:ECX pair are enumerated as 0. If ECX[3:0] = 0001b, then this section has confidentiality and integrity protection. If ECX[3:0] = 0010b, then this section has confidentiality protection only. All other encodings are reserved. ECX[11:04]: Reserved (enumerate 0). ECX[31:12]: Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[19:00]: Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[31:20]: Reserved.</p>
<i>Intel Processor Trace Enumeration Main Leaf (EAX = 14H, ECX = 0)</i>	
14H	<p>NOTES: Leaf 14H main leaf (ECX = 0).</p> <p>EAX Bits 31-00: Reports the maximum sub-leaf supported in leaf 14H.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EBX	<p>Bit 00: If 1, indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed.</p> <p>Bit 01: If 1, indicates support of Configurable PSB and Cycle-Accurate Mode.</p> <p>Bit 02: If 1, indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset.</p> <p>Bit 03: If 1, indicates support of MTC timing packet and suppression of COFI-based packets.</p> <p>Bit 04: If 1, indicates support of PTWRITE. Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets.</p> <p>Bit 05: If 1, indicates support of Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation.</p> <p>Bit 06: If 1, indicates support for PSB and PMI preservation. Writes can set IA32_RTIT_CTL[56] (InjectPsbPmiOnEnable), enabling the processor to set IA32_RTIT_STATUS[7] (PendTopaPMI) and/or IA32_RTIT_STATUS[6] (PendPSB) in order to preserve ToPA PMIs and/or PSBs otherwise lost due to Intel PT disable. Writes can also set PendToPAPMI and PendPSB.</p> <p>Bit 07: If 1, writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation.</p> <p>Bit 08: If 1, writes can set IA32_RTIT_CTL[55] (DistNT), disabling TNT packet generation.</p> <p>Bit 31-09: Reserved.</p>
	ECX	<p>Bit 00: If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed.</p> <p>Bit 01: If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOffsetTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS.</p> <p>Bit 02: If 1, indicates support of Single-Range Output scheme.</p> <p>Bit 03: If 1, indicates support of output to Trace Transport subsystem.</p> <p>Bit 30-04: Reserved.</p> <p>Bit 31: If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.</p>
	EDX	Bits 31-00: Reserved.
<i>Intel Processor Trace Enumeration Sub-leaf (EAX = 14H, ECX = 1)</i>		
14H	EAX	<p>Bits 02-00: Number of configurable Address Ranges for filtering.</p> <p>Bits 15-03: Reserved.</p> <p>Bits 31-16: Bitmap of supported MTC period encodings.</p>
	EBX	<p>Bits 15-00: Bitmap of supported Cycle Threshold value encodings.</p> <p>Bit 31-16: Bitmap of supported Configurable PSB frequency encodings.</p>
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
<i>Time Stamp Counter and Nominal Core Crystal Clock Information Leaf</i>		
15H		<p>NOTES:</p> <p>If EBX[31:0] is 0, the TSC/"core crystal clock" ratio is not enumerated.</p> <p>EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency.</p> <p>If ECX is 0, the nominal core crystal clock frequency is not enumerated.</p> <p>"TSC frequency" = "core crystal clock frequency" * EBX/EAX.</p> <p>The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.</p>
	EAX	Bits 31-00: An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31-00: An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.
	ECX	Bits 31-00: An unsigned integer which is the nominal frequency of the core crystal clock in Hz.
	EDX	Bits 31-00: Reserved = 0.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
<i>Processor Frequency Information Leaf</i>		
16H	EAX	Bits 15-00: Processor Base Frequency (in MHz). Bits 31-16: Reserved = 0.
	EBX	Bits 15-00: Maximum Frequency (in MHz). Bits 31-16: Reserved = 0.
	ECX	Bits 15-00: Bus (Reference) Frequency (in MHz). Bits 31-16: Reserved = 0.
	EDX	Reserved.
<p>NOTES:</p> <p>* Data is returned from this interface in accordance with the processor's specification and does not reflect actual values. Suitable use of this data includes the display of processor information in like manner to the processor brand string and for determining the appropriate range to use when displaying processor information e.g. frequency history graphs. The returned information should not be used for any other purpose as the returned information does not accurately correlate to information / counters returned by other processor interfaces.</p> <p>While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.</p>		
<i>System-On-Chip Vendor Attribute Enumeration Main Leaf (EAX = 17H, ECX = 0)</i>		
17H		<p>NOTES:</p> <p>Leaf 17H main leaf (ECX = 0). Leaf 17H output depends on the initial value in ECX. Leaf 17H sub-leaves 1 through 3 reports SOC Vendor Brand String. Leaf 17H is valid if MaxSOCID_Index >= 3. Leaf 17H sub-leaves 4 and above are reserved.</p>
	EAX	Bits 31-00: MaxSOCID_Index. Reports the maximum input value of supported sub-leaf in leaf 17H.
	EBX	Bits 15-00: SOC Vendor ID. Bit 16: IsVendorScheme. If 1, the SOC Vendor ID field is assigned via an industry standard enumeration scheme. Otherwise, the SOC Vendor ID field is assigned by Intel. Bits 31-17: Reserved = 0.
	ECX	Bits 31-00: Project ID. A unique number an SOC vendor assigns to its SOC projects.
	EDX	Bits 31-00: Stepping ID. A unique number within an SOC project that an SOC vendor assigns.
<i>System-On-Chip Vendor Attribute Enumeration Sub-leaf (EAX = 17H, ECX = 1..3)</i>		
17H	EAX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	EBX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	ECX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	EDX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
<p>NOTES:</p> <p>Leaf 17H output depends on the initial value in ECX. SOC Vendor Brand String is a UTF-8 encoded string padded with trailing bytes of 00H. The complete SOC Vendor Brand String is constructed by concatenating in ascending order of EAX:EBX:ECX:EDX and from the sub-leaf 1 fragment towards sub-leaf 3.</p>		

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
<i>System-On-Chip Vendor Attribute Enumeration Sub-leaves (EAX = 17H, ECX > MaxSOCID_Index)</i>	
17H	<p>NOTES: Leaf 17H output depends on the initial value in ECX.</p> <p>EAX Bits 31-00: Reserved = 0. EBX Bits 31-00: Reserved = 0. ECX Bits 31-00: Reserved = 0. EDX Bits 31-00: Reserved = 0.</p>
<i>Deterministic Address Translation Parameters Main Leaf (EAX = 18H, ECX = 0)</i>	
18H	<p>NOTES: Each sub-leaf enumerates a different address translation structure. If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX. A sub-leaf index is also invalid if EDX[4:0] returns 0. Valid sub-leaves do not need to be contiguous or in any particular order. A valid sub-leaf may be in a higher input ECX value than an invalid sub-leaf or than a valid sub-leaf of a higher or lower-level structure. * Some unified TLBs will allow a single TLB entry to satisfy data read/write and instruction fetches. Others will require separate entries (e.g., one loaded on data read/write and another loaded on an instruction fetch) . Please see the Intel® 64 and IA-32 Architectures Optimization Reference Manual for details of a particular product. ** Add one to the return value to get the result.</p> <p>EAX Bits 31-00: Reports the maximum input value of supported sub-leaf in leaf 18H. EBX Bit 00: 4K page size entries supported by this structure. Bit 01: 2MB page size entries supported by this structure. Bit 02: 4MB page size entries supported by this structure. Bit 03: 1 GB page size entries supported by this structure. Bits 07-04: Reserved. Bits 10-08: Partitioning (0: Soft partitioning between the logical processors sharing this structure). Bits 15-11: Reserved. Bits 31-16: W = Ways of associativity. ECX Bits 31-00: S = Number of Sets. EDX Bits 04-00: Translation cache type field. 00000b: Null (indicates this sub-leaf is not valid). 00001b: Data TLB. 00010b: Instruction TLB. 00011b: Unified TLB*. 00100b: Load Only TLB. Hit on loads; fills on both loads and stores. 00101b: Store Only TLB. Hit on stores; fill on stores. All other encodings are reserved. Bits 07-05: Translation cache level (starts at 1). Bit 08: Fully associative structure. Bits 13-09: Reserved. Bits 25-14: Maximum number of addressable IDs for logical processors sharing this translation cache.** Bits 31-26: Reserved.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
<i>Deterministic Address Translation Parameters Sub-leaf (EAX = 18H, ECX ≥ 1)</i>		
18H		<p>NOTES:</p> <p>Each sub-leaf enumerates a different address translation structure. If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX. A sub-leaf index is also invalid if EDX[4:0] returns 0. Valid sub-leaves do not need to be contiguous or in any particular order. A valid sub-leaf may be in a higher input ECX value than an invalid sub-leaf or than a valid sub-leaf of a higher or lower-level structure.</p> <p>* Some unified TLBs will allow a single TLB entry to satisfy data read/write and instruction fetches. Others will require separate entries (e.g., one loaded on data read/write and another loaded on an instruction fetch. See the Intel® 64 and IA-32 Architectures Optimization Reference Manual for details of a particular product.</p> <p>** Add one to the return value to get the result.</p> <p>EAX Bits 31-00: Reserved.</p> <p>EBX Bit 00: 4K page size entries supported by this structure. Bit 01: 2MB page size entries supported by this structure. Bit 02: 4MB page size entries supported by this structure. Bit 03: 1 GB page size entries supported by this structure. Bits 07-04: Reserved. Bits 10-08: Partitioning (0: Soft partitioning between the logical processors sharing this structure). Bits 15-11: Reserved. Bits 31-16: W = Ways of associativity.</p> <p>ECX Bits 31-00: S = Number of Sets.</p> <p>EDX Bits 04-00: Translation cache type field. 0000b: Null (indicates this sub-leaf is not valid). 0001b: Data TLB. 0010b: Instruction TLB. 0011b: Unified TLB*. All other encodings are reserved. Bits 07-05: Translation cache level (starts at 1). Bit 08: Fully associative structure. Bits 13-09: Reserved. Bits 25-14: Maximum number of addressable IDs for logical processors sharing this translation cache** Bits 31-26: Reserved.</p>
<i>Key Locker Leaf (EAX = 19H)</i>		
19H		<p>EAX Bit 00: Key Locker restriction of CPL0-only supported. Bit 01: Key Locker restriction of no-encrypt supported. Bit 02: Key Locker restriction of no-decrypt supported. Bits 31-03: Reserved.</p> <p>EBX Bit 00: AESKLE. If 1, the AES Key Locker instructions are fully enabled. Bit 01: Reserved. Bit 02: If 1, the AES wide Key Locker instructions are supported. Bit 03: Reserved. Bit 04: If 1, the platform supports the Key Locker MSRs (IA32_COPY_LOCAL_TO_PLATFORM, IA23_COPY_PLATFORM_TO_LOCAL, IA32_COPY_STATUS, and IA32_IWKEYBACKUP_STATUS) and backing up the internal wrapping key. Bits 31-05: Reserved.</p> <p>ECX Bit 00: If 1, the NoBackup parameter to LOADIWKEY is supported. Bit 01: If 1, KeySource encoding of 1 (randomization of the internal wrapping key) is supported. Bits 31-02: Reserved.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
EDX	Reserved.
<i>Native Model ID Enumeration Leaf (EAX = 1AH, ECX = 0)</i>	
1AH	<p>NOTES:</p> <p>This leaf exists on all hybrid parts, however this leaf is not only available on hybrid parts. The following algorithm is used for detection of this leaf: If CPUID.0.MAXLEAF \geq 1AH and CPUID.1A.EAX \neq 0, then the leaf exists.</p> <p>EAX Enumerates the native model ID and core type. Bits 31-24: Core type* 10H: Reserved 20H: Intel Atom® 30H: Reserved 40H: Intel® Core™ Bits 23-00: Native model ID of the core. The core-type and native model ID can be used to uniquely identify the microarchitecture of the core. This native model ID is not unique across core types, and not related to the model ID reported in CPUID leaf 01H, and does not identify the SOC. * The core type may only be used as an identification of the microarchitecture for this logical processor and its numeric value has no significance, neither large nor small. This field neither implies nor expresses any other attribute to this logical processor and software should not assume any.</p> <p>EBX Reserved.</p> <p>ECX Reserved.</p> <p>EDX Reserved.</p>
<i>PCONFIG Information Sub-leaf (EAX = 1BH, ECX \geq 0)</i>	
1BH	<p>For details on this sub-leaf, see “INPUT EAX = 1BH: Returns PCONFIG Information” on page 3-252.</p> <p>NOTE:</p> <p>Leaf 1BH is supported if CPUID.(EAX=07H, ECX=0H):EDX[18] = 1.</p>
<i>Last Branch Records Information Leaf (EAX = 1CH)</i>	
1CH	<p>NOTE:</p> <p>This leaf pertains to the architectural feature.</p> <p>EAX Bits 07-00: Supported LBR Depth Values. For each bit n set in this field, the IA32_LBR_DEPTH.DEPTH value $8*(n+1)$ is supported. Bits 29-08: Reserved. Bit 30: Deep C-state Reset. If set, indicates that LBRs may be cleared on an MWAIT that requests a C-state numerically greater than C1. Bit 31: IP Values Contain LIP. If set, LBR IP values contain LIP. If clear, IP values contain Effective IP.</p> <p>EBX Bit 00: CPL Filtering Supported. If set, the processor supports setting IA32_LBR_CTL[2:1] to non-zero value. Bit 01: Branch Filtering Supported. If set, the processor supports setting IA32_LBR_CTL[22:16] to non-zero value. Bit 02: Call-stack Mode Supported. If set, the processor supports setting IA32_LBR_CTL[3] to 1. Bits 31-03: Reserved.</p> <p>ECX Bit 00: Mispredict Bit Supported. IA32_LBR_x_INFO[63] holds indication of branch misprediction (MISPRED). Bit 01: Timed LBRs Supported. IA32_LBR_x_INFO[15:0] holds CPU cycles since last LBR entry (CYC_CNT), and IA32_LBR_x_INFO[60] holds an indication of whether the value held there is valid (CYC_CNT_VALID). Bit 02: Branch Type Field Supported. IA32_LBR_INFO_x[59:56] holds indication of the recorded operation's branch type (BR_TYPE). Bits 31-03: Reserved.</p>

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 31-00: Reserved.
<i>Tile Information Main Leaf (EAX = 1DH, ECX = 0)</i>		
1DH	<p>NOTES:</p> <p>For sub-leaves of 1DH, they are indexed by the palette id. Leaf 1DH sub-leaves 2 and above are reserved.</p> <p>EAX Bits 31-00: max_palette. Highest numbered palette sub-leaf. Value = 1. EBX Bits 31-00: Reserved = 0. ECX Bits 31-00: Reserved = 0. EDX Bits 31-00: Reserved = 0.</p>	
<i>Tile Palette 1 Sub-leaf (EAX = 1DH, ECX = 1)</i>		
1DH	EAX	Bits 15-00: Palette 1 total_tile_bytes. Value = 8192. Bits 31-16: Palette 1 bytes_per_tile. Value = 1024.
	EBX	Bits 15-00: Palette 1 bytes_per_row. Value = 64. Bits 31-16: Palette 1 max_names (number of tile registers). Value = 8.
	ECX	Bits 15-00: Palette 1 max_rows. Value = 16. Bits 31-16: Reserved = 0.
	EDX	Bits 31-00: Reserved = 0.
<i>TMUL Information Main Leaf (EAX = 1EH, ECX = 0)</i>		
1EH	<p>NOTE:</p> <p>Leaf 1EH sub-leaves 1 and above are reserved.</p> <p>EAX Bits 31-00: Reserved = 0. EBX Bits 07-00: tmul_maxk (rows or columns). Value = 16. Bits 23-08: tmul_maxn (column bytes). Value = 64. Bits 31-24: Reserved = 0. ECX Bits 31-00: Reserved = 0. EDX Bits 31-00: Reserved = 0.</p>	
<i>V2 Extended Topology Enumeration Leaf</i>		
1FH	<p>NOTES:</p> <p><i>CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends using leaf 1FH when available rather than leaf 0BH and ensuring that any leaf 0BH algorithms are updated to support leaf 1FH.</i></p> <p><i>The sub-leaves of CPUID leaf 1FH describe an ordered hierarchy of logical processors starting from the smallest-scoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largest-scoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket).</i></p> <p><i>The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated. As an example, a processor may report an ordered hierarchy consisting only of "Logical Processor," "Core," and "Die."</i></p> <p><i>For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor.</i></p> <p><i>If sub-leaf index "N" returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than "N" shall also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.</i></p>	

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor																									
	EAX	<p>Bits 04-00: The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly.</p> <p>Bits 31-05: Reserved.</p>																								
	EBX	<p>Bits 15-00: The number of logical processors across all instances of this domain within the next higher-scoped domain. (For example, in a processor socket/package comprising “M” dies of “N” cores each, where each core has “L” logical processors, the “die” domain sub-leaf value of this field would be M*N*L.) This number reflects configuration as shipped by Intel. Note, software must not use this field to enumerate processor topology*.</p> <p>Bits 31-16: Reserved.</p>																								
	ECX	<p>Bits 07-00: The input ECX sub-leaf index.</p> <p>Bits 15-08: Domain Type. This field provides an identification value which indicates the domain as shown below. Although domains are ordered, as also shown below, their assigned identification values are not and software should not depend on it. (For example, if a new domain between core and module is specified, it will have an identification value higher than 5.)</p> <table border="1" style="margin-left: 40px;"> <thead> <tr> <th>Hierarchy</th> <th>Domain</th> <th>Domain Type Identification Value</th> </tr> </thead> <tbody> <tr> <td>Lowest</td> <td>Logical Processor</td> <td>1</td> </tr> <tr> <td>...</td> <td>Core</td> <td>2</td> </tr> <tr> <td>...</td> <td>Module</td> <td>3</td> </tr> <tr> <td>...</td> <td>Tile</td> <td>4</td> </tr> <tr> <td>...</td> <td>Die</td> <td>5</td> </tr> <tr> <td>...</td> <td>DieGrp</td> <td>6</td> </tr> <tr> <td>Highest</td> <td>Package/Socket</td> <td>(implied)</td> </tr> </tbody> </table> <p>(Note that enumeration values of 0 and 7-255 are reserved.)</p> <p>Bits 31-16: Reserved.</p>	Hierarchy	Domain	Domain Type Identification Value	Lowest	Logical Processor	1	...	Core	2	...	Module	3	...	Tile	4	...	Die	5	...	DieGrp	6	Highest	Package/Socket	(implied)
Hierarchy	Domain	Domain Type Identification Value																								
Lowest	Logical Processor	1																								
...	Core	2																								
...	Module	3																								
...	Tile	4																								
...	Die	5																								
...	DieGrp	6																								
Highest	Package/Socket	(implied)																								
	EDX	<p>Bits 31-00: x2APIC ID of the current logical processor. It is always valid and does not vary with the sub-leaf index in ECX.</p> <p>NOTES:</p> <p>* Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.</p>																								
<i>Processor History Reset Sub-leaf (EAX = 20H, ECX = 0)</i>																										
20H	EAX	Reports the maximum number of sub-leaves that are supported in leaf 20H.																								
	EBX	<p>Indicates which bits may be set in the IA32_HRESET_ENABLE MSR to enable reset of different components of hardware-maintained history.</p> <p>Bit 00: Indicates support for both HRESET’s EAX[0] parameter, and IA32_HRESET_ENABLE[0] set by the OS to enable reset of Intel® Thread Director history.</p> <p>Bits 31-01: Reserved = 0.</p>																								
	ECX	Reserved.																								
	EDX	Reserved.																								
<i>Unimplemented CPUID Leaf Functions</i>																										
21H		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is 21H. If the value returned by CPUID.0:EAX (the maximum input value for basic CPUID information) is at least 21H, 0 is returned in the registers EAX, EBX, ECX, and EDX. Otherwise, the data for the highest basic information leaf is returned.																								

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
40000000H - 4FFFFFFFH	Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.	
<i>Extended Function CPUID Information</i>		
80000000H	EAX EBX ECX EDX	Maximum Input Value for Extended Function CPUID Information. Reserved. Reserved. Reserved.
80000001H	EAX EBX ECX EDX	Extended Processor Signature and Feature Bits. Reserved. Bit 00: LAHF/SAHF available in 64-bit mode.* Bits 04-01: Reserved. Bit 05: LZCNT. Bits 07-06: Reserved. Bit 08: PREFETCHW. Bits 31-09: Reserved. Bits 10-00: Reserved. Bit 11: SYSCALL/SYSRET.** Bits 19-12: Reserved = 0. Bit 20: Execute Disable Bit available. Bits 25-21: Reserved = 0. Bit 26: 1-GByte pages are available if 1. Bit 27: RDTSCP and IA32_TSC_AUX are available if 1. Bit 28: Reserved = 0. Bit 29: Intel® 64 Architecture available if 1. Bits 31-30: Reserved = 0. NOTES: * LAHF and SAHF are always available in other modes, regardless of the enumeration of this feature flag. ** Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.
80000002H	EAX EBX ECX EDX	Processor Brand String. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.
80000003H	EAX EBX ECX EDX	Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.
80000004H	EAX EBX ECX EDX	Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.
80000005H	EAX EBX ECX EDX	Reserved = 0. Reserved = 0. Reserved = 0. Reserved = 0.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
80000006H	EAX EBX ECX EDX	Reserved = 0. Reserved = 0. Bits 07-00: Cache Line size in bytes. Bits 11-08: Reserved. Bits 15-12: L2 Associativity field *. Bits 31-16: Cache size in 1K units. Reserved = 0. NOTES: * L2 associativity field encodings: 00H - Disabled 01H - 1 way (direct mapped) 02H - 2 ways 03H - Reserved 04H - 4 ways 05H - Reserved 06H - 8 ways 07H - See CPUID leaf 04H, sub-leaf 2** 08H - 16 ways 09H - Reserved 0AH - 32 ways 0BH - 48 ways 0CH - 64 ways 0DH - 96 ways 0EH - 128 ways 0FH - Fully associative ** CPUID leaf 04H provides details of deterministic cache parameters, including the L2 cache in sub-leaf 2
80000007H	EAX EBX ECX EDX	Reserved = 0. Reserved = 0. Reserved = 0. Bits 07-00: Reserved = 0. Bit 08: Invariant TSC available if 1. Bits 31-09: Reserved = 0.
80000008H	EAX EBX ECX EDX	Linear/Physical Address size. Bits 07-00: #Physical Address Bits*. Bits 15-08: #Linear Address Bits. Bits 31-16: Reserved = 0. Bits 08-00: Reserved = 0. Bit 09: WBNOINVD is available if 1. Bits 31-10: Reserved = 0. Reserved = 0. Reserved = 0. NOTES: * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

INPUT EAX = 0: Returns CPUID's Highest Value for Basic Processor Information and the Vendor Identification String

When CPUID executes with EAX set to 0, the processor returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register and is processor specific.

A vendor identification string is also returned in EBX, EDX, and ECX. For Intel processors, the string is "GenuineIntel" and is expressed:

EBX := 756e6547h (* "Genu", with G in the low eight bits of BL *)

EDX := 49656e69h (* "inel", with i in the low eight bits of DL *)

ECX := 6c65746eh (* "ntel", with n in the low eight bits of CL *)

INPUT EAX = 80000000H: Returns CPUID’s Highest Value for Extended Processor Information

When CPUID executes with EAX set to 80000000H, the processor returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register and is processor specific.

IA32_BIOS_SIGN_ID Returns Microcode Update Signature

For processors that support the microcode update facility, the IA32_BIOS_SIGN_ID MSR is loaded with the update signature whenever CPUID executes. The signature is returned in the upper DWORD. For details, see Chapter 10 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

INPUT EAX = 01H: Returns Model, Family, Stepping Information

When CPUID executes with EAX set to 01H, version information is returned in EAX (see Figure 3-6). For example: model, family, and processor type for the Intel Xeon processor 5100 series is as follows:

- Model — 1111B
- Family — 0101B
- Processor Type — 00B

See Table 3-9 for available processor type values. Stepping IDs are provided as needed.

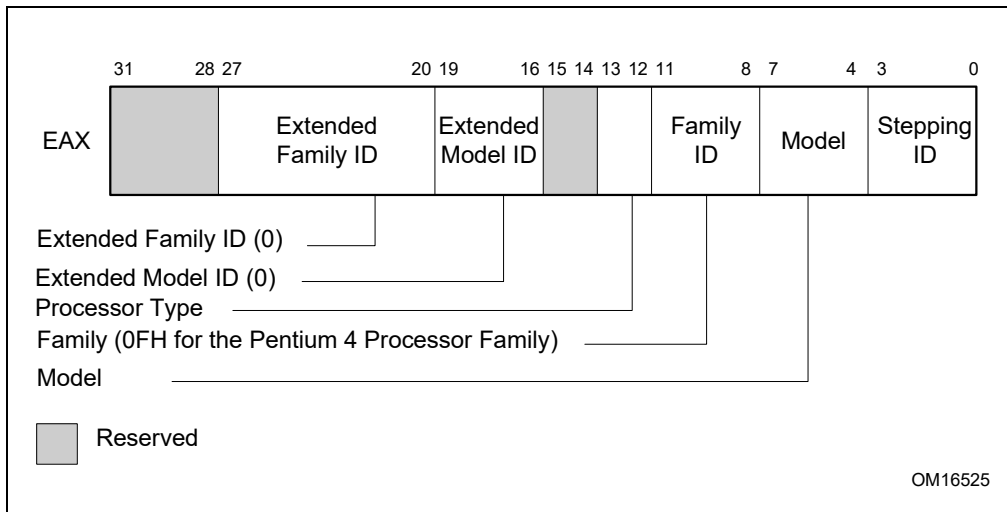


Figure 3-6. Version Information Returned by CPUID in EAX

Table 3-9. Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B
Dual processor (not applicable to Intel486 processors)	10B
Intel reserved	11B

NOTE

See Chapter 20 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for information on identifying earlier IA-32 processors.

The Extended Family ID needs to be examined only when the Family ID is 0FH. Integrate the fields into a display using the following rule:

```
IF Family_ID ≠ 0FH
    THEN DisplayFamily = Family_ID;
    ELSE DisplayFamily = Extended_Family_ID + Family_ID;
FI;
(* Show DisplayFamily as HEX field. *)
```

The Extended Model ID needs to be examined only when the Family ID is 06H or 0FH. Integrate the field into a display using the following rule:

```
IF (Family_ID = 06H or Family_ID = 0FH)
    THEN DisplayModel = (Extended_Model_ID « 4) + Model_ID;
    (* Right justify and zero-extend 4-bit field; display Model_ID as HEX field.*)
    ELSE DisplayModel = Model_ID;
FI;
(* Show DisplayModel as HEX field. *)
```

INPUT EAX = 01H: Returns Additional Information in EBX

When CPUID executes with EAX set to 01H, additional information is returned to the EBX register:

- Brand index (low byte of EBX) — this number provides an entry into a brand string table that contains brand strings for IA-32 processors. More information about this field is provided later in this section.
- CLFLUSH instruction cache line size (second byte of EBX) — this number indicates the size of the cache line flushed by the CLFLUSH and CLFLUSHOPT instructions in 8-byte increments. This field was introduced in the Pentium 4 processor.
- Local APIC ID (high byte of EBX) — this number is the 8-bit ID that is assigned to the local APIC on the processor during power up. This field was introduced in the Pentium 4 processor.

INPUT EAX = 01H: Returns Feature Information in ECX and EDX

When CPUID executes with EAX set to 01H, feature information is returned in ECX and EDX.

- Figure 3-7 and Table 3-10 show encodings for ECX.
- Figure 3-8 and Table 3-11 show encodings for EDX.

For all feature flags, a 1 indicates that the feature is supported. Use Intel to properly interpret feature flags.

NOTE

Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

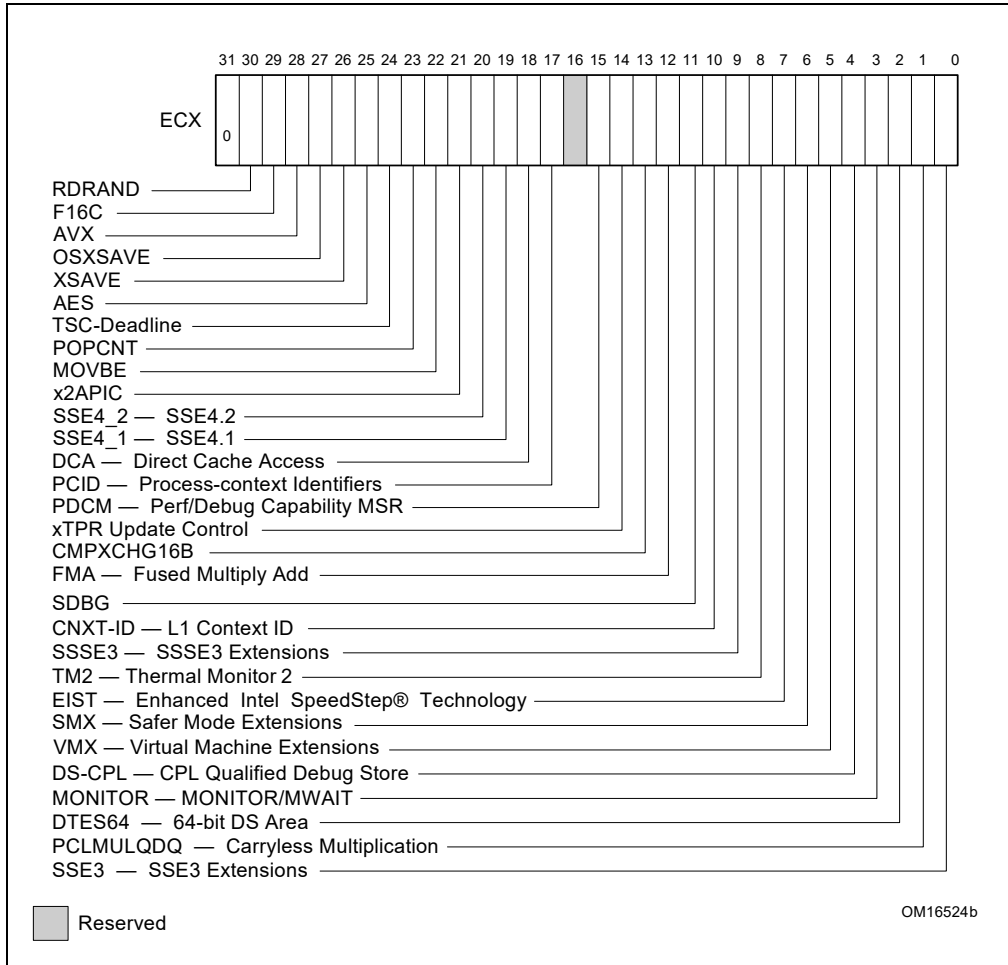


Figure 3-7. Feature Information Returned in the ECX Register

Table 3-10. Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	Streaming SIMD Extensions 3 (SSE3). A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	PCLMULQDQ. A value of 1 indicates the processor supports the PCLMULQDQ instruction.
2	DTES64	64-bit DS Area. A value of 1 indicates the processor supports DS area using 64-bit layout.
3	MONITOR	MONITOR/MWAIT. A value of 1 indicates the processor supports this feature.
4	DS-CPL	CPL Qualified Debug Store. A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	Virtual Machine Extensions. A value of 1 indicates that the processor supports this technology.
6	SMX	Safer Mode Extensions. A value of 1 indicates that the processor supports this technology. See Chapter 7, “Safer Mode Extensions Reference”.
7	EIST	Enhanced Intel SpeedStep® technology. A value of 1 indicates that the processor supports this technology.
8	TM2	Thermal Monitor 2. A value of 1 indicates whether the processor supports this technology.
9	SSSE3	A value of 1 indicates the presence of the Supplemental Streaming SIMD Extensions 3 (SSSE3). A value of 0 indicates the instruction extensions are not present in the processor.

Table 3-10. Feature Information Returned in the ECX Register (Contd.)

Bit #	Mnemonic	Description
10	CNXT-ID	L1 Context ID. A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
11	SDBG	A value of 1 indicates the processor supports IA32_DEBUG_INTERFACE MSR for silicon debug.
12	FMA	A value of 1 indicates the processor supports FMA extensions using YMM state.
13	CMPXCHG16B	CMPXCHG16B Available. A value of 1 indicates that the feature is available. See the “CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes” section in this chapter for a description.
14	xTPR Update Control	xTPR Update Control. A value of 1 indicates that the processor supports changing IA32_MISC_ENABLE[bit 23].
15	PDCM	Perfmon and Debug Capability: A value of 1 indicates the processor supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.
16	Reserved	Reserved
17	PCID	Process-context identifiers. A value of 1 indicates that the processor supports PCIDs and that software may set CR4.PCIDE to 1.
18	DCA	A value of 1 indicates the processor supports the ability to prefetch data from a memory mapped device.
19	SSE4_1	A value of 1 indicates that the processor supports SSE4.1.
20	SSE4_2	A value of 1 indicates that the processor supports SSE4.2.
21	x2APIC	A value of 1 indicates that the processor supports x2APIC feature.
22	MOVBE	A value of 1 indicates that the processor supports MOVBE instruction.
23	POPCNT	A value of 1 indicates that the processor supports the POPCNT instruction.
24	TSC-Deadline	A value of 1 indicates that the processor’s local APIC timer supports one-shot operation using a TSC deadline value.
25	AESNI	A value of 1 indicates that the processor supports the AESNI instruction extensions.
26	XSAVE	A value of 1 indicates that the processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCR0.
27	OSXSAVE	A value of 1 indicates that the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCR0 and to support processor extended state management using XSAVE/XRSTOR.
28	AVX	A value of 1 indicates the processor supports the AVX instruction extensions.
29	F16C	A value of 1 indicates that processor supports 16-bit floating-point conversion instructions.
30	RDRAND	A value of 1 indicates that processor supports RDRAND instruction.
31	Not Used	Always returns 0.

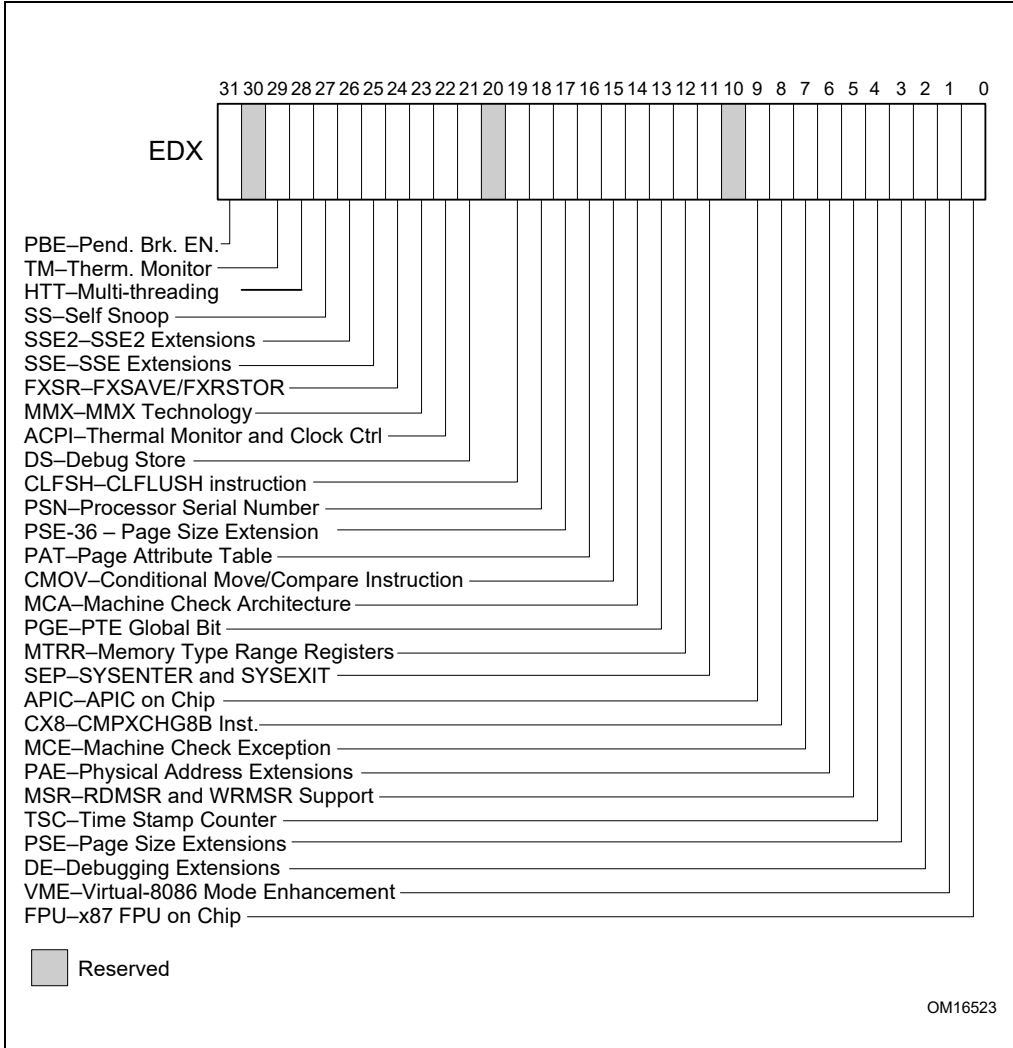


Figure 3-8. Feature Information Returned in the EDX Register

Table 3-11. More on Feature Information Returned in the EDX Register

Bit #	Mnemonic	Description
0	FPU	Floating-Point Unit On-Chip. The processor contains an x87 FPU.
1	VME	Virtual 8086 Mode Enhancements. Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.
2	DE	Debugging Extensions. Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	Page Size Extension. Large pages of size 4 MByte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.
4	TSC	Time Stamp Counter. The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	Model Specific Registers RDMSR and WRMSR Instructions. The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	Physical Address Extension. Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1.
7	MCE	Machine Check Exception. Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.
8	CX8	CMPXCHG8B Instruction. The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	APIC On-Chip. The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).
10	Reserved	Reserved
11	SEP	SYSENTER and SYSEXIT Instructions. The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	Memory Type Range Registers. MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.
13	PGE	Page Global Bit. The global bit is supported in paging-structure entries that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.
14	MCA	Machine Check Architecture. A value of 1 indicates the Machine Check Architecture of reporting machine errors is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	Conditional Move Instructions. The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported
16	PAT	Page Attribute Table. Page Attribute Table is supported. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory accessed through a linear address on a 4KB granularity.
17	PSE-36	36-Bit Page Size Extension. 4-MByte pages addressing physical memory beyond 4 GBytes are supported with 32-bit paging. This feature indicates that upper bits of the physical address of a 4-MByte page are encoded in bits 20:13 of the page-directory entry. Such physical addresses are limited by MAXPHYADDR and may be up to 40 bits in size.
18	PSN	Processor Serial Number. The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	CLFLUSH Instruction. CLFLUSH Instruction is supported.
20	Reserved	Reserved

Table 3-11. More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
21	DS	Debug Store. The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and processor event-based sampling (PEBS) facilities (see Chapter 24, "Introduction to Virtual-Machine Extensions," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C).
22	ACPI	Thermal Monitor and Software Controlled Clock Facilities. The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	Intel MMX Technology. The processor supports the Intel MMX technology.
24	FXSR	FXSAVE and FXRSTOR Instructions. The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating-point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	SSE. The processor supports the SSE extensions.
26	SSE2	SSE2. The processor supports the SSE2 extensions.
27	SS	Self Snoop. The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	Max APIC IDs reserved field is Valid. A value of 0 for HTT indicates there is only a single logical processor in the package and software should assume only a single APIC ID is reserved. A value of 1 for HTT indicates the value in CPUID.1.EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package) is valid for the package.
29	TM	Thermal Monitor. The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved
31	PBE	Pending Break Enable. The processor supports the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt.

INPUT EAX = 02H: TLB/Cache/Prefetch Information Returned in EAX, EBX, ECX, EDX

When CPUID executes with EAX set to 02H, the processor returns information about the processor's internal TLBs, cache, and prefetch hardware in the EAX, EBX, ECX, and EDX registers. The information is reported in encoded form and fall into the following categories:

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors. There are four types of encoding values for the byte descriptor, the encoding type is noted in the second column of Table 3-12. Table 3-12 lists the encoding of these descriptors. Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache, prefetch, or TLB types. The descriptors may appear in any order. Note also a processor may report a general descriptor type (FFH) and not report any byte descriptor of "cache type" via CPUID leaf 2.

Table 3-12. Encoding of CPUID Leaf 2 Descriptors

Value	Type	Description
00H	General	Null descriptor, this byte contains no information
01H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries
02H	TLB	Instruction TLB: 4 MByte pages, fully associative, 2 entries
03H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 64 entries
04H	TLB	Data TLB: 4 MByte pages, 4-way set associative, 8 entries
05H	TLB	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries
06H	Cache	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size
08H	Cache	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size
09H	Cache	1st-level instruction cache: 32KBytes, 4-way set associative, 64 byte line size
0AH	Cache	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size
0BH	TLB	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries
0CH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size
0DH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 64 byte line size
0EH	Cache	1st-level data cache: 24 KBytes, 6-way set associative, 64 byte line size
1DH	Cache	2nd-level cache: 128 KBytes, 2-way set associative, 64 byte line size
21H	Cache	2nd-level cache: 256 KBytes, 8-way set associative, 64 byte line size
22H	Cache	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector
23H	Cache	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
24H	Cache	2nd-level cache: 1 MBytes, 16-way set associative, 64 byte line size
25H	Cache	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
29H	Cache	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
2CH	Cache	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size
30H	Cache	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size
40H	Cache	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache
41H	Cache	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size
42H	Cache	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size
43H	Cache	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size
44H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size
45H	Cache	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size
46H	Cache	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size
47H	Cache	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size
48H	Cache	2nd-level cache: 3MByte, 12-way set associative, 64 byte line size
49H	Cache	3rd-level cache: 4MB, 16-way set associative, 64-byte line size (Intel Xeon processor MP, Family 0FH, Model 06H); 2nd-level cache: 4 MByte, 16-way set associative, 64 byte line size
4AH	Cache	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size
4BH	Cache	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size
4CH	Cache	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries

Table 3-12. Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
50H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries
51H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries
52H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries
55H	TLB	Instruction TLB: 2-MByte or 4-MByte pages, fully associative, 7 entries
56H	TLB	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries
57H	TLB	Data TLB0: 4 KByte pages, 4-way associative, 16 entries
59H	TLB	Data TLB0: 4 KByte pages, fully associative, 16 entries
5AH	TLB	Data TLB0: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries
5BH	TLB	Data TLB: 4 KByte and 4 MByte pages, 64 entries
5CH	TLB	Data TLB: 4 KByte and 4 MByte pages, 128 entries
5DH	TLB	Data TLB: 4 KByte and 4 MByte pages, 256 entries
60H	Cache	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size
61H	TLB	Instruction TLB: 4 KByte pages, fully associative, 48 entries
63H	TLB	Data TLB: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries and a separate array with 1 GByte pages, 4-way set associative, 4 entries
64H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 512 entries
66H	Cache	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size
67H	Cache	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size
68H	Cache	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size
6AH	Cache	uTLB: 4 KByte pages, 8-way set associative, 64 entries
6BH	Cache	DTLB: 4 KByte pages, 8-way set associative, 256 entries
6CH	Cache	DTLB: 2M/4M pages, 8-way set associative, 128 entries
6DH	Cache	DTLB: 1 GByte pages, fully associative, 16 entries
70H	Cache	Trace cache: 12 K- μ op, 8-way set associative
71H	Cache	Trace cache: 16 K- μ op, 8-way set associative
72H	Cache	Trace cache: 32 K- μ op, 8-way set associative
76H	TLB	Instruction TLB: 2M/4M pages, fully associative, 8 entries
78H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 64byte line size
79H	Cache	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7AH	Cache	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7BH	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7CH	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector
7DH	Cache	2nd-level cache: 2 MByte, 8-way set associative, 64byte line size
7FH	Cache	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size
80H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64-byte line size
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size

Table 3-12. Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
A0H	DTLB	DTLB: 4k pages, fully associative, 32 entries
B0H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries
B1H	TLB	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries
B2H	TLB	Instruction TLB: 4KByte pages, 4-way set associative, 64 entries
B3H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 128 entries
B4H	TLB	Data TLB1: 4 KByte pages, 4-way associative, 256 entries
B5H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 64 entries
B6H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 128 entries
BAH	TLB	Data TLB1: 4 KByte pages, 4-way associative, 64 entries
C0H	TLB	Data TLB: 4 KByte and 4 MByte pages, 4-way associative, 8 entries
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries
C2H	DTLB	DTLB: 4 KByte/2 MByte pages, 4-way associative, 16 entries
C3H	STLB	Shared 2nd-Level TLB: 4 KByte /2 MByte pages, 6-way associative, 1536 entries. Also 1GByte pages, 4-way, 16 entries.
C4H	DTLB	DTLB: 2M/4M Byte pages, 4-way associative, 32 entries
CAH	STLB	Shared 2nd-Level TLB: 4 KByte pages, 4-way associative, 512 entries
D0H	Cache	3rd-level cache: 512 KByte, 4-way set associative, 64 byte line size
D1H	Cache	3rd-level cache: 1 MByte, 4-way set associative, 64 byte line size
D2H	Cache	3rd-level cache: 2 MByte, 4-way set associative, 64 byte line size
D6H	Cache	3rd-level cache: 1 MByte, 8-way set associative, 64 byte line size
D7H	Cache	3rd-level cache: 2 MByte, 8-way set associative, 64 byte line size
D8H	Cache	3rd-level cache: 4 MByte, 8-way set associative, 64 byte line size
DCH	Cache	3rd-level cache: 1.5 MByte, 12-way set associative, 64 byte line size
DDH	Cache	3rd-level cache: 3 MByte, 12-way set associative, 64 byte line size
DEH	Cache	3rd-level cache: 6 MByte, 12-way set associative, 64 byte line size
E2H	Cache	3rd-level cache: 2 MByte, 16-way set associative, 64 byte line size
E3H	Cache	3rd-level cache: 4 MByte, 16-way set associative, 64 byte line size
E4H	Cache	3rd-level cache: 8 MByte, 16-way set associative, 64 byte line size
EAH	Cache	3rd-level cache: 12MByte, 24-way set associative, 64 byte line size
EBH	Cache	3rd-level cache: 18MByte, 24-way set associative, 64 byte line size
ECH	Cache	3rd-level cache: 24MByte, 24-way set associative, 64 byte line size
F0H	Prefetch	64-Byte prefetching
F1H	Prefetch	128-Byte prefetching
FEH	General	CPUID leaf 2 does not report TLB descriptor information; use CPUID leaf 18H to query TLB and other address translation parameters.
FFH	General	CPUID leaf 2 does not report cache descriptor information, use CPUID leaf 4 to query cache parameters

Example 3-1. Example of Cache and TLB Interpretation

The first member of the family of Pentium 4 processors returns the following information about caches and TLBs when the CPUID executes with an input value of 2:

```
EAX    66 5B 50 01H
EBX    0H
ECX    0H
EDX    00 7A 70 00H
```

Which means:

- The least-significant byte (byte 0) of register EAX is set to 01H. This value should be ignored.
- The most-significant bit of all four registers (EAX, EBX, ECX, and EDX) is set to 0, indicating that each register contains valid 1-byte descriptors.
- Bytes 1, 2, and 3 of register EAX indicate that the processor has:
 - 50H - a 64-entry instruction TLB, for mapping 4-KByte and 2-MByte or 4-MByte pages.
 - 5BH - a 64-entry data TLB, for mapping 4-KByte and 4-MByte pages.
 - 66H - an 8-KByte 1st level data cache, 4-way set associative, with a 64-Byte cache line size.
- The descriptors in registers EBX and ECX are valid, but contain NULL descriptors.
- Bytes 0, 1, 2, and 3 of register EDX indicate that the processor has:
 - 00H - NULL descriptor.
 - 70H - Trace cache: 12 K-μop, 8-way set associative.
 - 7AH - a 256-KByte 2nd level cache, 8-way set associative, with a sectored, 64-byte cache line size.
 - 00H - NULL descriptor.

INPUT EAX = 04H: Returns Deterministic Cache Parameters for Each Level

When CPUID executes with EAX set to 04H and ECX contains an index value, the processor returns encoded data that describe a set of deterministic cache parameters (for the cache level associated with the input in ECX). Valid index values start from 0.

Software can enumerate the deterministic cache parameters for each level of the cache hierarchy starting with an index value of 0, until the parameters report the value associated with the cache type field is 0. The architecturally defined fields reported by deterministic cache parameters are documented in Table 3-8.

This Cache Size in Bytes

$$= (\text{Ways} + 1) * (\text{Partitions} + 1) * (\text{Line_Size} + 1) * (\text{Sets} + 1)$$

$$= (\text{EBX}[31:22] + 1) * (\text{EBX}[21:12] + 1) * (\text{EBX}[11:0] + 1) * (\text{ECX} + 1)$$

The CPUID leaf 04H also reports data that can be used to derive the topology of processor cores in a physical package. This information is constant for all valid index values. Software can query the raw data reported by executing CPUID with EAX=04H and ECX=0 and use it as part of the topology enumeration algorithm described in Chapter 9, “Multiple-Processor Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

INPUT EAX = 05H: Returns MONITOR and MWAIT Features

When CPUID executes with EAX set to 05H, the processor returns information about features available to MONITOR/MWAIT instructions. The MONITOR instruction is used for address-range monitoring in conjunction with MWAIT instruction. The MWAIT instruction optionally provides additional extensions for advanced power management. See Table 3-8.

INPUT EAX = 06H: Returns Thermal and Power Management Features

When CPUID executes with EAX set to 06H, the processor returns information about thermal and power management features. See Table 3-8.

INPUT EAX = 07H: Returns Structured Extended Feature Enumeration Information

When CPUID executes with EAX set to 07H and ECX = 0, the processor returns information about the maximum input value for sub-leaves that contain extended feature flags. See Table 3-8.

When CPUID executes with EAX set to 07H and the input value of ECX is invalid (see leaf 07H entry in Table 3-8), the processor returns 0 in EAX/EBX/ECX/EDX. In subleaf 0, EAX returns the maximum input value of the highest leaf 7 sub-leaf, and EBX, ECX & EDX contain information of extended feature flags.

INPUT EAX = 09H: Returns Direct Cache Access Information

When CPUID executes with EAX set to 09H, the processor returns information about Direct Cache Access capabilities. See Table 3-8.

INPUT EAX = 0AH: Returns Architectural Performance Monitoring Features

When CPUID executes with EAX set to 0AH, the processor returns information about support for architectural performance monitoring capabilities. Architectural performance monitoring is supported if the version ID (see Table 3-8) is greater than Pn 0. See Table 3-8.

For each version of architectural performance monitoring capability, software must enumerate this leaf to discover the programming facilities and the architectural performance events available in the processor. The details are described in Chapter 24, "Introduction to Virtual-Machine Extensions," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

INPUT EAX = 0BH: Returns Extended Topology Information

CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH.

When CPUID executes with EAX set to 0BH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 0BH by verifying (a) the highest leaf index supported by CPUID is $\geq 0BH$, and (b) CPUID.0BH:EBX[15:0] reports a non-zero value. See Table 3-8.

INPUT EAX = 0DH: Returns Processor Extended States Enumeration Information

When CPUID executes with EAX set to 0DH and ECX = 0, the processor returns information about the bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area. See Table 3-8.

When CPUID executes with EAX set to 0DH and ECX = n ($n > 1$, and is a valid sub-leaf index), the processor returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. See Table 3-8. Software can use the forward-extendable technique depicted below to query the valid sub-leaves and obtain size and offset information for each processor extended state save area:

```
For i = 2 to 62 // sub-leaf 1 is reserved
  IF (CPUID.(EAX=0DH, ECX=0):VECTOR[i] = 1 ) // VECTOR is the 64-bit value of EDX:EAX
    Execute CPUID.(EAX=0DH, ECX = i) to examine size and offset for sub-leaf i;
  FI;
```

INPUT EAX = 0FH: Returns Intel Resource Director Technology (Intel RDT) Monitoring Enumeration Information

When CPUID executes with EAX set to 0FH and ECX = 0, the processor returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type. See Table 3-8.

When CPUID executes with EAX set to 0FH and ECX = n ($n \geq 1$, and is a valid ResID), the processor returns information software can use to program IA32_PQR_ASSOC, IA32_QM_EVTSEL MSRs before reading QoS data from the IA32_QM_CTR MSR.

INPUT EAX = 10H: Returns Intel Resource Director Technology (Intel RDT) Allocation Enumeration Information

When CPUID executes with EAX set to 10H and ECX = 0, the processor returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type. See Table 3-8.

When CPUID executes with EAX set to 10H and ECX = n (n >= 1, and is a valid ResID), the processor returns information about available classes of service and range of QoS mask MSRs that software can use to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

INPUT EAX = 12H: Returns Intel SGX Enumeration Information

When CPUID executes with EAX set to 12H and ECX = 0H, the processor returns information about Intel SGX capabilities. See Table 3-8.

When CPUID executes with EAX set to 12H and ECX = 1H, the processor returns information about Intel SGX attributes. See Table 3-8.

When CPUID executes with EAX set to 12H and ECX = n (n > 1), the processor returns information about Intel SGX Enclave Page Cache. See Table 3-8.

INPUT EAX = 14H: Returns Intel Processor Trace Enumeration Information

When CPUID executes with EAX set to 14H and ECX = 0H, the processor returns information about Intel Processor Trace extensions. See Table 3-8.

When CPUID executes with EAX set to 14H and ECX = n (n > 0 and less than the number of non-zero bits in CPUID.(EAX=14H, ECX= 0H).EAX), the processor returns information about packet generation in Intel Processor Trace. See Table 3-8.

INPUT EAX = 15H: Returns Time Stamp Counter and Nominal Core Crystal Clock Information

When CPUID executes with EAX set to 15H and ECX = 0H, the processor returns information about Time Stamp Counter and Core Crystal Clock. See Table 3-8.

INPUT EAX = 16H: Returns Processor Frequency Information

When CPUID executes with EAX set to 16H, the processor returns information about Processor Frequency Information. See Table 3-8.

INPUT EAX = 17H: Returns System-On-Chip Information

When CPUID executes with EAX set to 17H, the processor returns information about the System-On-Chip Vendor Attribute Enumeration. See Table 3-8.

INPUT EAX = 18H: Returns Deterministic Address Translation Parameters Information

When CPUID executes with EAX set to 18H, the processor returns information about the Deterministic Address Translation Parameters. See Table 3-8.

INPUT EAX = 19H: Returns Key Locker Information

When CPUID executes with EAX set to 19H, the processor returns information about Key Locker. See Table 3-8.

INPUT EAX = 1AH: Returns Native Model ID Information

When CPUID executes with EAX set to 1AH, the processor returns information about Native Model Identification. See Table 3-8.

INPUT EAX = 1BH: Returns PCONFIG Information

When CPUID executes with EAX set to 1BH, the processor returns information about PCONFIG capabilities. This information is enumerated in sub-leaves selected by the value of ECX (starting with 0).

Each sub-leaf of CPUID function 1BH enumerates its **sub-leaf type** in EAX. If a sub-leaf type is 0, the sub-leaf is invalid and zero is returned in EBX, ECX, and EDX. In this case, all subsequent sub-leaves (selected by larger input values of ECX) are also invalid.

The only valid sub-leaf type currently defined is 1, indicating that the sub-leaf enumerates target identifiers for the PCONFIG instruction. Any non-zero value returned in EBX, ECX, or EDX indicates a valid target **identifier** of the PCONFIG instruction (any value of zero should be ignored). The only target identifier currently defined is 1, indicating MKTME. See the “PCONFIG—Platform Configuration” instruction in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B for more information.

INPUT EAX = 1CH: Returns Last Branch Record Information

When CPUID executes with EAX set to 1CH, the processor returns information about LBRs (the architectural feature). See Table 3-8.

INPUT EAX = 1DH: Returns Tile Information

When CPUID executes with EAX set to 1DH and ECX = 0H, the processor returns information about tile architecture. See Table 3-8.

When CPUID executes with EAX set to 1DH and ECX = 1H, the processor returns information about tile palette 1. See Table 3-8.

INPUT EAX = 1EH: Returns TMUL Information

When CPUID executes with EAX set to 1EH and ECX = 0H, the processor returns information about TMUL capabilities. See Table 3-8.

INPUT EAX = 1FH: Returns V2 Extended Topology Information

When CPUID executes with EAX set to 1FH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 1FH by verifying (a) the highest leaf index supported by CPUID is $\geq 1FH$, and (b) CPUID.1FH:EBX[15:0] reports a non-zero value. See Table 3-8.

INPUT EAX = 20H: Returns History Reset Information

When CPUID executes with EAX set to 20H, the processor returns information about History Reset. See Table 3-8.

METHODS FOR RETURNING BRANDING INFORMATION

Use the following techniques to access branding information:

1. Processor brand string method.
2. Processor brand index; this method uses a software supplied brand string table.

These two methods are discussed in the following sections. For methods that are available in early processors, see Section: “Identification of Earlier IA-32 Processors” in Chapter 20 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

The Processor Brand String Method

Figure 3-9 describes the algorithm used for detection of the brand string. Processor brand identification software should execute this algorithm on all Intel 64 and IA-32 processors.

This method (introduced with Pentium 4 processors) returns an ASCII brand identification string and the Processor Base frequency of the processor to the EAX, EBX, ECX, and EDX registers.

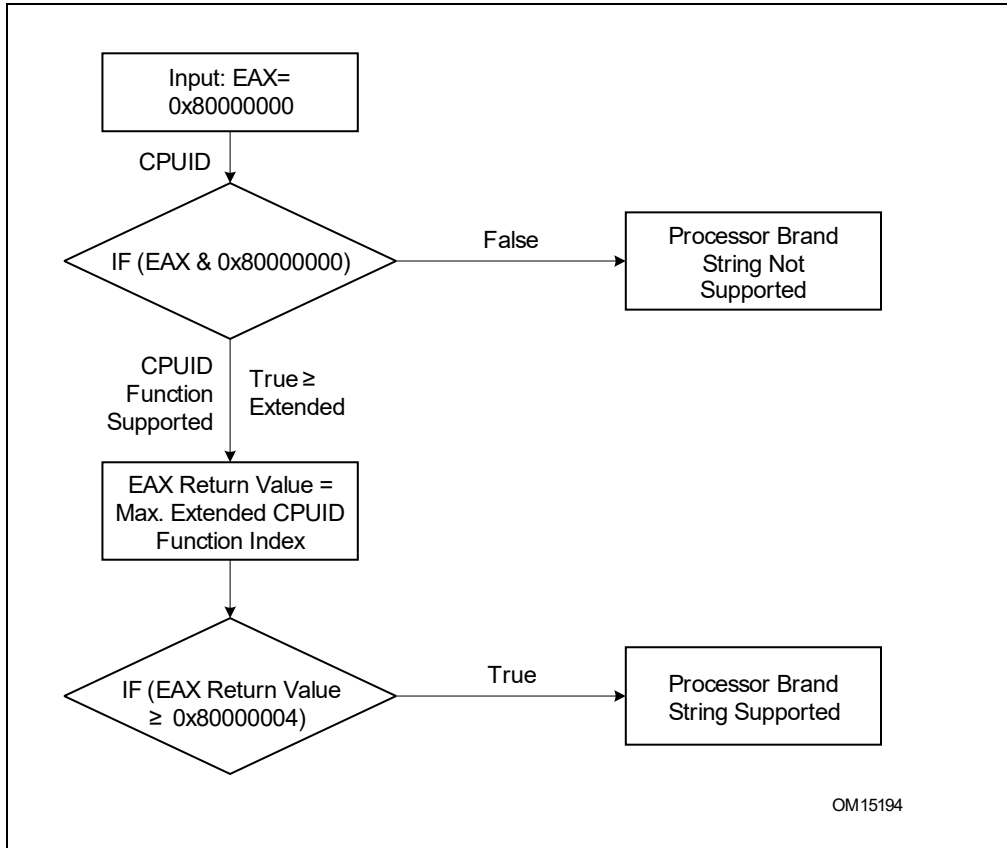


Figure 3-9. Determination of Support for the Processor Brand String

How Brand Strings Work

To use the brand string method, execute CPUID with EAX input of 8000002H through 80000004H. For each input value, CPUID returns 16 ASCII characters using EAX, EBX, ECX, and EDX. The returned string will be NULL-terminated.

Table 3-13 shows the brand string that is returned by the first processor in the Pentium 4 processor family.

Table 3-13. Processor Brand String Returned with Pentium 4 Processor

EAX Input Value	Return Values	ASCII Equivalent
80000002H	EAX = 20202020H EBX = 20202020H ECX = 20202020H EDX = 6E492020H	" " " " " " " " " "nl "
80000003H	EAX = 286C6574H EBX = 50202952H ECX = 69746E65H EDX = 52286D75H	"(let" "P)R" "itne" "R(mu"

Table 3-13. Processor Brand String Returned with Pentium 4 Processor (Contd.)

EAX Input Value	Return Values	ASCII Equivalent
80000004H	EAX = 20342029H EBX = 20555043H ECX = 30303531H EDX = 007A484DH	" 4)" " UPC" "0051" "\0zHM"

Extracting the Processor Frequency from Brand Strings

Figure 3-10 provides an algorithm which software can use to extract the Processor Base frequency from the processor brand string.

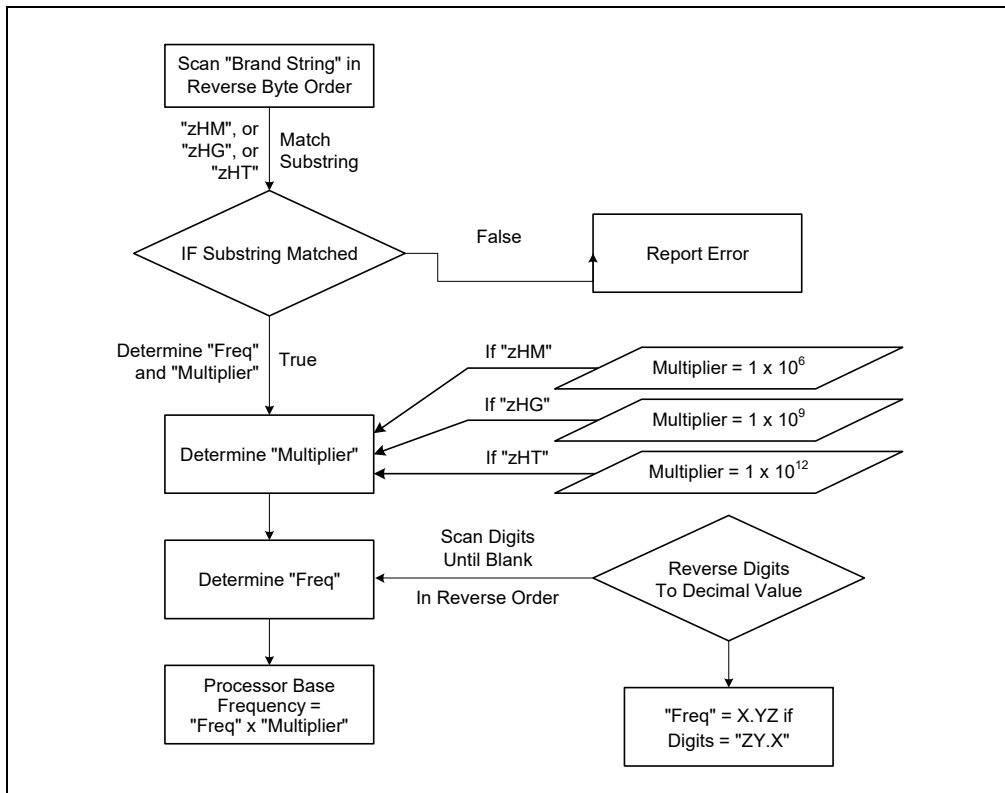


Figure 3-10. Algorithm for Extracting Processor Frequency

The Processor Brand Index Method

The brand index method (introduced with Pentium® III Xeon® processors) provides an entry point into a brand identification table that is maintained in memory by system software and is accessible from system- and user-level code. In this table, each brand index is associated with an ASCII brand identification string that identifies the official Intel family and model number of a processor.

When CPUID executes with EAX set to 1, the processor returns a brand index to the low byte in EBX. Software can then use this index to locate the brand identification string for the processor in the brand identification table. The first entry (brand index 0) in this table is reserved, allowing for backward compatibility with processors that do not support the brand identification feature. Starting with processor signature family ID = 0FH, model = 03H, brand index method is no longer supported. Use brand string method instead.

Table 3-14 shows brand indices that have identification strings associated with them.

Table 3-14. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

Brand Index	Brand String
00H	This processor does not support the brand identification feature
01H	Intel(R) Celeron(R) processor ¹
02H	Intel(R) Pentium(R) III processor ¹
03H	Intel(R) Pentium(R) III Xeon(R) processor; If processor signature = 000006B1h, then Intel(R) Celeron(R) processor
04H	Intel(R) Pentium(R) III processor
06H	Mobile Intel(R) Pentium(R) III processor-M
07H	Mobile Intel(R) Celeron(R) processor ¹
08H	Intel(R) Pentium(R) 4 processor
09H	Intel(R) Pentium(R) 4 processor
0AH	Intel(R) Celeron(R) processor ¹
0BH	Intel(R) Xeon(R) processor; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor MP
0CH	Intel(R) Xeon(R) processor MP
0EH	Mobile Intel(R) Pentium(R) 4 processor-M; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor
0FH	Mobile Intel(R) Celeron(R) processor ¹
11H	Mobile Genuine Intel(R) processor
12H	Intel(R) Celeron(R) M processor
13H	Mobile Intel(R) Celeron(R) processor ¹
14H	Intel(R) Celeron(R) processor
15H	Mobile Genuine Intel(R) processor
16H	Intel(R) Pentium(R) M processor
17H	Mobile Intel(R) Celeron(R) processor ¹
18H – 0FFH	RESERVED

NOTES:

1. Indicates versions of these processors that were introduced after the Pentium III

IA-32 Architecture Compatibility

CPUID is not supported in early models of the Intel486 processor or in any IA-32 processor earlier than the Intel486 processor.

Operation

IA32_BIOS_SIGN_ID MSR := Update with installed microcode revision number;

CASE (EAX) OF

EAX = 0:

EAX := Highest basic function input value understood by CPUID;

EBX := Vendor identification string;

EDX := Vendor identification string;

ECX := Vendor identification string;

BREAK;

EAX = 1H:

EAX[3:0] := Stepping ID;

EAX[7:4] := Model;

EAX[11:8] := Family;

EAX[13:12] := Processor type;
 EAX[15:14] := Reserved;
 EAX[19:16] := Extended Model;
 EAX[27:20] := Extended Family;
 EAX[31:28] := Reserved;
 EBX[7:0] := Brand Index; (* Reserved if the value is zero. *)
 EBX[15:8] := CLFLUSH Line Size;
 EBX[16:23] := Reserved; (* Number of threads enabled = 2 if MT enable fuse set. *)
 EBX[24:31] := Initial APIC ID;
 ECX := Feature flags; (* See Figure 3-7. *)
 EDX := Feature flags; (* See Figure 3-8. *)

BREAK;

EAX = 2H:

EAX := Cache and TLB information;
 EBX := Cache and TLB information;
 ECX := Cache and TLB information;
 EDX := Cache and TLB information;

BREAK;

EAX = 3H:

EAX := Reserved;
 EBX := Reserved;
 ECX := ProcessorSerialNumber[31:0];
 (* Pentium III processors only, otherwise reserved. *)
 EDX := ProcessorSerialNumber[63:32];
 (* Pentium III processors only, otherwise reserved. *)

BREAK

EAX = 4H:

EAX := Deterministic Cache Parameters Leaf; (* See Table 3-8. *)
 EBX := Deterministic Cache Parameters Leaf;
 ECX := Deterministic Cache Parameters Leaf;
 EDX := Deterministic Cache Parameters Leaf;

BREAK;

EAX = 5H:

EAX := MONITOR/MWAIT Leaf; (* See Table 3-8. *)
 EBX := MONITOR/MWAIT Leaf;
 ECX := MONITOR/MWAIT Leaf;
 EDX := MONITOR/MWAIT Leaf;

BREAK;

EAX = 6H:

EAX := Thermal and Power Management Leaf; (* See Table 3-8. *)
 EBX := Thermal and Power Management Leaf;
 ECX := Thermal and Power Management Leaf;
 EDX := Thermal and Power Management Leaf;

BREAK;

EAX = 7H:

EAX := Structured Extended Feature Flags Enumeration Leaf; (* See Table 3-8. *)
 EBX := Structured Extended Feature Flags Enumeration Leaf;
 ECX := Structured Extended Feature Flags Enumeration Leaf;
 EDX := Structured Extended Feature Flags Enumeration Leaf;

BREAK;

EAX = 8H:

EAX := Reserved = 0;
 EBX := Reserved = 0;
 ECX := Reserved = 0;

```

    EDX := Reserved = 0;
BREAK;
EAX = 9H:
    EAX := Direct Cache Access Information Leaf; (* See Table 3-8. *)
    EBX := Direct Cache Access Information Leaf;
    ECX := Direct Cache Access Information Leaf;
    EDX := Direct Cache Access Information Leaf;
BREAK;
EAX = AH:
    EAX := Architectural Performance Monitoring Leaf; (* See Table 3-8. *)
    EBX := Architectural Performance Monitoring Leaf;
    ECX := Architectural Performance Monitoring Leaf;
    EDX := Architectural Performance Monitoring Leaf;
BREAK;
EAX = BH:
    EAX := Extended Topology Enumeration Leaf; (* See Table 3-8. *)
    EBX := Extended Topology Enumeration Leaf;
    ECX := Extended Topology Enumeration Leaf;
    EDX := Extended Topology Enumeration Leaf;
BREAK;
EAX = CH:
    EAX := Reserved = 0;
    EBX := Reserved = 0;
    ECX := Reserved = 0;
    EDX := Reserved = 0;
BREAK;
EAX = DH:
    EAX := Processor Extended State Enumeration Leaf; (* See Table 3-8. *)
    EBX := Processor Extended State Enumeration Leaf;
    ECX := Processor Extended State Enumeration Leaf;
    EDX := Processor Extended State Enumeration Leaf;
BREAK;
EAX = EH:
    EAX := Reserved = 0;
    EBX := Reserved = 0;
    ECX := Reserved = 0;
    EDX := Reserved = 0;
BREAK;
EAX = FH:
    EAX := Intel Resource Director Technology Monitoring Enumeration Leaf; (* See Table 3-8. *)
    EBX := Intel Resource Director Technology Monitoring Enumeration Leaf;
    ECX := Intel Resource Director Technology Monitoring Enumeration Leaf;
    EDX := Intel Resource Director Technology Monitoring Enumeration Leaf;
BREAK;
EAX = 10H:
    EAX := Intel Resource Director Technology Allocation Enumeration Leaf; (* See Table 3-8. *)
    EBX := Intel Resource Director Technology Allocation Enumeration Leaf;
    ECX := Intel Resource Director Technology Allocation Enumeration Leaf;
    EDX := Intel Resource Director Technology Allocation Enumeration Leaf;
BREAK;
EAX = 12H:
    EAX := Intel SGX Enumeration Leaf; (* See Table 3-8. *)
    EBX := Intel SGX Enumeration Leaf;
    ECX := Intel SGX Enumeration Leaf;

```

EDX := Intel SGX Enumeration Leaf;
BREAK;
EAX = 14H:
EAX := Intel Processor Trace Enumeration Leaf; (* See Table 3-8. *)
EBX := Intel Processor Trace Enumeration Leaf;
ECX := Intel Processor Trace Enumeration Leaf;
EDX := Intel Processor Trace Enumeration Leaf;
BREAK;
EAX = 15H:
EAX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf; (* See Table 3-8. *)
EBX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf;
ECX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf;
EDX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf;
BREAK;
EAX = 16H:
EAX := Processor Frequency Information Enumeration Leaf; (* See Table 3-8. *)
EBX := Processor Frequency Information Enumeration Leaf;
ECX := Processor Frequency Information Enumeration Leaf;
EDX := Processor Frequency Information Enumeration Leaf;
BREAK;
EAX = 17H:
EAX := System-On-Chip Vendor Attribute Enumeration Leaf; (* See Table 3-8. *)
EBX := System-On-Chip Vendor Attribute Enumeration Leaf;
ECX := System-On-Chip Vendor Attribute Enumeration Leaf;
EDX := System-On-Chip Vendor Attribute Enumeration Leaf;
BREAK;
EAX = 18H:
EAX := Deterministic Address Translation Parameters Enumeration Leaf; (* See Table 3-8. *)
EBX := Deterministic Address Translation Parameters Enumeration Leaf;
ECX := Deterministic Address Translation Parameters Enumeration Leaf;
EDX := Deterministic Address Translation Parameters Enumeration Leaf;
BREAK;
EAX = 19H:
EAX := Key Locker Enumeration Leaf; (* See Table 3-8. *)
EBX := Key Locker Enumeration Leaf;
ECX := Key Locker Enumeration Leaf;
EDX := Key Locker Enumeration Leaf;
BREAK;
EAX = 1AH:
EAX := Native Model ID Enumeration Leaf; (* See Table 3-8. *)
EBX := Native Model ID Enumeration Leaf;
ECX := Native Model ID Enumeration Leaf;
EDX := Native Model ID Enumeration Leaf;
BREAK;
EAX = 1BH:
EAX := PCONFIG Information Enumeration Leaf; (* See "INPUT EAX = 1BH: Returns PCONFIG Information" on page 3-252. *)
EBX := PCONFIG Information Enumeration Leaf;
ECX := PCONFIG Information Enumeration Leaf;
EDX := PCONFIG Information Enumeration Leaf;
BREAK;
EAX = 1CH:
EAX := Last Branch Record Information Enumeration Leaf; (* See Table 3-8. *)
EBX := Last Branch Record Information Enumeration Leaf;
ECX := Last Branch Record Information Enumeration Leaf;

EDX := Last Branch Record Information Enumeration Leaf;
 BREAK;
 EAX = 1DH:
 EAX := Tile Information Enumeration Leaf; (* See Table 3-8. *)
 EBX := Tile Information Enumeration Leaf;
 ECX := Tile Information Enumeration Leaf;
 EDX := Tile Information Enumeration Leaf;
 BREAK;
 EAX = 1EH:
 EAX := TMUL Information Enumeration Leaf; (* See Table 3-8. *)
 EBX := TMUL Information Enumeration Leaf;
 ECX := TMUL Information Enumeration Leaf;
 EDX := TMUL Information Enumeration Leaf;
 BREAK;
 EAX = 1FH:
 EAX := V2 Extended Topology Enumeration Leaf; (* See Table 3-8. *)
 EBX := V2 Extended Topology Enumeration Leaf;
 ECX := V2 Extended Topology Enumeration Leaf;
 EDX := V2 Extended Topology Enumeration Leaf;
 BREAK;
 EAX = 20H:
 EAX := Processor History Reset Sub-leaf; (* See Table 3-8. *)
 EBX := Processor History Reset Sub-leaf;
 ECX := Processor History Reset Sub-leaf;
 EDX := Processor History Reset Sub-leaf;
 BREAK;
 EAX = 80000000H:
 EAX := Highest extended function input value understood by CPUID;
 EBX := Reserved;
 ECX := Reserved;
 EDX := Reserved;
 BREAK;
 EAX = 80000001H:
 EAX := Reserved;
 EBX := Reserved;
 ECX := Extended Feature Bits (* See Table 3-8.*);
 EDX := Extended Feature Bits (* See Table 3-8. *);
 BREAK;
 EAX = 80000002H:
 EAX := Processor Brand String;
 EBX := Processor Brand String, continued;
 ECX := Processor Brand String, continued;
 EDX := Processor Brand String, continued;
 BREAK;
 EAX = 80000003H:
 EAX := Processor Brand String, continued;
 EBX := Processor Brand String, continued;
 ECX := Processor Brand String, continued;
 EDX := Processor Brand String, continued;
 BREAK;
 EAX = 80000004H:
 EAX := Processor Brand String, continued;
 EBX := Processor Brand String, continued;
 ECX := Processor Brand String, continued;

```

    EDX := Processor Brand String, continued;
BREAK;
EAX = 80000005H:
    EAX := Reserved = 0;
    EBX := Reserved = 0;
    ECX := Reserved = 0;
    EDX := Reserved = 0;
BREAK;
EAX = 80000006H:
    EAX := Reserved = 0;
    EBX := Reserved = 0;
    ECX := Cache information;
    EDX := Reserved = 0;
BREAK;
EAX = 80000007H:
    EAX := Reserved = 0;
    EBX := Reserved = 0;
    ECX := Reserved = 0;
    EDX := Reserved = Misc Feature Flags;
BREAK;
EAX = 80000008H:
    EAX := Reserved = Physical Address Size Information;
    EBX := Reserved = Virtual Address Size Information;
    ECX := Reserved = 0;
    EDX := Reserved = 0;
BREAK;
EAX >= 40000000H and EAX <= 4FFFFFFFH:
DEFAULT: (* EAX = Value outside of recognized range for CPUID. *)
    (* If the highest basic information leaf data depend on ECX input value, ECX is honored. *)
    EAX := Reserved; (* Information returned for highest basic information leaf. *)
    EBX := Reserved; (* Information returned for highest basic information leaf. *)
    ECX := Reserved; (* Information returned for highest basic information leaf. *)
    EDX := Reserved; (* Information returned for highest basic information leaf. *)
BREAK;
ESAC;

```

Flags Affected

None.

Exceptions (All Operating Modes)

#UD	If the LOCK prefix is used. In earlier IA-32 processors that do not support the CPUID instruction, execution of the instruction results in an invalid opcode (#UD) exception being generated.
-----	--

ENQCMD—Enqueue Command

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 0F 38 F8 !{(11);rrr;bbb ENQCMD r32/r64, m512	A	V/V	ENQCMD	Atomically enqueue 64-byte user command from source memory operand to destination offset in ES segment specified in register operand as offset in ES segment.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

The ENQCMD instruction allows software to write commands to **enqueue registers**, which are special device registers accessed using memory-mapped I/O (MMIO).

Enqueue registers expect writes to have the following format:

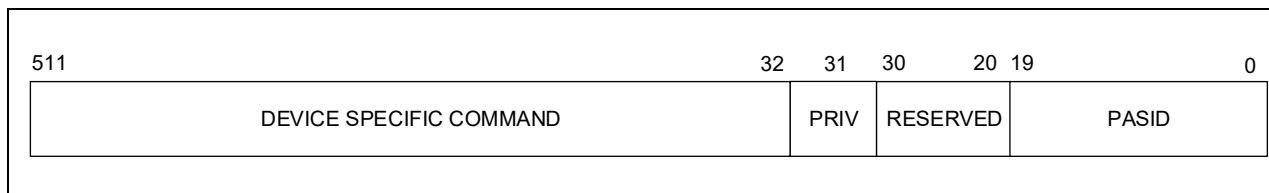


Figure 3-16. 64-Byte Data Written to Enqueue Registers

Bits 19:0 convey the process address space identifier (PASID), a value which system software may assign to individual software threads. Bit 31 contains privilege identification (0 = user; 1 = supervisor). Devices implementing enqueue registers may use these two values along with a device-specific command in the upper 60 bytes.

The ENQCMD instruction begins by reading 64 bytes of command data from its source memory operand. This is an ordinary load with cacheability and memory ordering implied normally by the memory type. The source operand need not be aligned, and there is no guarantee that all 64 bytes are loaded atomically. Bits 31:0 of the source operand must be zero.

The instruction then formats those 64 bytes into **command data** with a format consistent with that given in Figure 3-16:

- Command[19:0] get IA32_PASID[19:0].¹
- Command[30:20] are zero.
- Command[31] is 0 (indicating user; this value is used regardless of CPL).
- Command[511:32] get bits 511:32 of the source operand that was read from memory.

The ENQCMD instruction uses an **enqueue store** (defined below) to write this command data to the destination operand. The address of the destination operand is specified in a general-purpose register as an offset into the ES segment (the segment cannot be overridden).² The destination linear address must be 64-byte aligned. The operation of an enqueue store disregards the memory type of the destination memory address.

1. It is expected that system software will load the IA32_PASID MSR so that bits 19:0 contain the PASID of the current software thread. The MSR's valid bit, IA32_PASID[31], must be 1. For additional details on the IA32_PASID MSR, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

2. In 64-bit mode, the width of the register operand is 64 bits (32 bits with a 67H prefix). Outside 64-bit mode when CS.D = 1, the width is 32 bits (16 bits with a 67H prefix). Outside 64-bit mode when CS.D=0, the width is 16 bits (32 bits with a 67H prefix).

An enqueue store is not ordered relative to older stores to WB or WC memory (including non-temporal stores) or to executions of the CLFLUSHOPT or CLWB (when applied to addresses other than that of the enqueue store). Software can enforce such ordering by executing a fencing instruction such as SFENCE or MFENCE before the enqueue store.

An enqueue store does not write the data into the cache hierarchy, nor does it fetch any data into the cache hierarchy. An enqueue store's command data is never combined with that of any other store to the same address.

Unlike other stores, an enqueue store returns a status, which the ENQCMD instruction loads into the ZF flag in the RFLAGS register:

- ZF = 0 (success) reports that the 64-byte command data was written atomically to a device's enqueue register and has been accepted by the device. (It does not guarantee that the device has acted on the command; it may have queued it for later execution.)
- ZF = 1 (retry) reports that the command data was not accepted. This status is returned if the destination address is an enqueue register but the command was not accepted due to capacity or other temporal reasons. This status is also returned if the destination address was not an enqueue register (including the case of a memory address); in these cases, the store is dropped and is written neither to MMIO nor to memory.

Availability of the ENQCMD instruction is indicated by the presence of the CPUID feature flag ENQCMD (CPUID.(EAX=07H, ECX=0H):ECX[bit 29]).

Operation

```
IF IA32_PASID[31] = 0
    THEN #GP;
ELSE
    COMMAND := (SRC & ~FFFFFFFFH) | (IA32_PASID & FFFFFFFFH);
    DEST := COMMAND;
FI;
```

Intel C/C++ Compiler Intrinsic Equivalent

```
ENQCMD int_enqcmd(void *dst, const void *src)
```

Flags Affected

The ZF flag is set if the enqueue-store completion returns the retry status; otherwise it is cleared. All other flags are cleared.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If destination linear address is not aligned to a 64-byte boundary. If the PASID Valid field (bit 31) is 0 in IA32_PASID MSR. If bits 31:0 of the source operand are not all zero.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#UD	If CPUID.07H.0H:ECX.ENQCMD[bit 29] = 0. If the LOCK prefix is used.

Real-Address Mode Exceptions

- #GP
 - If any part of the operand lies outside the effective address space from 0 to FFFFH.
 - If destination linear address is not aligned to a 64-byte boundary.
 - If the PASID Valid field (bit 31) is 0 in IA32_PASID MSR.
 - If bits 31:0 of the source operand are not all zero.
- #UD
 - If CPUID.07H.0H:ECX.ENQCMD[bit 29] = 0.
 - If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real-address mode. Additionally:

- #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

- #SS(0) If a memory address referencing the SS segment is in non-canonical form.
- #GP(0)
 - If the memory address is in non-canonical form.
 - If destination linear address is not aligned to a 64-byte boundary.
 - If the PASID Valid field (bit 31) is 0 in IA32_PASID MSR.
 - If bits 31:0 of the source operand are not all zero.
- #PF(fault-code) For a page fault.
- #UD
 - If CPUID.07H.0H:ECX.ENQCMD[bit 29].
 - If the LOCK prefix is used.

FPREM—Partial Remainder

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 F8	FPREM	Valid	Valid	Replace ST(0) with the remainder obtained from dividing ST(0) by ST(1).

Description

Computes the remainder obtained from dividing the value in the ST(0) register (the dividend) by the value in the ST(1) register (the divisor or **modulus**), and stores the result in ST(0). The remainder represents the following value:

$$\text{Remainder} := \text{ST}(0) - (Q * \text{ST}(1))$$

Here, Q is an integer value that is obtained by truncating the floating-point number quotient of [ST(0) / ST(1)] toward zero. The sign of the remainder is the same as the sign of the dividend. The magnitude of the remainder is less than that of the modulus, unless a partial remainder was computed (as described below).

This instruction produces an exact result; the inexact-result exception does not occur and the rounding control has no effect. The following table shows the results obtained when computing the remainder of various classes of numbers, assuming that underflow does not occur.

Table 3-31. FPREM Results

		ST(1)						NaN
		$-\infty$	-F	-0	+0	+F	$+\infty$	
ST(0)	$-\infty$	*	*	*	*	*	*	NaN
	-F	ST(0)	-F or -0	*	*	-F or -0	ST(0)	NaN
	-0	-0	-0	*	*	-0	-0	NaN
	+0	+0	+0	*	*	+0	+0	NaN
	+F	ST(0)	+F or +0	*	*	+F or +0	ST(0)	NaN
	$+\infty$	*	*	*	*	*	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

NOTES:

F Means finite floating-point value.

* Indicates floating-point invalid-arithmetic-operand (#IA) exception.

When the result is 0, its sign is the same as that of the dividend. When the modulus is ∞ , the result is equal to the value in ST(0).

The FPREM instruction does not compute the remainder specified in IEEE Std 754. The IEEE specified remainder can be computed with the FPREM1 instruction. The FPREM instruction is provided for compatibility with the Intel 8087 and Intel287 math coprocessors.

The FPREM instruction gets its name “partial remainder” because of the way it computes the remainder. This instruction arrives at a remainder through iterative subtraction. It can, however, reduce the exponent of ST(0) by no more than 63 in one execution of the instruction. If the instruction succeeds in producing a remainder that is less than the modulus, the operation is complete and the C2 flag in the FPU status word is cleared. Otherwise, C2 is set, and the result in ST(0) is called the **partial remainder**. The exponent of the partial remainder will be less than the exponent of the original dividend by at least 32. Software can re-execute the instruction (using the partial remainder in ST(0) as the dividend) until C2 is cleared. (Note that while executing such a remainder-computation loop, a higher-priority interrupting routine that needs the FPU can force a context switch in-between the instructions in the loop.)

An important use of the FPREM instruction is to reduce the arguments of periodic functions. When reduction is complete, the instruction stores the three least-significant bits of the quotient in the C3, C1, and C0 flags of the FPU

status word. This information is important in argument reduction for the tangent function (using a modulus of $\pi/4$), because it locates the original angle in the correct one of eight sectors of the unit circle.

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

Operation

```
D := exponent(ST(0)) - exponent(ST(1));
IF D < 64
  THEN
    Q := Integer(TruncateTowardZero(ST(0) / ST(1)));
    ST(0) := ST(0) - (ST(1) * Q);
    C2 := 0;
    C0, C3, C1 := LeastSignificantBits(Q); (* Q2, Q1, Q0 *)
  ELSE
    C2 := 1;
    N := An implementation-dependent number between 32 and 63;
    QQ := Integer(TruncateTowardZero((ST(0) / ST(1)) / 2(D-N)));
    ST(0) := ST(0) - (ST(1) * QQ * 2(D-N));
FI;
```

FPU Flags Affected

C0	Set to bit 2 (Q2) of the quotient.
C1	Set to 0 if stack underflow occurred; otherwise, set to least significant bit of quotient (Q0).
C2	Set to 0 if reduction complete; set to 1 if incomplete.
C3	Set to bit 1 (Q1) of the quotient.

Floating-Point Exceptions

#IS	Stack underflow occurred.
#IA	Source operand is an SNaN value, modulus is 0, dividend is ∞ , or unsupported format.
#D	Source operand is a denormal value.
#U	Result is too small for destination format.

Protected Mode Exceptions

#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#MF	If there is a pending x87 FPU exception.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

FPREM1—Partial Remainder

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 F5	FPREM1	Valid	Valid	Replace ST(0) with the IEEE remainder obtained from dividing ST(0) by ST(1).

Description

Computes the IEEE remainder obtained from dividing the value in the ST(0) register (the dividend) by the value in the ST(1) register (the divisor or **modulus**), and stores the result in ST(0). The remainder represents the following value:

$$\text{Remainder} := \text{ST}(0) - (Q * \text{ST}(1))$$

Here, Q is an integer value that is obtained by rounding the floating-point number quotient of $[\text{ST}(0) / \text{ST}(1)]$ toward the nearest integer value. The magnitude of the remainder is less than or equal to half the magnitude of the modulus, unless a partial remainder was computed (as described below).

This instruction produces an exact result; the precision (inexact) exception does not occur and the rounding control has no effect. The following table shows the results obtained when computing the remainder of various classes of numbers, assuming that underflow does not occur.

Table 3-32. FPREM1 Results

		ST(1)						NaN
		$-\infty$	$-F$	-0	$+0$	$+F$	$+\infty$	
ST(0)	$-\infty$	*	*	*	*	*	*	NaN
	$-F$	ST(0)	$\pm F$ or -0	*	*	$\pm F$ or -0	ST(0)	NaN
	-0	-0	-0	*	*	-0	-0	NaN
	$+0$	$+0$	$+0$	*	*	$+0$	$+0$	NaN
	$+F$	ST(0)	$\pm F$ or $+0$	*	*	$\pm F$ or $+0$	ST(0)	NaN
	$+\infty$	*	*	*	*	*	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

NOTES:

F Means finite floating-point value.

* Indicates floating-point invalid-arithmetic-operand (#IA) exception.

When the result is 0, its sign is the same as that of the dividend. When the modulus is ∞ , the result is equal to the value in ST(0).

The FPREM1 instruction computes the remainder specified in IEEE Standard 754. This instruction operates differently from the FPREM instruction in the way that it rounds the quotient of ST(0) divided by ST(1) to an integer (see the "Operation" section below).

Like the FPREM instruction, FPREM1 computes the remainder through iterative subtraction, but can reduce the exponent of ST(0) by no more than 63 in one execution of the instruction. If the instruction succeeds in producing a remainder that is less than one half the modulus, the operation is complete and the C2 flag in the FPU status word is cleared. Otherwise, C2 is set, and the result in ST(0) is called the **partial remainder**. The exponent of the partial remainder will be less than the exponent of the original dividend by at least 32. Software can re-execute the instruction (using the partial remainder in ST(0) as the dividend) until C2 is cleared. (Note that while executing such a remainder-computation loop, a higher-priority interrupting routine that needs the FPU can force a context switch in-between the instructions in the loop.)

An important use of the FPREM1 instruction is to reduce the arguments of periodic functions. When reduction is complete, the instruction stores the three least-significant bits of the quotient in the C3, C1, and C0 flags of the FPU status word. This information is important in argument reduction for the tangent function (using a modulus of $\pi/4$), because it locates the original angle in the correct one of eight sectors of the unit circle.

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

Operation

```

D := exponent(ST(0)) - exponent(ST(1));
IF D < 64
  THEN
    Q := Integer(RoundTowardNearestInteger(ST(0) / ST(1)));
    ST(0) := ST(0) - (ST(1) * Q);
    C2 := 0;
    C0, C3, C1 := LeastSignificantBits(Q); (* Q2, Q1, Q0 *)
  ELSE
    C2 := 1;
    N := An implementation-dependent number between 32 and 63;
    QQ := Integer(TruncateTowardZero((ST(0) / ST(1)) / 2(D-N)));
    ST(0) := ST(0) - (ST(1) * QQ * 2(D-N));
FI;

```

FPU Flags Affected

C0	Set to bit 2 (Q2) of the quotient.
C1	Set to 0 if stack underflow occurred; otherwise, set to least significant bit of quotient (Q0).
C2	Set to 0 if reduction complete; set to 1 if incomplete.
C3	Set to bit 1 (Q1) of the quotient.

Floating-Point Exceptions

#IS	Stack underflow occurred.
#IA	Source operand is an SNaN value, modulus (divisor) is 0, dividend is ∞ , or unsupported format.
#D	Source operand is a denormal value.
#U	Result is too small for destination format.

Protected Mode Exceptions

#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#MF	If there is a pending x87 FPU exception.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

3. Updates to Chapter 4, Volume 2B

Change bars and green text show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-U*.

Changes to this chapter:

- Updated the PTWRITE instruction to add the CPUID feature flag to the instruction box.

PTWRITE—Write Data to a Processor Trace Packet

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 REX.W OF AE /4 PTWRITE r64/m64	RM	V/N.E	PTWRITE	Reads the data from r64/m64 to encode into a PTW packet if dependencies are met (see details below).
F3 OF AE /4 PTWRITE r32/m32	RM	V/V	PTWRITE	Reads the data from r32/m32 to encode into a PTW packet if dependencies are met (see details below).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:rm (r)	N/A	N/A	N/A

Description

This instruction reads data in the source operand and sends it to the Intel Processor Trace hardware to be encoded in a PTW packet if TriggerEn, ContextEn, FilterEn, and PTWEn are all set to 1. For more details on these values, see Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C, Section 33.2.2, “Software Trace Instrumentation with PTWRITE.” The size of data is 64-bit if using REX.W in 64-bit mode, otherwise 32-bits of data are copied from the source operand.

Note: The instruction will #UD if prefix 66H is used.

Operation

IF (IA32_RTIT_STATUS.TriggerEn & IA32_RTIT_STATUS.ContextEn & IA32_RTIT_STATUS.FilterEn & IA32_RTIT_CTL.PTWEn) = 1

PTW.PayloadBytes := Encoded payload size;

PTW.IP := IA32_RTIT_CTL.FUPonPTW

IF IA32_RTIT_CTL.FUPonPTW = 1

Insert FUP packet with IP of PTWRITE;

FI;

FI;

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.
#UD	If CPUID.(EAX=14H, ECX=0):EBX.PTWRITE [Bit 4] = 0. If LOCK prefix is used. If 66H prefix is used.

Real-Address Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If CPUID.(EAX=14H, ECX=0):EBX.PTWRITE [Bit 4] = 0. If LOCK prefix is used. If 66H prefix is used.

Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.
#UD	If CPUID.(EAX=14H, ECX=0):EBX.PTWRITE [Bit 4] = 0. If LOCK prefix is used. If 66H prefix is used.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If CPUID.(EAX=14H, ECX=0):EBX.PTWRITE [Bit 4] = 0. If LOCK prefix is used. If 66H prefix is used.

4. Updates to Chapter 5, Volume 2C

Change bars and green text show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference, V*.

Changes to this chapter:

- Updated the VFCMADDCPH/VMADDCPH instructions to correct a heading in the operation section.

VFCMADDCPH/VFMADDCPH—Complex Multiply and Accumulate FP16 Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.F2.MAP6.W0 56 /r VFCMADDCPH xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL	Complex multiply a pair of FP16 values from xmm2 and xmm3/m128/m32bcst, add to xmm1 and store the result in xmm1 subject to writemask k1.
EVEX.256.F2.MAP6.W0 56 /r VFCMADDCPH ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL	Complex multiply a pair of FP16 values from ymm2 and ymm3/m256/m32bcst, add to ymm1 and store the result in ymm1 subject to writemask k1.
EVEX.512.F2.MAP6.W0 56 /r VFCMADDCPH zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst {er}	A	V/V	AVX512-FP16	Complex multiply a pair of FP16 values from zmm2 and zmm3/m512/m32bcst, add to zmm1 and store the result in zmm1 subject to writemask k1.
EVEX.128.F3.MAP6.W0 56 /r VFMADDCPH xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL	Complex multiply a pair of FP16 values from xmm2 and the complex conjugate of xmm3/m128/m32bcst, add to xmm1 and store the result in xmm1 subject to writemask k1.
EVEX.256.F3.MAP6.W0 56 /r VFMADDCPH ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL	Complex multiply a pair of FP16 values from ymm2 and the complex conjugate of ymm3/m256/m32bcst, add to ymm1 and store the result in ymm1 subject to writemask k1.
EVEX.512.F3.MAP6.W0 56 /r VFMADDCPH zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst {er}	A	V/V	AVX512-FP16	Complex multiply a pair of FP16 values from zmm2 and the complex conjugate of zmm3/m512/m32bcst, add to zmm1 and store the result in zmm1 subject to writemask k1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (r, w)	VEX.vvvv (r)	ModRM:r/m (r)	N/A

Description

This instruction performs a complex multiply and accumulate operation. There are normal and complex conjugate forms of the operation.

The broadcasting and masking for this operation is done on 32-bit quantities representing a pair of FP16 values.

Rounding is performed at every FMA (fused multiply and add) boundary. Execution occurs as if all MXCSR exceptions are masked. MXCSR status bits are updated to reflect exceptional conditions.

Operation**VFMADDCPH dest{k1}, src1, src2 (AVX512)**

VL = 128, 256, 512

KL := VL / 32

FOR i := 0 to KL-1:

IF k1[i] or *no writemask*:

IF broadcasting and src2 is memory:

tsrc2.fp16[2*i+0] := src2.fp16[0]

tsrc2.fp16[2*i+1] := src2.fp16[1]

ELSE:

tsrc2.fp16[2*i+0] := src2.fp16[2*i+0]

tsrc2.fp16[2*i+1] := src2.fp16[2*i+1]

FOR i := 0 to KL-1:

IF k1[i] or *no writemask*:

tmp[2*i+0] := dest.fp16[2*i+0] + src1.fp16[2*i+0] * tsrc2.fp16[2*i+0]

tmp[2*i+1] := dest.fp16[2*i+1] + src1.fp16[2*i+1] * tsrc2.fp16[2*i+0]

FOR i := 0 to KL-1:

IF k1[i] or *no writemask*:

// non-conjugate version subtracts even term

dest.fp16[2*i+0] := tmp[2*i+0] - src1.fp16[2*i+1] * tsrc2.fp16[2*i+1]

dest.fp16[2*i+1] := tmp[2*i+1] + src1.fp16[2*i+0] * tsrc2.fp16[2*i+1]

ELSE IF *zeroing*:

dest.fp16[2*i+0] := 0

dest.fp16[2*i+1] := 0

DEST[MAXVL-1:VL] := 0

VFCMADDCPH dest{k1}, src1, src2 (AVX512)

VL = 128, 256, 512

KL := VL / 32

FOR i := 0 to KL-1:

IF k1[i] or *no writemask*:

IF broadcasting and src2 is memory:

tsrc2.fp16[2*i+0] := src2.fp16[0]

tsrc2.fp16[2*i+1] := src2.fp16[1]

ELSE:

tsrc2.fp16[2*i+0] := src2.fp16[2*i+0]

tsrc2.fp16[2*i+1] := src2.fp16[2*i+1]

FOR i := 0 to KL-1:

IF k1[i] or *no writemask*:

tmp[2*i+0] := dest.fp16[2*i+0] + src1.fp16[2*i+0] * tsrc2.fp16[2*i+0]

tmp[2*i+1] := dest.fp16[2*i+1] + src1.fp16[2*i+1] * tsrc2.fp16[2*i+0]

FOR i := 0 to KL-1:

IF k1[i] or *no writemask*:

// conjugate version subtracts odd final term

dest.fp16[2*i+0] := tmp[2*i+0] + src1.fp16[2*i+1] * tsrc2.fp16[2*i+1]

dest.fp16[2*i+1] := tmp[2*i+1] - src1.fp16[2*i+0] * tsrc2.fp16[2*i+1]

ELSE IF *zeroing*:

```
dest.fp16[2*i+0] := 0
dest.fp16[2*i+1] := 0
```

```
DEST[MAXVL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VFCMADDCPH __m128h __mm_fcmadd_pch (__m128h a, __m128h b, __m128h c);
VFCMADDCPH __m128h __mm_mask_fcmadd_pch (__m128h a, __mmask8 k, __m128h b, __m128h c);
VFCMADDCPH __m128h __mm_mask3_fcmadd_pch (__m128h a, __m128h b, __m128h c, __mmask8 k);
VFCMADDCPH __m128h __mm_maskz_fcmadd_pch (__mmask8 k, __m128h a, __m128h b, __m128h c);
VFCMADDCPH __m256h __mm256_fcmadd_pch (__m256h a, __m256h b, __m256h c);
VFCMADDCPH __m256h __mm256_mask_fcmadd_pch (__m256h a, __mmask8 k, __m256h b, __m256h c);
VFCMADDCPH __m256h __mm256_mask3_fcmadd_pch (__m256h a, __m256h b, __m256h c, __mmask8 k);
VFCMADDCPH __m256h __mm256_maskz_fcmadd_pch (__mmask8 k, __m256h a, __m256h b, __m256h c);
VFCMADDCPH __m512h __mm512_fcmadd_pch (__m512h a, __m512h b, __m512h c);
VFCMADDCPH __m512h __mm512_mask_fcmadd_pch (__m512h a, __mmask16 k, __m512h b, __m512h c);
VFCMADDCPH __m512h __mm512_mask3_fcmadd_pch (__m512h a, __m512h b, __m512h c, __mmask16 k);
VFCMADDCPH __m512h __mm512_maskz_fcmadd_pch (__mmask16 k, __m512h a, __m512h b, __m512h c);
VFCMADDCPH __m512h __mm512_fcmadd_round_pch (__m512h a, __m512h b, __m512h c, const int rounding);
VFCMADDCPH __m512h __mm512_mask_fcmadd_round_pch (__m512h a, __mmask16 k, __m512h b, __m512h c, const int rounding);
VFCMADDCPH __m512h __mm512_mask3_fcmadd_round_pch (__m512h a, __m512h b, __m512h c, __mmask16 k, const int rounding);
VFCMADDCPH __m512h __mm512_maskz_fcmadd_round_pch (__mmask16 k, __m512h a, __m512h b, __m512h c, const int rounding);

VFMADDCPH __m128h __mm_fmadd_pch (__m128h a, __m128h b, __m128h c);
VFMADDCPH __m128h __mm_mask_fmadd_pch (__m128h a, __mmask8 k, __m128h b, __m128h c);
VFMADDCPH __m128h __mm_mask3_fmadd_pch (__m128h a, __m128h b, __m128h c, __mmask8 k);
VFMADDCPH __m128h __mm_maskz_fmadd_pch (__mmask8 k, __m128h a, __m128h b, __m128h c);
VFMADDCPH __m256h __mm256_fmadd_pch (__m256h a, __m256h b, __m256h c);
VFMADDCPH __m256h __mm256_mask_fmadd_pch (__m256h a, __mmask8 k, __m256h b, __m256h c);
VFMADDCPH __m256h __mm256_mask3_fmadd_pch (__m256h a, __m256h b, __m256h c, __mmask8 k);
VFMADDCPH __m256h __mm256_maskz_fmadd_pch (__mmask8 k, __m256h a, __m256h b, __m256h c);
VFMADDCPH __m512h __mm512_fmadd_pch (__m512h a, __m512h b, __m512h c);
VFMADDCPH __m512h __mm512_mask_fmadd_pch (__m512h a, __mmask16 k, __m512h b, __m512h c);
VFMADDCPH __m512h __mm512_mask3_fmadd_pch (__m512h a, __m512h b, __m512h c, __mmask16 k);
VFMADDCPH __m512h __mm512_maskz_fmadd_pch (__mmask16 k, __m512h a, __m512h b, __m512h c);
VFMADDCPH __m512h __mm512_fmadd_round_pch (__m512h a, __m512h b, __m512h c, const int rounding);
VFMADDCPH __m512h __mm512_mask_fmadd_round_pch (__m512h a, __mmask16 k, __m512h b, __m512h c, const int rounding);
VFMADDCPH __m512h __mm512_mask3_fmadd_round_pch (__m512h a, __m512h b, __m512h c, __mmask16 k, const int rounding);
VFMADDCPH __m512h __mm512_maskz_fmadd_round_pch (__mmask16 k, __m512h a, __m512h b, __m512h c, const int rounding);
```

SIMD Floating-Point Exceptions

Invalid, Underflow, Overflow, Precision, Denormal.

Other Exceptions

EVEX-encoded instructions, see Table 2-49, “Type E4 Class Exception Conditions.”

Additionally:

#UD If (dest_reg == src1_reg) or (dest_reg == src2_reg).

5. Updates to Chapter 6, Volume 2D

Change bars and green text show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference, W-Z*.

Changes to this chapter:

- Updated the exceptions section for the XRSTORS instruction.

XRSTORS—Restore Processor Extended States Supervisor

Opcode / Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP OF C7 /3 XRSTORS mem	M	V/V	XSS	Restore state components specified by EDX:EAX from mem.
NP REX.W + OF C7 /3 XRSTORS64 mem	M	V/N.E.	XSS	Restore state components specified by EDX:EAX from mem.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	N/A	N/A	N/A

Description

Performs a full or partial restore of processor state components from the XSAVE area located at the memory address specified by the source operand. The implicit EDX:EAX register pair specifies a 64-bit instruction mask. The specific state components restored correspond to the bits set in the requested-feature bitmap (RFBM), which is the logical-AND of EDX:EAX and the logical-OR of XCR0 with the IA32_XSS MSR. XRSTORS may be executed only if CPL = 0.

The format of the XSAVE area is detailed in Section 13.4, “XSAVE Area,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1. Like FXRSTOR and FXSAVE, the memory format used for x87 state depends on a REX.W prefix; see Section 13.5.1, “x87 State” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

Section 13.12, “Operation of XRSTORS,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1 provides a detailed description of the operation of the XRSTOR instruction. The following items provide a high-level outline:

- Execution of XRSTORS is similar to that of the compacted form of XRSTOR; XRSTORS cannot restore from an XSAVE area in which the extended region is in the standard format (see Section 13.4.3, “Extended Region of an XSAVE Area” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).
- XRSTORS differs from XRSTOR in that it can restore state components corresponding to bits set in the IA32_XSS MSR.
- If RFBM[*i*] = 0, XRSTORS does not update state component *i*.
- If RFBM[*i*] = 1 and bit *i* is clear in the XSTATE_BV field in the XSAVE header, XRSTORS initializes state component *i*.
- If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 1, XRSTORS loads state component *i* from the XSAVE area.
- If XRSTORS attempts to load MXCSR with an illegal value, a general-protection exception (#GP) occurs.
- XRSTORS loads the internal value XRSTOR_INFO, which may be used to optimize a subsequent execution of XSAVEOPT or XSAVES.
- Immediately following an execution of XRSTORS, the processor tracks as in-use (not in initial configuration) any state component *i* for which RFBM[*i*] = 1 and XSTATE_BV[*i*] = 1; it tracks as modified any state component *i* for which RFBM[*i*] = 0.

Use of a source operand not aligned to 64-byte boundary (for 64-bit and 32-bit modes) results in a general-protection (#GP) exception. In 64-bit mode, the upper 32 bits of RDX and RAX are ignored.

See Section 13.6, “Processor Tracking of XSAVE-Managed State,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1 for discussion of the bitmaps XINUSE and XMODIFIED and of the quantity XRSTOR_INFO.

Operation

```

RFBM := (XCRO OR IA32_XSS) AND EDX:EAX;          /* bitwise logical OR and AND */
COMPMASK := XCOMP_BV field from XSAVE header;
RSTORMASK := XSTATE_BV field from XSAVE header;

FORMAT = COMPMASK AND 7FFFFFFF_FFFFFFFFH;
RESTORE_FEATURES = FORMAT AND RFBM;
TO_BE_RESTORED := RESTORE_FEATURES AND RSTORMASK;
FORCE_INIT := RFBM AND NOT FORMAT;
TO_BE_INITIALIZED = (RFBM AND NOT RSTORMASK) OR FORCE_INIT;

IF TO_BE_RESTORED[0] = 1
    THEN
        XINUSE[0] := 1;
        load x87 state from legacy region of XSAVE area;
    ELSIF TO_BE_INITIALIZED[0] = 1
        THEN
            XINUSE[0] := 0;
            initialize x87 state;
FI;

IF TO_BE_RESTORED[1] = 1
    THEN
        XINUSE[1] := 1;
        load SSE state from legacy region of XSAVE area; // this step loads the XMM registers and MXCSR
    ELSIF TO_BE_INITIALIZED[1] = 1
        THEN
            set all XMM registers to 0;
            XINUSE[1] := 0;
            MXCSR := 1F80H;
FI;

NEXT_FEATURE_OFFSET = 576;          // Legacy area and XSAVE header consume 576 bytes
FOR i := 2 TO 62
    IF FORMAT[i] = 1
        THEN
            IF TO_BE_RESTORED[i] = 1
                THEN
                    XINUSE[i] := 1;
                    load XSAVE state component i at offset NEXT_FEATURE_OFFSET from base of XSAVE area;
                FI;
                NEXT_FEATURE_OFFSET = NEXT_FEATURE_OFFSET + n (n enumerated by CPUID(EAX=0DH,ECX=i):EAX);
            FI;
        IF TO_BE_INITIALIZED[i] = 1
            THEN
                XINUSE[i] := 0;
                initialize XSAVE state component i;
            FI;
        ENDFOR;

XMODIFIED := NOT RFBM;

IF in VMX non-root operation
    THEN VMXNR := 1;

```

```

ELSE VMXNR := 0;
FI;
LAXA := linear address of XSAVE area;
XRSTOR_INFO := <CPL,VMXNR,LAXA,COMPMASK>;

```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

```

XRSTORS void _xrstors( void *, unsigned __int64);
XRSTORS64 void _xrstors64( void *, unsigned __int64);

```

Protected Mode Exceptions

#GP(0)	<p>If CPL > 0.</p> <p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If a memory operand is not aligned on a 64-byte boundary, regardless of segment.</p> <p>If bit 63 of the XCOMP_BV field of the XSAVE header is 0.</p> <p>If a bit in XCR0 IA32_XSS is 0 and the corresponding bit in the XCOMP_BV field of the XSAVE header is 1.</p> <p>If a bit in the XCOMP_BV field in the XSAVE header is 0 and the corresponding bit in the XSTATE_BV field is 1.</p> <p>If bytes 63:16 of the XSAVE header are not all zero.</p> <p>If attempting to write any reserved bits of the MXCSR register with 1.</p>
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	<p>If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSS[bit 3] = 0.</p> <p>If CR4.OSXSAVE[bit 18] = 0.</p> <p>If the LOCK prefix is used.</p>

Real-Address Mode Exceptions

#GP	<p>If a memory operand is not aligned on a 64-byte boundary, regardless of segment.</p> <p>If any part of the operand lies outside the effective address space from 0 to FFFFH.</p> <p>If bit 63 of the XCOMP_BV field of the XSAVE header is 0.</p> <p>If a bit in XCR0 IA32_XSS is 0 and the corresponding bit in the XCOMP_BV field of the XSAVE header is 1.</p> <p>If a bit in the XCOMP_BV field in the XSAVE header is 0 and the corresponding bit in the XSTATE_BV field is 1.</p> <p>If bytes 63:16 of the XSAVE header are not all zero.</p> <p>If attempting to write any reserved bits of the MXCSR register with 1.</p>
#NM	If CR0.TS[bit 3] = 1.
#UD	<p>If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSS[bit 3] = 0.</p> <p>If CR4.OSXSAVE[bit 18] = 0.</p> <p>If the LOCK prefix is used.</p>

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	<p>If $CPL > 0$.</p> <p>If a memory address is in a non-canonical form.</p> <p>If a memory operand is not aligned on a 64-byte boundary, regardless of segment.</p> <p>If bit 63 of the XCOMP_BV field of the XSAVE header is 0.</p> <p>If a bit in XCR0 IA32_XSS is 0 and the corresponding bit in the XCOMP_BV field of the XSAVE header is 1.</p> <p>If a bit in the XCOMP_BV field in the XSAVE header is 0 and the corresponding bit in the XSTATE_BV field is 1.</p> <p>If bytes 63:16 of the XSAVE header are not all zero.</p> <p>If attempting to write any reserved bits of the MXCSR register with 1.</p>
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#NM	If $CR0.TS[\text{bit } 3] = 1$.
#UD	<p>If $CPUID.01H:ECX.XSAVE[\text{bit } 26] = 0$ or $CPUID.(EAX=0DH,ECX=1):EAX.XSS[\text{bit } 3] = 0$.</p> <p>If $CR4.OSXSAVE[\text{bit } 18] = 0$.</p> <p>If the LOCK prefix is used.</p>

6. Updates to Chapter 9, Volume 3A

Change bars and green text show changes to Chapter 9 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Updated Section 9.1.2, "Bus Locking," and the sub-sections that follow. Added new sub-section 9.1.2.3, "Features to Disable Bus Locks."
- Updated Section 9.9.1, "Hierarchical Mapping of Shared Resources," to remove an inaccurate statement from the definition of "Tile," and wording and figure updates to match the CPUID topology enumeration updates.
- Updated Section 9.9.2, "Hierarchical Mapping of CPUID Extended Topology Leaf," with wording and figure changes to match the updated CPUID topology enumeration updates.
- Some terminology updates throughout the chapter where necessary.

The Intel 64 and IA-32 architectures provide mechanisms for managing and improving the performance of multiple processors connected to the same system bus. These include:

- Bus locking and/or cache coherency management for performing atomic operations on system memory.
- Serializing instructions.
- An advance programmable interrupt controller (APIC) located on the processor chip (see Chapter 11, “Advanced Programmable Interrupt Controller (APIC)”). This feature was introduced by the Pentium processor.
- A second-level cache (level 2, L2). For the Pentium 4, Intel Xeon, and P6 family processors, the L2 cache is included in the processor package and is tightly coupled to the processor. For the Pentium and Intel486 processors, pins are provided to support an external L2 cache.
- A third-level cache (level 3, L3). For Intel Xeon processors, the L3 cache is included in the processor package and is tightly coupled to the processor.
- Intel Hyper-Threading Technology. This extension to the Intel 64 and IA-32 architectures enables a single processor core to execute two or more threads concurrently (see Section 9.5, “Intel® Hyper-Threading Technology and Intel® Multi-Core Technology”).

These mechanisms are particularly useful in symmetric-multiprocessing (SMP) systems. However, they can also be used when an Intel 64 or IA-32 processor and a special-purpose processor (such as a communications, graphics, or video processor) share the system bus.

These multiprocessing mechanisms have the following characteristics:

- To maintain system memory coherency — When two or more processors are attempting simultaneously to access the same address in system memory, some communication mechanism or memory access protocol must be available to promote data coherency and, in some instances, to allow one processor to temporarily lock a memory location.
- To maintain cache consistency — When one processor accesses data cached on another processor, it must not receive incorrect data. If it modifies data, all other processors that access that data must receive the modified data.
- To allow predictable ordering of writes to memory — In some circumstances, it is important that memory writes be observed externally in precisely the same order as programmed.
- To distribute interrupt handling among a group of processors — When several processors are operating in a system in parallel, it is useful to have a centralized mechanism for receiving interrupts and distributing them to available processors for servicing.
- To increase system performance by exploiting the multi-threaded and multi-process nature of contemporary operating systems and applications.

The caching mechanism and cache consistency of Intel 64 and IA-32 processors are discussed in Chapter 12. The APIC architecture is described in Chapter 11. Bus and memory locking, serializing instructions, memory ordering, and Intel Hyper-Threading Technology are discussed in the following sections.

9.1 LOCKED ATOMIC OPERATIONS

The 32-bit IA-32 processors support locked atomic operations on locations in system memory. These operations are typically used to manage shared data structures (such as semaphores, segment descriptors, system segments, or page tables) in which two or more processors may try simultaneously to modify the same field or flag. The processor uses three interdependent mechanisms for carrying out locked atomic operations:

- Guaranteed atomic operations.
- Bus locking, using the LOCK# signal and the LOCK instruction prefix.

- Cache coherency protocols that ensure that atomic operations can be carried out on cached data structures (cache lock); this mechanism is present in the Pentium 4, Intel Xeon, and P6 family processors.

These mechanisms are interdependent in the following ways. Certain basic memory transactions (such as reading or writing a byte in system memory) are always guaranteed to be handled atomically. That is, once started, the processor guarantees that the operation will be completed before another processor or bus agent is allowed access to the memory location. The processor also supports bus locking for performing selected memory operations (such as a read-modify-write operation in a shared area of memory) that typically need to be handled atomically, but are not automatically handled this way. Because frequently used memory locations are often cached in a processor's L1 or L2 caches, atomic operations can often be carried out inside a processor's caches without asserting the bus lock. Here the processor's cache coherency protocols ensure that other processors that are caching the same memory locations are managed properly while atomic operations are performed on cached memory locations.

NOTE

Where there are contested lock accesses, software may need to implement algorithms that ensure fair access to resources in order to prevent lock starvation. The hardware provides no resource that guarantees fairness to participating agents. It is the responsibility of software to manage the fairness of semaphores and exclusive locking functions.

The mechanisms for handling locked atomic operations have evolved with the complexity of IA-32 processors. More recent IA-32 processors (such as the Pentium 4, Intel Xeon, and P6 family processors) and Intel 64 provide a more refined locking mechanism than earlier processors. These mechanisms are described in the following sections.

9.1.1 Guaranteed Atomic Operations

The Intel486 processor (and newer processors since) guarantees that the following basic memory operations will always be carried out atomically:

- Reading or writing a byte.
- Reading or writing a word aligned on a 16-bit boundary.
- Reading or writing a doubleword aligned on a 32-bit boundary.

The Pentium processor (and newer processors since) guarantees that the following additional memory operations will always be carried out atomically:

- Reading or writing a quadword aligned on a 64-bit boundary.
- 16-bit accesses to uncached memory locations that fit within a 32-bit data bus.

The P6 family processors (and newer processors since) guarantee that the following additional memory operation will always be carried out atomically:

- Unaligned 16-, 32-, and 64-bit accesses to cached memory that fit within a cache line.

Processors that enumerate support for Intel® AVX (by setting the feature flag CPUID.01H:ECX.AVX[bit 28]) guarantee that the 16-byte memory operations performed by the following instructions will always be carried out atomically:

- MOVAPD, MOVAPS, and MOVDQA.
- VMOVAPD, VMOVAPS, and VMOVDQA when encoded with VEX.128.
- VMOVAPD, VMOVAPS, VMOVDQA32, and VMOVDQA64 when encoded with EVEX.128 and k0 (masking disabled).

(Note that these instructions require the linear addresses of their memory operands to be 16-byte aligned.)

Accesses to cacheable memory that are split across cache lines and page boundaries are not guaranteed to be atomic by the Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4, Intel Xeon, and P6 family processors provide bus control signals that permit external memory subsystems to make split accesses atomic; however, nonaligned data accesses will seriously impact the performance of the processor and should be avoided.

Except as noted above, an x87 instruction or an SSE instruction that accesses data larger than a quadword may be implemented using multiple memory accesses. If such an instruction stores to memory, some of the accesses may complete (writing to memory) while another causes the operation to fault for architectural reasons (e.g., due an page-table entry that is marked “not present”). In this case, the effects of the completed accesses may be visible to software even though the overall instruction caused a fault. If TLB invalidation has been delayed (see Section 4.10.4.4), such page faults may occur even if all accesses are to the same page.

9.1.2 Bus Locking

Intel 64 and IA-32 processors provide a LOCK# signal that is asserted automatically during certain critical memory operations to lock the system bus or equivalent link. Assertion of this signal is called a **bus lock**. While this output signal is asserted, requests from other processors or bus agents for control of the bus are blocked. Software can specify other occasions when the LOCK semantics are to be followed by prepending the LOCK prefix to an instruction.

In the case of the Intel386, Intel486, and Pentium processors, explicitly locked instructions will result in the assertion of the LOCK# signal. It is the responsibility of the hardware designer to make the LOCK# signal available in system hardware to control memory accesses among processors.

For the P6 and more recent processor families, if the memory area being accessed is cached internally in the processor, the LOCK# signal is generally not asserted; instead, locking is only applied to the processor’s caches (see Section 9.1.4, “Effects of a LOCK Operation on Internal Processor Caches”). These processors will assert a bus lock for a locked access in either of the following situations: (1) the access is to multiple cache lines (a **split lock**); or (2) the access uses a memory type other than WB (a **UC lock**)¹.

9.1.2.1 Automatic Locking

The operations on which the processor automatically follows the LOCK semantics are as follows:

- When executing an XCHG instruction that references memory.
- When switching to a task, the processor tests and sets the busy flag in the type field of the TSS descriptor. To ensure that two processors do not switch to the same task simultaneously, the processor follows the LOCK semantics while testing and setting this flag.
- When loading a segment descriptor, the processor sets the accessed flag in the segment descriptor if the flag is clear. During this operation, the processor follows the LOCK semantics so that the descriptor will not be modified by another processor while it is being updated. For this action to be effective, operating-system procedures that update descriptors should use the following steps:
 - Use a locked operation to modify the access-rights byte to indicate that the segment descriptor is not-present, and specify a value for the type field that indicates that the descriptor is being updated.
 - Update the fields of the segment descriptor. (This operation may require several memory accesses; therefore, locked operations cannot be used.)
 - Use a locked operation to modify the access-rights byte to indicate that the segment descriptor is valid and present.
 - The Intel386 processor always updates the accessed flag in the segment descriptor, whether it is clear or not. The Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors only update this flag if it is not already set.
- The processor uses locked cycles to set the accessed and dirty flag in paging-structure entries.
- After an interrupt request, an interrupt controller may use the data bus to send the interrupt’s vector to the processor. The processor follows the LOCK semantics during this time to ensure that no other data appears on the data bus while the vector is being transmitted.

1. The term “UC lock” is used because the most common situation regards accesses to UC memory. Despite the name, locked accesses to WC, WP, and WT memory also cause bus locks.

9.1.2.2 Software Controlled Bus Locking

To explicitly force the LOCK semantics, software can use the LOCK prefix with the following instructions when they are used to modify a memory location. An invalid-opcode exception (#UD) is generated when the LOCK prefix is used with any other instruction or when no write operation is made to memory (that is, when the destination operand is in a register).

- The bit test and modify instructions (BTS, BTR, and BTC).
- The exchange instructions (XADD, CMPXCHG, CMPXCHG8B, and CMPXCHG16B).
- The LOCK prefix is automatically assumed for XCHG instruction.
- The following single-operand arithmetic and logical instructions: INC, DEC, NOT, and NEG.
- The following two-operand arithmetic and logical instructions: ADD, ADC, SUB, SBB, AND, OR, and XOR.

A locked instruction is guaranteed to lock only the area of memory defined by the destination operand, but may be interpreted by the system as a lock for a larger memory area.

Software should access semaphores (shared memory used for signalling between multiple processors) using identical addresses and operand lengths. For example, if one processor accesses a semaphore using a word access, other processors should not access the semaphore using a byte access.

NOTE

Do not implement semaphores using the WC memory type. Do not perform non-temporal stores to a cache line containing a location used to implement a semaphore.

The integrity of a bus lock is not affected by the alignment of the memory field. The LOCK semantics are followed for as many bus cycles as necessary to update the entire operand. However, it is recommended that locked accesses be aligned on their natural boundaries for better system performance:

- Any boundary for an 8-bit access (locked or otherwise).
- 16-bit boundary for locked word accesses.
- 32-bit boundary for locked doubleword accesses.
- 64-bit boundary for locked quadword accesses.

Locked operations are atomic with respect to all other memory operations and all externally visible events. Only instruction fetch and page table accesses can pass locked instructions. Locked instructions can be used to synchronize data written by one processor and read by another processor.

For the P6 family processors, locked operations serialize all outstanding load and store operations (that is, wait for them to complete). This rule is also true for the Pentium 4 and Intel Xeon processors, with one exception. Load operations that reference weakly ordered memory types (such as the WC memory type) may not be serialized.

Locked instructions should not be used to ensure that data written can be fetched as instructions.

NOTE

The locked instructions for the current versions of the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors allow data written to be fetched as instructions. However, Intel recommends that developers who require the use of self-modifying code use a different synchronizing mechanism, described in the following sections.

9.1.2.3 Features to Disable Bus Locks

Because bus locks may adversely affect performance in certain situations, processors may support two features that system software can use to disable bus locking. These are called **UC-lock disable** and **split-lock disable**.

A processor enumerates support for UC-lock disable by setting bit 4 of the IA32_CORE_CAPABILITIES MSR (MSR index CFH). Support for split-lock disable is enumerated by IA32_CORE_CAPABILITIES[5].

Software enables UC-lock disable by setting bit 28 of the MSR_MEMORY_CTRL MSR (MSR index 33H). When this bit is set, a locked access using a memory type other than WB causes a general-protection exception (#GP) with a zero error code. The locked access does not occur.

Software enables split-lock disable by setting MSR_MEMORY_CTRL[29]. When this bit is set, a locked access to multiple cache lines causes an alignment-check exception (#AC) with a zero error code.¹ The locked access does not occur.

While MSR_MEMORY_CTRL is not an architectural MSR, the behavior described above is consistent across processor models that enumerate the support in IA32_CORE_CAPABILITIES.

In addition to these features that disable bus locks, there are features that allow software to detect when a bus lock has occurred. See Section 18.3.1.6 for information about OS bus-lock detection and Section 26.2 for information about the VMM bus-lock detection.

9.1.3 Handling Self- and Cross-Modifying Code

The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. IA-32 processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified.

As processor microarchitectures become more complex and start to speculatively execute code ahead of the retirement point (as in P6 and more recent processor families), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future versions of the IA-32 architectures, use one of the following coding options:

(* OPTION 1 *)

```
Store modified code (as data) into code segment;
Jump to new code or an intermediate location;
Execute new code;
```

(* OPTION 2 *)

```
Store modified code (as data) into code segment;
Execute a serializing instruction; (* For example, CPUID instruction *)
Execute new code;
```

The use of one of these options is not required for programs intended to run on the Pentium or Intel486 processors, but are recommended to ensure compatibility with the P6 and more recent processor families.

Self-modifying code will execute at a lower level of performance than non-self-modifying or normal code. The degree of the performance deterioration will depend upon the frequency of modification and specific characteristics of the code.

The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code**. As with self-modifying code, IA-32 processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified.

To write cross-modifying code and ensure that it is compliant with current and future versions of the IA-32 architecture, the following processor synchronization algorithm must be implemented:

(* Action of Modifying Processor *)

```
Memory_Flag := 0; (* Set Memory_Flag to value other than 1 *)
Store modified code (as data) into code segment;
Memory_Flag := 1;
```

(* Action of Executing Processor *)

```
WHILE (Memory_Flag ≠ 1)
    Wait for code to update;
ELIHW;
Execute serializing instruction; (* For example, CPUID instruction *)
Begin executing modified code;
```

1. Other alignment-check exceptions occur only if CR0.AM = 1, EFLAGS.AC = 1, and CPL = 3. The alignment-check exceptions resulting from split-lock disable may occur even if CR0.AM = 0, EFLAGS.AC = 0, or CPL < 3.

(The use of this option is not required for programs intended to run on the Intel486 processor, but is recommended to ensure compatibility with the Pentium 4, Intel Xeon, P6 family, and Pentium processors.)

Like self-modifying code, cross-modifying code will execute at a lower level of performance than non-cross-modifying (normal) code, depending upon the frequency of modification and specific characteristics of the code.

The restrictions on self-modifying code and cross-modifying code also apply to the Intel 64 architecture.

9.1.4 Effects of a LOCK Operation on Internal Processor Caches

For the Intel486 and Pentium processors, the LOCK# signal is always asserted on the bus during a LOCK operation, even if the area of memory being locked is cached in the processor.

For the P6 and more recent processor families, if the area of memory being locked during a LOCK operation is cached in the processor that is performing the LOCK operation as write-back memory and is completely contained in a cache line, the processor may not assert the LOCK# signal on the bus. Instead, it will modify the memory location internally and allow its cache coherency mechanism to ensure that the operation is carried out atomically. This operation is called “cache locking.” The cache coherency mechanism automatically prevents two or more processors that have cached the same area of memory from simultaneously modifying data in that area.

9.2 MEMORY ORDERING

The term **memory ordering** refers to the order in which the processor issues reads (loads) and writes (stores) through the system bus to system memory. The Intel 64 and IA-32 architectures support several memory-ordering models depending on the implementation of the architecture. For example, the Intel386 processor enforces **program ordering** (generally referred to as **strong ordering**), where reads and writes are issued on the system bus in the order they occur in the instruction stream under all circumstances.

To allow performance optimization of instruction execution, the IA-32 architecture allows departures from strong-ordering model called **processor ordering** in Pentium 4, Intel Xeon, and P6 family processors. These **processor-ordering** variations (called here the **memory-ordering model**) allow performance enhancing operations such as allowing reads to go ahead of buffered writes. The goal of any of these variations is to increase instruction execution speeds, while maintaining memory coherency, even in multiple-processor systems.

Section 9.2.1 and Section 9.2.2 describe the memory-ordering implemented by Intel486, Pentium, Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, Intel Xeon, and P6 family processors. Section 9.2.3 gives examples illustrating the behavior of the memory-ordering model on IA-32 and Intel-64 processors. Section 9.2.4 considers the special treatment of stores for string operations and Section 9.2.5 discusses how memory-ordering behavior may be modified through the use of specific instructions.

9.2.1 Memory Ordering in the Intel® Pentium® and Intel486™ Processors

The Pentium and Intel486 processors follow the processor-ordered memory model; however, they operate as strongly-ordered processors under most circumstances. Reads and writes always appear in programmed order at the system bus—except for the following situation where processor ordering is exhibited. Read misses are permitted to go ahead of buffered writes on the system bus when all the buffered writes are cache hits and, therefore, are not directed to the same address being accessed by the read miss.

In the case of I/O operations, both reads and writes always appear in programmed order.

Software intended to operate correctly in processor-ordered processors (such as the Pentium 4, Intel Xeon, and P6 family processors) should not depend on the relatively strong ordering of the Pentium or Intel486 processors. Instead, it should ensure that accesses to shared variables that are intended to control concurrent execution among processors are explicitly required to obey program ordering through the use of appropriate locking or serializing operations (see Section 9.2.5, “Strengthening or Weakening the Memory-Ordering Model”).

9.2.2 Memory Ordering in P6 and More Recent Processor Families

The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, and P6 family processors also use a processor-ordered memory-ordering model that can be further defined as “write ordered with store-buffer forwarding.” This model can be characterized as follows.

In a single-processor system for memory regions defined as write-back cacheable, the memory-ordering model respects the following principles (**Note** the memory-ordering principles for single-processor and multiple-processor systems are written from the perspective of software executing on the processor, where the term “processor” refers to a logical processor. For example, a physical processor supporting multiple cores and/or Intel Hyper-Threading Technology is treated as a multi-processor systems.):

- Reads are not reordered with other reads.
- Writes are not reordered with older reads.
- Writes to memory are not reordered with other writes, with the following exceptions:
 - streaming stores (writes) executed with the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD); and
 - string operations (see Section 9.2.4.1).
- No write to memory may be reordered with an execution of the CLFLUSH instruction; a write may be reordered with an execution of the CLFLUSHOPT instruction that flushes a cache line other than the one being written.¹ Executions of the CLFLUSH instruction are not reordered with each other. Executions of CLFLUSHOPT that access different cache lines may be reordered with each other. An execution of CLFLUSHOPT may be reordered with an execution of CLFLUSH that accesses a different cache line.
- Reads may be reordered with older writes to different locations but not with older writes to the same location.
- Reads or writes cannot be reordered with I/O instructions, locked instructions, or serializing instructions.
- Reads cannot pass earlier LFENCE and MFENCE instructions.
- Writes and executions of CLFLUSH and CLFLUSHOPT cannot pass earlier LFENCE, SFENCE, and MFENCE instructions.
- LFENCE instructions cannot pass earlier reads.
- SFENCE instructions cannot pass earlier writes or executions of CLFLUSH and CLFLUSHOPT.
- MFENCE instructions cannot pass earlier reads, writes, or executions of CLFLUSH and CLFLUSHOPT.

In a multiple-processor system, the following ordering principles apply:

- Individual processors use the same ordering principles as in a single-processor system.
- Writes by a single processor are observed in the same order by all processors.
- Writes from an individual processor are NOT ordered with respect to the writes from other processors.
- Memory ordering obeys causality (memory ordering respects transitive visibility).
- Any two stores are seen in a consistent order by processors other than those performing the stores
- Locked instructions have a total order.

See the example in Figure 9-1. Consider three processors in a system and each processor performs three writes, one to each of three defined locations (A, B, and C). Individually, the processors perform the writes in the same program order, but because of bus arbitration and other memory access mechanisms, the order that the three processors write the individual memory locations can differ each time the respective code sequences are executed on the processors. The final values in location A, B, and C would possibly vary on each execution of the write sequence.

The processor-ordering model described in this section is virtually identical to that used by the Pentium and Intel486 processors. The only enhancements in the Pentium 4, Intel Xeon, and P6 family processors are:

- Added support for speculative reads, while still adhering to the ordering principles above.
- Store-buffer forwarding, when a read passes a write to the same memory location.

1. Earlier versions of this manual specified that writes to memory may be reordered with executions of the CLFLUSH instruction. No processors implementing the CLFLUSH instruction allow such reordering.

- Out of order store from long string store and string move operations (see Section 9.2.4, “Fast-String Operation and Out-of-Order Stores,” below).

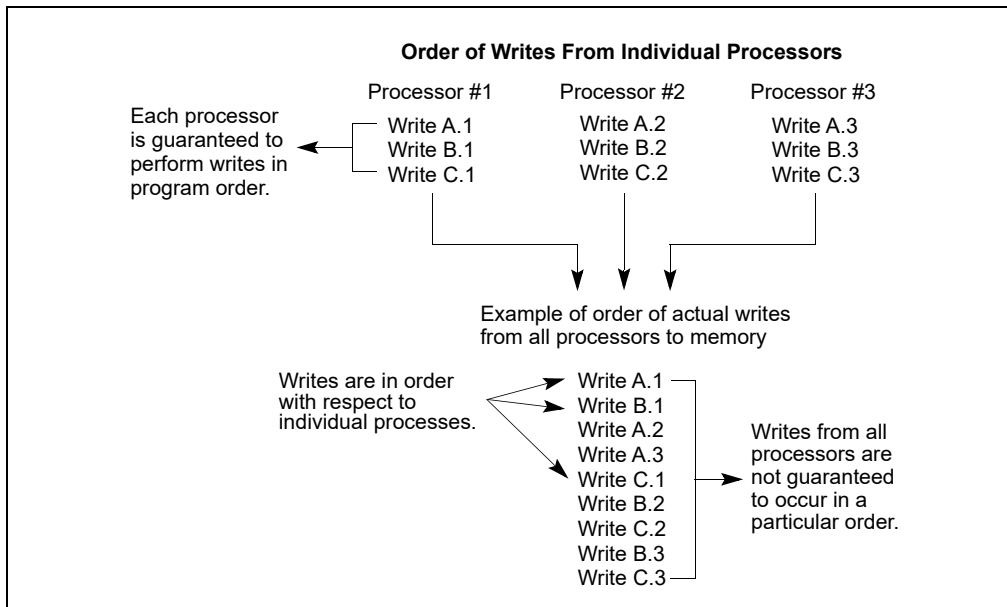


Figure 9-1. Example of Write Ordering in Multiple-Processor Systems

NOTE

In P6 processor family, store-buffer forwarding to reads of WC memory from streaming stores to the same address does not occur due to errata.

9.2.3 Examples Illustrating the Memory-Ordering Principles

This section provides a set of examples that illustrate the behavior of the memory-ordering principles introduced in Section 9.2.2. They are designed to give software writers an understanding of how memory ordering may affect the results of different sequences of instructions.

These examples are limited to accesses to memory regions defined as write-back cacheable (WB). (Section 9.2.3.1 describes other limitations on the generality of the examples.) The reader should understand that they describe only software-visible behavior. A logical processor may reorder two accesses even if one of examples indicates that they may not be reordered. Such an example states only that software cannot detect that such a reordering occurred. Similarly, a logical processor may execute a memory access more than once as long as the behavior visible to software is consistent with a single execution of the memory access.

9.2.3.1 Assumptions, Terminology, and Notation

As noted above, the examples in this section are limited to accesses to memory regions defined as write-back cacheable (WB). They apply only to ordinary loads stores and to locked read-modify-write instructions. They do not necessarily apply to any of the following: out-of-order stores for string instructions (see Section 9.2.4); accesses with a non-temporal hint; reads from memory by the processor as part of address translation (e.g., page walks); and updates to segmentation and paging structures by the processor (e.g., to update “accessed” bits).

The principles underlying the examples in this section apply to individual memory accesses and to locked read-modify-write instructions. The Intel-64 memory-ordering model guarantees that, for each of the following memory-access instructions, the constituent memory operation appears to execute as a single memory access:

- Instructions that read or write a single byte.
- Instructions that read or write a word (2 bytes) whose address is aligned on a 2 byte boundary.

- Instructions that read or write a doubleword (4 bytes) whose address is aligned on a 4 byte boundary.
- Instructions that read or write a quadword (8 bytes) whose address is aligned on an 8 byte boundary.

Any locked instruction (either the XCHG instruction or another read-modify-write instruction with a LOCK prefix) appears to execute as an indivisible and uninterruptible sequence of load(s) followed by store(s) regardless of alignment.

Other instructions may be implemented with multiple memory accesses. From a memory-ordering point of view, there are no guarantees regarding the relative order in which the constituent memory accesses are made. There is also no guarantee that the constituent operations of a store are executed in the same order as the constituent operations of a load.

Section 9.2.3.2 through Section 9.2.3.7 give examples using the MOV instruction. The principles that underlie these examples apply to load and store accesses in general and to other instructions that load from or store to memory. Section 9.2.3.8 and Section 9.2.3.9 give examples using the XCHG instruction. The principles that underlie these examples apply to other locked read-modify-write instructions.

This section uses the term “processor” is to refer to a logical processor. The examples are written using Intel-64 assembly-language syntax and use the following notational conventions:

- Arguments beginning with an “r”, such as r1 or r2 refer to registers (e.g., EAX) visible only to the processor being considered.
- Memory locations are denoted with x, y, z.
- Stores are written as *mov [_x], val*, which implies that *val* is being stored into the memory location x.
- Loads are written as *mov r, [_x]*, which implies that the contents of the memory location x are being loaded into the register r.

As noted earlier, the examples refer only to software visible behavior. When the succeeding sections make statement such as “the two stores are reordered,” the implication is only that “the two stores appear to be reordered from the point of view of software.”

9.2.3.2 Neither Loads Nor Stores Are Reordered with Like Operations

The Intel-64 memory-ordering model allows neither loads nor stores to be reordered with the same kind of operation. That is, it ensures that loads are seen in program order and that stores are seen in program order. This is illustrated by the following example:

Example 9-1. Stores Are Not Reordered with Other Stores

Processor 0	Processor 1
<i>mov [_x], 1</i>	<i>mov r1, [_y]</i>
<i>mov [_y], 1</i>	<i>mov r2, [_x]</i>
Initially x = y = 0	
r1 = 1 and r2 = 0 is not allowed	

The disallowed return values could be exhibited only if processor 0’s two stores are reordered (with the two loads occurring between them) or if processor 1’s two loads are reordered (with the two stores occurring between them).

If r1 = 1, the store to y occurs before the load from y. Because the Intel-64 memory-ordering model does not allow stores to be reordered, the earlier store to x occurs before the load from y. Because the Intel-64 memory-ordering model does not allow loads to be reordered, the store to x also occurs before the later load from x. This r2 = 1.

9.2.3.3 Stores Are Not Reordered With Earlier Loads

The Intel-64 memory-ordering model ensures that a store by a processor may not occur before a previous load by the same processor. This is illustrated in Example 9-2.

Example 9-2. Stores Are Not Reordered with Older Loads

Processor 0	Processor 1
mov r1, [_x] mov [_y], 1	mov r2, [_y] mov [_x], 1
Initially x = y = 0 r1 = 1 and r2 = 1 is not allowed	

Assume r1 = 1.

- Because r1 = 1, processor 1’s store to x occurs before processor 0’s load from x.
- Because the Intel-64 memory-ordering model prevents each store from being reordered with the earlier load by the same processor, processor 1’s load from y occurs before its store to x.
- Similarly, processor 0’s load from x occurs before its store to y.
- Thus, processor 1’s load from y occurs before processor 0’s store to y, implying r2 = 0.

9.2.3.4 Loads May Be Reordered with Earlier Stores to Different Locations

The Intel-64 memory-ordering model allows a load to be reordered with an earlier store to a different location. However, loads are not reordered with stores to the same location.

The fact that a load may be reordered with an earlier store to a different location is illustrated by the following example:

Example 9-3. Loads May be Reordered with Older Stores

Processor 0	Processor 1
mov [_x], 1 mov r1, [_y]	mov [_y], 1 mov r2, [_x]
Initially x = y = 0 r1 = 0 and r2 = 0 is allowed	

At each processor, the load and the store are to different locations and hence may be reordered. Any interleaving of the operations is thus allowed. One such interleaving has the two loads occurring before the two stores. This would result in each load returning value 0.

The fact that a load may not be reordered with an earlier store to the same location is illustrated by the following example:

Example 9-4. Loads Are not Reordered with Older Stores to the Same Location

Processor 0
mov [_x], 1 mov r1, [_x]
Initially x = 0 r1 = 0 is not allowed

The Intel-64 memory-ordering model does not allow the load to be reordered with the earlier store because the accesses are to the same location. Therefore, r1 = 1 must hold.

9.2.3.5 Intra-Processor Forwarding Is Allowed

The memory-ordering model allows concurrent stores by two processors to be seen in different orders by those two processors; specifically, each processor may perceive its own store occurring before that of the other. This is illustrated by the following example:

Example 9-5. Intra-Processor Forwarding is Allowed

Processor 0	Processor 1
mov [_x], 1 mov r1, [_x] mov r2, [_y]	mov [_y], 1 mov r3, [_y] mov r4, [_x]
Initially x = y = 0 r2 = 0 and r4 = 0 is allowed	

The memory-ordering model imposes no constraints on the order in which the two stores appear to execute by the two processors. This fact allows processor 0 to see its store before seeing processor 1's, while processor 1 sees its store before seeing processor 0's. (Each processor is self consistent.) This allows r2 = 0 and r4 = 0.

In practice, the reordering in this example can arise as a result of store-buffer forwarding. While a store is temporarily held in a processor's store buffer, it can satisfy the processor's own loads but is not visible to (and cannot satisfy) loads by other processors.

9.2.3.6 Stores Are Transitively Visible

The memory-ordering model ensures transitive visibility of stores; stores that are causally related appear to all processors to occur in an order consistent with the causality relation. This is illustrated by the following example:

Example 9-6. Stores Are Transitively Visible

Processor 0	Processor 1	Processor 2
mov [_x], 1	mov r1, [_x] mov [_y], 1	mov r2, [_y] mov r3, [_x]
Initially x = y = 0 r1 = 1, r2 = 1, r3 = 0 is not allowed		

Assume that r1 = 1 and r2 = 1.

- Because r1 = 1, processor 0's store occurs before processor 1's load.
- Because the memory-ordering model prevents a store from being reordered with an earlier load (see Section 9.2.3.3), processor 1's load occurs before its store. Thus, processor 0's store causally precedes processor 1's store.
- Because processor 0's store causally precedes processor 1's store, the memory-ordering model ensures that processor 0's store appears to occur before processor 1's store from the point of view of all processors.
- Because r2 = 1, processor 1's store occurs before processor 2's load.
- Because the Intel-64 memory-ordering model prevents loads from being reordered (see Section 9.2.3.2), processor 2's load occur in order.
- The above items imply that processor 0's store to x occurs before processor 2's load from x. This implies that r3 = 1.

9.2.3.7 Stores Are Seen in a Consistent Order by Other Processors

As noted in Section 9.2.3.5, the memory-ordering model allows stores by two processors to be seen in different orders by those two processors. However, any two stores must appear to execute in the same order to all processors other than those performing the stores. This is illustrated by the following example:

Example 9-7. Stores Are Seen in a Consistent Order by Other Processors

Processor 0	Processor 1	Processor 2	Processor 3
mov [_x], 1	mov [_y], 1	mov r1, [_x] mov r2, [_y]	mov r3, [_y] mov r4, [_x]
Initially x = y = 0 r1 = 1, r2 = 0, r3 = 1, r4 = 0 is not allowed			

By the principles discussed in Section 9.2.3.2:

- Processor 2’s first and second load cannot be reordered.
- Processor 3’s first and second load cannot be reordered.
- If r1 = 1 and r2 = 0, processor 0’s store appears to precede processor 1’s store with respect to processor 2.
- Similarly, r3 = 1 and r4 = 0 imply that processor 1’s store appears to precede processor 0’s store with respect to processor 1.

Because the memory-ordering model ensures that any two stores appear to execute in the same order to all processors (other than those performing the stores), this set of return values is not allowed.

9.2.3.8 Locked Instructions Have a Total Order

The memory-ordering model ensures that all processors agree on a single execution order of all locked instructions, including those that are larger than 8 bytes or are not naturally aligned. This is illustrated by the following example:

Example 9-8. Locked Instructions Have a Total Order

Processor 0	Processor 1	Processor 2	Processor 3
xchg [_x], r1	xchg [_y], r2	mov r3, [_x] mov r4, [_y]	mov r5, [_y] mov r6, [_x]
Initially r1 = r2 = 1, x = y = 0 r3 = 1, r4 = 0, r5 = 1, r6 = 0 is not allowed			

Processor 2 and processor 3 must agree on the order of the two executions of XCHG. Without loss of generality, suppose that processor 0’s XCHG occurs first.

- If r5 = 1, processor 1’s XCHG into y occurs before processor 3’s load from y.
- Because the Intel-64 memory-ordering model prevents loads from being reordered (see Section 9.2.3.2), processor 3’s loads occur in order and, therefore, processor 1’s XCHG occurs before processor 3’s load from x.
- Since processor 0’s XCHG into x occurs before processor 1’s XCHG (by assumption), it occurs before processor 3’s load from x. Thus, r6 = 1.

A similar argument (referring instead to processor 2’s loads) applies if processor 1’s XCHG occurs before processor 0’s XCHG.

9.2.3.9 Loads and Stores Are Not Reordered with Locked Instructions

The memory-ordering model prevents loads and stores from being reordered with locked instructions that execute earlier or later. The examples in this section illustrate only cases in which a locked instruction is executed before a

load or a store. The reader should note that reordering is prevented also if the locked instruction is executed after a load or a store.

The first example illustrates that loads may not be reordered with earlier locked instructions:

Example 9-9. Loads Are not Reordered with Locks

Processor 0	Processor 1
xchg [_x], r1 mov r2, [_y]	xchg [_y], r3 mov r4, [_x]
Initially x = y = 0, r1 = r3 = 1 r2 = 0 and r4 = 0 is not allowed	

As explained in Section 9.2.3.8, there is a total order of the executions of locked instructions. Without loss of generality, suppose that processor 0’s XCHG occurs first.

Because the Intel-64 memory-ordering model prevents processor 1’s load from being reordered with its earlier XCHG, processor 0’s XCHG occurs before processor 1’s load. This implies r4 = 1.

A similar argument (referring instead to processor 2’s accesses) applies if processor 1’s XCHG occurs before processor 0’s XCHG.

The second example illustrates that a store may not be reordered with an earlier locked instruction:

Example 9-10. Stores Are not Reordered with Locks

Processor 0	Processor 1
xchg [_x], r1 mov [_y], 1	mov r2, [_y] mov r3, [_x]
Initially x = y = 0, r1 = 1 r2 = 1 and r3 = 0 is not allowed	

Assume r2 = 1.

- Because r2 = 1, processor 0’s store to y occurs before processor 1’s load from y.
- Because the memory-ordering model prevents a store from being reordered with an earlier locked instruction, processor 0’s XCHG into x occurs before its store to y. Thus, processor 0’s XCHG into x occurs before processor 1’s load from y.
- Because the memory-ordering model prevents loads from being reordered (see Section 9.2.3.2), processor 1’s loads occur in order and, therefore, processor 1’s XCHG into x occurs before processor 1’s load from x. Thus, r3 = 1.

9.2.4 Fast-String Operation and Out-of-Order Stores

Section 7.3.9.3 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, described an optimization of repeated string operations called **fast-string operation**.

As explained in that section, the stores produced by fast-string operation may appear to execute out of order. Software dependent upon sequential store ordering should not use string operations for the entire data structure to be stored. Data and semaphores should be separated. Order-dependent code should write to a discrete semaphore variable after any string operations to allow correctly ordered data to be seen by all processors. Atomicity of load and store operations is guaranteed only for native data elements of the string with native data size, and only if they are included in a single cache line.

Section 9.2.4.1 and Section 9.2.4.2 provide further explain and examples.

9.2.4.1 Memory-Ordering Model for String Operations on Write-Back (WB) Memory

This section deals with the memory-ordering model for string operations on write-back (WB) memory for the Intel 64 architecture.

The memory-ordering model respects the follow principles:

1. Stores within a single string operation may be executed out of order.
2. Stores from separate string operations (for example, stores from consecutive string operations) do not execute out of order. All the stores from an earlier string operation will complete before any store from a later string operation.
3. String operations are not reordered with other store operations.

Fast string operations (e.g., string operations initiated with the MOVSB/STOSB instructions and the REP prefix) may be interrupted by exceptions or interrupts. The interrupts are precise but may be delayed - for example, the interruptions may be taken at cache line boundaries, after every few iterations of the loop, or after operating on every few bytes. Different implementations may choose different options, or may even choose not to delay interrupt handling, so software should not rely on the delay. When the interrupt/trap handler is reached, the source/destination registers point to the next string element to be operated on, while the EIP stored in the stack points to the string instruction, and the ECX register has the value it held following the last successful iteration. The return from that trap/interrupt handler should cause the string instruction to be resumed from the point where it was interrupted.

The string operation memory-ordering principles, (item 2 and 3 above) should be interpreted by taking the corruptibility of fast string operations into account. For example, if a fast string operation gets interrupted after k iterations, then stores performed by the interrupt handler will become visible after the fast string stores from iteration 0 to k, and before the fast string stores from the (k+1)th iteration onward.

Stores within a single string operation may execute out of order (item 1 above) only if fast string operation is enabled. Fast string operations are enabled/disabled through the IA32_MISC_ENABLE model specific register.

9.2.4.2 Examples Illustrating Memory-Ordering Principles for String Operations

The following examples uses the same notation and convention as described in Section 9.2.3.1.

In Example 9-11, processor 0 does one round of (128 iterations) doubleword string store operation via rep:stosd, writing the value 1 (value in EAX) into a block of 512 bytes from location `_x` (kept in ES:EDI) in ascending order. Since each operation stores a doubleword (4 bytes), the operation is repeated 128 times (value in ECX). The block of memory initially contained 0. Processor 1 is reading two memory locations that are part of the memory block being updated by processor 0, i.e, reading locations in the range `_x` to `(_x+511)`.

Example 9-11. Stores Within a String Operation May be Reordered

Processor 0	Processor 1
rep:stosd [<code>_x</code>]	mov r1, [<code>_z</code>] mov r2, [<code>_y</code>]
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = <code>_x</code> Initially [<code>_x</code>] to 511[<code>_x</code>]= 0, <code>_x</code> <= <code>_y</code> < <code>_z</code> < <code>_x</code> +512 r1 = 1 and r2 = 0 is allowed	

It is possible for processor 1 to perceive that the repeated string stores in processor 0 are happening out of order. Assume that fast string operations are enabled on processor 0.

In Example 9-12, processor 0 does two separate rounds of rep stosd operation of 128 doubleword stores, writing the value 1 (value in EAX) into the first block of 512 bytes from location `_x` (kept in ES:EDI) in ascending order. It then writes 1 into a second block of memory from `(_x+512)` to `(_x+1023)`. All of the memory locations initially contain 0. The block of memory initially contained 0. Processor 1 performs two load operations from the two blocks of memory.

Example 9-12. Stores Across String Operations Are not Reordered

Processor 0	Processor 1
rep:stosd [_x] mov ecx, \$128 rep:stosd 512[_x]	mov r1, [_z] mov r2, [_y]
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = _x Initially [_x] to 1023[_x]= 0, _x <= _y < _x+512 < _z < _x+1024 r1 = 1 and r2 = 0 is not allowed	

It is not possible in the above example for processor 1 to perceive any of the stores from the later string operation (to the second 512 block) in processor 0 before seeing the stores from the earlier string operation to the first 512 block.

The above example assumes that writes to the second block (_x+512 to _x+1023) does not get executed while processor 0's string operation to the first block has been interrupted. If the string operation to the first block by processor 0 is interrupted, and a write to the second memory block is executed by the interrupt handler, then that change in the second memory block will be visible before the string operation to the first memory block resumes.

In Example 9-13, processor 0 does one round of (128 iterations) doubleword string store operation via rep:stosd, writing the value 1 (value in EAX) into a block of 512 bytes from location _x (kept in ES:EDI) in ascending order. It then writes to a second memory location outside the memory block of the previous string operation. Processor 1 performs two read operations, the first read is from an address outside the 512-byte block but to be updated by processor 0, the second ready is from inside the block of memory of string operation.

Example 9-13. String Operations Are not Reordered with later Stores

Processor 0	Processor 1
rep:stosd [_x] mov [_z], \$1	mov r1, [_z] mov r2, [_y]
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = _x Initially [_y] = [_z] = 0, [_x] to 511[_x]= 0, _x <= _y < _x+512, _z is a separate memory location r1 = 1 and r2 = 0 is not allowed	

Processor 1 cannot perceive the later store by processor 0 until it sees all the stores from the string operation. Example 9-13 assumes that processor 0's store to [_z] is not executed while the string operation has been interrupted. If the string operation is interrupted and the store to [_z] by processor 0 is executed by the interrupt handler, then changes to [_z] will become visible before the string operation resumes.

Example 9-14 illustrates the visibility principle when a string operation is interrupted.

Example 9-14. Interrupted String Operation

Processor 0	Processor 1
rep:stosd [_x] // interrupted before es:edi reach _y mov [_z], \$1 // interrupt handler	mov r1, [_z] mov r2, [_y]
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = _x Initially [_y] = [_z] = 0, [_x] to 511[_x]= 0, _x <= _y < _x+512, _z is a separate memory location r1 = 1 and r2 = 0 is allowed	

In Example 9-14, processor 0 started a string operation to write to a memory block of 512 bytes starting at address `_x`. Processor 0 got interrupted after `k` iterations of store operations. The address `_y` has not yet been updated by processor 0 when processor 0 got interrupted. The interrupt handler that took control on processor 0 writes to the address `_z`. Processor 1 may see the store to `_z` from the interrupt handler, before seeing the remaining stores to the 512-byte memory block that are executed when the string operation resumes.

Example 9-15 illustrates the ordering of string operations with earlier stores. No store from a string operation can be visible before all prior stores are visible.

Example 9-15. String Operations Are not Reordered with Earlier Stores

Processor 0	Processor 1
<code>mov [_z], \$1</code> <code>rep:stosd [_x]</code>	<code>mov r1, [_y]</code> <code>mov r2, [_z]</code>
Initially on processor 0: <code>EAX = 1, ECX=128, ES:EDI = _x</code> Initially <code>[_y] = [_z] = 0, [_x] to 511[_x]= 0, _x <= _y < _x+512, _z</code> is a separate memory location <code>r1 = 1 and r2 = 0</code> is not allowed	

9.2.5 Strengthening or Weakening the Memory-Ordering Model

The Intel 64 and IA-32 architectures provide several mechanisms for strengthening or weakening the memory-ordering model to handle special programming situations. These mechanisms include:

- The I/O instructions, locked instructions, the LOCK prefix, and serializing instructions force stronger ordering on the processor.
- The SFENCE instruction (introduced to the IA-32 architecture in the Pentium III processor) and the LFENCE and MFENCE instructions (introduced in the Pentium 4 processor) provide memory-ordering and serialization capabilities for specific types of memory operations.
- The memory type range registers (MTRRs) can be used to strengthen or weaken memory ordering for specific area of physical memory (see Section 12.11, "Memory Type Range Registers (MTRRs)"). MTRRs are available only in the Pentium 4, Intel Xeon, and P6 family processors.
- The page attribute table (PAT) can be used to strengthen memory ordering for a specific page or group of pages (see Section 12.12, "Page Attribute Table (PAT)"). The PAT is available only in the Pentium 4, Intel Xeon, and Pentium III processors.

These mechanisms can be used as follows:

Memory mapped devices and other I/O devices on the bus are often sensitive to the order of writes to their I/O buffers. I/O instructions can be used to (the IN and OUT instructions) impose strong write ordering on such accesses as follows. Prior to executing an I/O instruction, the processor waits for all previous instructions in the program to complete and for all buffered writes to drain to memory. Only instruction fetch and page tables walks can pass I/O instructions. Execution of subsequent instructions do not begin until the processor determines that the I/O instruction has been completed.

Synchronization mechanisms in multiple-processor systems may depend upon a strong memory-ordering model. Here, a program can use a locked instruction such as the XCHG instruction or the LOCK prefix to ensure that a read-modify-write operation on memory is carried out atomically. Locked instructions typically operate like I/O instructions in that they wait for all previous memory accesses to complete and for all buffered writes to drain to memory (see Section 9.1.2, "Bus Locking"). Unlike I/O operations, locked instructions do not wait for all previous instructions to complete execution.

Program synchronization can also be carried out with serializing instructions (see Section 9.3). These instructions are typically used at critical procedure or task boundaries to force completion of all previous instructions before a jump to a new section of code or a context switch occurs. Like the I/O instructions, the processor waits until all previous instructions have been completed and all buffered writes have been drained to memory before executing the serializing instruction.

The SFENCE, LFENCE, and MFENCE instructions provide a performance-efficient way of ensuring load and store memory ordering between routines that produce weakly-ordered results and routines that consume that data. The functions of these instructions are as follows:

- **SFENCE** — Serializes all store (write) operations that occurred prior to the SFENCE instruction in the program instruction stream, but does not affect load operations.
- **LFENCE** — Serializes all load (read) operations that occurred prior to the LFENCE instruction in the program instruction stream, but does not affect store operations.¹
- **MFENCE** — Serializes all store and load operations that occurred prior to the MFENCE instruction in the program instruction stream.

Note that the SFENCE, LFENCE, and MFENCE instructions provide a more efficient method of controlling memory ordering than the CPUID instruction.

The MTRRs were introduced in the P6 family processors to define the cache characteristics for specified areas of physical memory. The following are two examples of how memory types set up with MTRRs can be used strengthen or weaken memory ordering for the Pentium 4, Intel Xeon, and P6 family processors:

- The strong uncached (UC) memory type forces a strong-ordering model on memory accesses. Here, all reads and writes to the UC memory region appear on the bus and out-of-order or speculative accesses are not performed. This memory type can be applied to an address range dedicated to memory mapped I/O devices to force strong memory ordering.
- For areas of memory where weak ordering is acceptable, the write back (WB) memory type can be chosen. Here, reads can be performed speculatively and writes can be buffered and combined. For this type of memory, cache locking is performed on atomic (locked) operations that do not split across cache lines, which helps to reduce the performance penalty associated with the use of the typical synchronization instructions, such as XCHG, that lock the bus during the entire read-modify-write operation. With the WB memory type, the XCHG instruction locks the cache instead of the bus if the memory access is contained within a cache line.

The PAT was introduced in the Pentium III processor to enhance the caching characteristics that can be assigned to pages or groups of pages. The PAT mechanism typically used to strengthen caching characteristics at the page level with respect to the caching characteristics established by the MTRRs. Table 12-7 shows the interaction of the PAT with the MTRRs.

Intel recommends that software written to run on Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, Intel Xeon, and P6 family processors assume the processor-ordering model or a weaker memory-ordering model. The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, Intel Xeon, and P6 family processors do not implement a strong memory-ordering model, except when using the UC memory type. Despite the fact that Pentium 4, Intel Xeon, and P6 family processors support processor ordering, Intel does not guarantee that future processors will support this model. To make software portable to future processors, it is recommended that operating systems provide critical region and resource control constructs and API's (application program interfaces) based on I/O, locking, and/or serializing instructions be used to synchronize access to shared areas of memory in multiple-processor systems. Also, software should not depend on processor ordering in situations where the system hardware does not support this memory-ordering model.

9.3 SERIALIZING INSTRUCTIONS

The Intel 64 and IA-32 architectures define several **serializing instructions**. These instructions force the processor to complete all modifications to flags, registers, and memory by previous instructions and to drain all buffered writes to memory before the next instruction is fetched and executed. For example, when a MOV to control register instruction is used to load a new value into control register CR0 to enable protected mode, the processor must perform a serializing operation before it enters protected mode. This serializing operation ensures

1. Specifically, LFENCE does not execute until all prior instructions have completed locally, and no later instruction begins execution until LFENCE completes. As a result, an instruction that loads from memory and that precedes an LFENCE receives data from memory prior to completion of the LFENCE. An LFENCE that follows an instruction that stores to memory might complete before the data being stored have become globally visible. Instructions following an LFENCE may be fetched from memory before the LFENCE, but they will not execute until the LFENCE completes.

that all operations that were started while the processor was in real-address mode are completed before the switch to protected mode is made.

The concept of serializing instructions was introduced into the IA-32 architecture with the Pentium processor to support parallel instruction execution. Serializing instructions have no meaning for the Intel486 and earlier processors that do not implement parallel instruction execution.

It is important to note that executing of serializing instructions on P6 and more recent processor families constrain speculative execution because the results of speculatively executed instructions are discarded. The following instructions are serializing instructions:

- **Privileged serializing instructions** — INVD, INVEPT, INVLPG, INVVPID, LGDT, LIDT, LLDT, LTR, MOV (to control register, with the exception of MOV CR8¹), MOV (to debug register), WBINVD, and WRMSR².
- **Non-privileged serializing instructions** — CPUID, IRET, RSM, and SERIALIZE.

When the processor serializes instruction execution, it ensures that all pending memory transactions are completed (including writes stored in its store buffer) before it executes the next instruction. Nothing can pass a serializing instruction and a serializing instruction cannot pass any other instruction (read, write, instruction fetch, or I/O). For example, CPUID can be executed at any privilege level to serialize instruction execution with no effect on program flow, except that the EAX, EBX, ECX, and EDX registers are modified.

The following instructions are memory-ordering instructions, not serializing instructions. These drain the data memory subsystem. They do not serialize the instruction execution stream:³

- **Non-privileged memory-ordering instructions** — SFENCE, LFENCE, and MFENCE.

The SFENCE, LFENCE, and MFENCE instructions provide more granularity in controlling the serialization of memory loads and stores (see Section 9.2.5, “Strengthening or Weakening the Memory-Ordering Model”).

The following additional information is worth noting regarding serializing instructions:

- The processor does not write back the contents of modified data in its data cache to external memory when it serializes instruction execution. Software can force modified data to be written back by executing the WBINVD instruction, which is a serializing instruction. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.
- When an instruction is executed that enables or disables paging (that is, changes the PG flag in control register CR0), the instruction should be followed by a jump instruction. The target instruction of the jump instruction is fetched with the new setting of the PG flag (that is, paging is enabled or disabled), but the jump instruction itself is fetched with the previous setting. The Pentium 4, Intel Xeon, and P6 family processors do not require the jump operation following the move to register CR0 (because any use of the MOV instruction in a Pentium 4, Intel Xeon, or P6 family processor to write to CR0 is completely serializing). However, to maintain backwards and forward compatibility with code written to run on other IA-32 processors, it is recommended that the jump operation be performed.
- Whenever an instruction is executed to change the contents of CR3 while paging is enabled, the next instruction is fetched using the translation tables that correspond to the new value of CR3. Therefore the next instruction and the sequentially following instructions should have a mapping based upon the new value of CR3. (Global entries in the TLBs are not invalidated, see Section 4.10.4, “Invalidation of TLBs and Paging-Structure Caches.”)
- The Pentium processor and more recent processor families use branch-prediction techniques to improve performance by prefetching the destination of a branch instruction before the branch instruction is executed. Consequently, instruction execution is not deterministically serialized when a branch instruction is executed.

1. MOV CR8 is not defined architecturally as a serializing instruction.

2. An execution of WRMSR to any non-serializing MSR is not serializing. Non-serializing MSRs include the following: IA32_SPEC_CTRL MSR (MSR index 48H), IA32_PRED_CMD MSR (MSR index 49H), IA32_TSX_CTRL MSR (MSR index 122H), IA32_TSC_DEADLINE MSR (MSR index 6E0H), IA32_PKRS MSR (MSR index 6E1H), IA32_HWP_REQUEST MSR (MSR index 774H), or any of the x2APIC MSRs (MSR indices 802H to 83FH).

3. LFENCE does provide some guarantees on instruction ordering. It does not execute until all prior instructions have completed locally, and no later instruction begins execution until LFENCE completes.

9.4 MULTIPLE-PROCESSOR (MP) INITIALIZATION

The IA-32 architecture (beginning with the P6 family processors) defines a multiple-processor (MP) initialization protocol called the *Multiprocessor Specification Version 1.4*. This specification defines the boot protocol to be used by IA-32 processors in multiple-processor systems. (Here, **multiple processors** is defined as two or more processors.) The MP initialization protocol has the following important features:

- It supports controlled booting of multiple processors without requiring dedicated system hardware.
- It allows hardware to initiate the booting of a system without the need for a dedicated signal or a predefined boot processor.
- It allows all IA-32 processors to be booted in the same manner, including those supporting Intel Hyper-Threading Technology.
- The MP initialization protocol also applies to MP systems using Intel 64 processors.

The mechanism for carrying out the MP initialization protocol differs depending on the Intel processor generations. The following bullets summarize the evolution of the changes:

- **For P6 family or older processors supporting MP operations**— The selection of the BSP and APs (see Section 9.4.1, “BSP and AP Processors”) is handled through arbitration on the APIC bus, using BIPI and FIPI messages. These processor generations have CPUID signatures of (family=06H, extended_model=0, model<=0DH), or family <06H. See Section 9.11.1, “Overview of the MP Initialization Process for P6 Family Processors” for a complete discussion of MP initialization for P6 family processors.
- **Early generations of IA processors with family 0FH** — The selection of the BSP and APs (see Section 9.4.1, “BSP and AP Processors”) is handled through arbitration on the system bus, using BIPI and FIPI messages (see Section 9.4.3, “MP Initialization Protocol Algorithm for MP Systems”). These processor generations have CPUID signatures of family=0FH, model=0H, stepping<=09H.
- **Later generations of IA processors with family 0FH, and IA processors with system bus** — The selection of the BSP and APs is handled through a special system bus cycle, without using BIPI and FIPI message arbitration (see Section 9.4.3, “MP Initialization Protocol Algorithm for MP Systems”). These processor generations have CPUID signatures of family=0FH with (model=0H, stepping>=0AH) or (model >0, all steppings); or family=06H, extended_model=0, model>=0EH.
- **All other modern IA processor generations supporting MP operations**— The selection of the BSP and APs in the system is handled by platform-specific arrangement of the combination of hardware, BIOS, and/or configuration input options. The basis of the selection mechanism is similar to those of the Later generations of family 0FH and other Intel processor using system bus (see Section 9.4.3, “MP Initialization Protocol Algorithm for MP Systems”). These processor generations have CPUID signatures of family=06H, extended_model>0.

The family, model, and stepping ID for a processor is given in the EAX register when the CPUID instruction is executed with a value of 1 in the EAX register.

9.4.1 BSP and AP Processors

The MP initialization protocol defines two classes of processors: the bootstrap processor (BSP) and the application processors (APs). Following a power-up or RESET of an MP system, system hardware dynamically selects one of the processors on the system bus as the BSP. The remaining processors are designated as APs.

As part of the BSP selection mechanism, the BSP flag is set in the IA32_APIC_BASE MSR (see Figure 11-5) of the BSP, indicating that it is the BSP. This flag is cleared for all other processors.

The BSP executes the BIOS’s boot-strap code to configure the APIC environment, sets up system-wide data structures, and starts and initializes the APs. When the BSP and APs are initialized, the BSP then begins executing the operating-system initialization code.

Following a power-up or reset, the APs complete a minimal self-configuration, then wait for a startup signal (a SIPI message) from the BSP processor. Upon receiving a SIPI message, an AP executes the BIOS AP configuration code, which ends with the AP being placed in halt state.

For Intel 64 and IA-32 processors supporting Intel Hyper-Threading Technology, the MP initialization protocol treats each of the logical processors on the system bus or coherent link domain as a separate processor (with a unique

APIC ID). During boot-up, one of the logical processors is selected as the BSP and the remainder of the logical processors are designated as APs.

9.4.2 MP Initialization Protocol Requirements and Restrictions

The MP initialization protocol imposes the following requirements and restrictions on the system:

- The MP protocol is executed only after a power-up or RESET. If the MP protocol has completed and a BSP is chosen, subsequent INITs (either to a specific processor or system wide) do not cause the MP protocol to be repeated. Instead, each logical processor examines its BSP flag (in the IA32_APIC_BASE MSR) to determine whether it should execute the BIOS boot-strap code (if it is the BSP) or enter a wait-for-SIPI state (if it is an AP).
- All devices in the system that are capable of delivering interrupts to the processors must be inhibited from doing so for the duration of the MP initialization protocol. The time during which interrupts must be inhibited includes the window between when the BSP issues an INIT-SIPI-SIPI sequence to an AP and when the AP responds to the last SIPI in the sequence.

9.4.3 MP Initialization Protocol Algorithm for MP Systems

Following a power-up or RESET of an MP system, the processors in the system execute the MP initialization protocol algorithm to initialize each of the logical processors on the system bus or coherent link domain. In the course of executing this algorithm, the following boot-up and initialization operations are carried out:

1. Each logical processor is assigned a unique APIC ID, based on system topology. The unique ID is a 32-bit value if the processor supports CPUID leaf 0BH, otherwise the unique ID is an 8-bit value. (see Section 9.4.5, "Identifying Logical Processors in an MP System").
2. Each logical processor is assigned a unique arbitration priority based on its APIC ID.
3. Each logical processor executes its internal BIST simultaneously with the other logical processors in the system.
4. Upon completion of the BIST, the logical processors use a hardware-defined selection mechanism to select the BSP and the APs from the available logical processors on the system bus. The BSP selection mechanism differs depending on the family, model, and stepping IDs of the processors, as follows:
 - Later generations of IA processors within family 0FH (see Section 9.4), IA processors with system bus (family=06H, extended_model=0, model>=0EH), or all other modern Intel processors (family=06H, extended_model>0):
 - The logical processors begin monitoring the BNR# signal, which is toggling. When the BNR# pin stops toggling, each processor attempts to issue a NOP special cycle on the system bus.
 - The logical processor with the highest arbitration priority succeeds in issuing a NOP special cycle and is nominated the BSP. This processor sets the BSP flag in its IA32_APIC_BASE MSR, then fetches and begins executing BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).
 - The remaining logical processors (that failed in issuing a NOP special cycle) are designated as APs. They leave their BSP flags in the clear state and enter a "wait-for-SIPI state."
 - Early generations of IA processors within family 0FH (family=0FH, model=0H, stepping<=09H), P6 family or older processors supporting MP operations (family=06H, extended_model=0, model<=0DH; or family<06H):
 - Each processor broadcasts a BIPI to "all including self." The first processor that broadcasts a BIPI (and thus receives its own BIPI vector), selects itself as the BSP and sets the BSP flag in its IA32_APIC_BASE MSR. (See Section 9.11.1, "Overview of the MP Initialization Process for P6 Family Processors" for a description of the BIPI, FIPI, and SIPI messages.)
 - The remainder of the processors (which were not selected as the BSP) are designated as APs. They leave their BSP flags in the clear state and enter a "wait-for-SIPI state."
 - The newly established BSP broadcasts an FIPI message to "all including self," which the BSP and APs treat as an end of MP initialization signal. Only the processor with its BSP flag set responds to the FIPI

message. It responds by fetching and executing the BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).

5. As part of the boot-strap code, the BSP creates an ACPI table and/or an MP table and adds its initial APIC ID to these tables as appropriate.
6. At the end of the boot-strap procedure, the BSP sets a processor counter to 1, then broadcasts a SIPI message to all the APs in the system. Here, the SIPI message contains a vector to the BIOS AP initialization code (at 000VV000H, where VV is the vector contained in the SIPI message).
7. The first action of the AP initialization code is to set up a race (among the APs) to a BIOS initialization semaphore. The first AP to the semaphore begins executing the initialization code. (See Section 9.4.4, “MP Initialization Example,” for semaphore implementation details.) As part of the AP initialization procedure, the AP adds its APIC ID number to the ACPI and/or MP tables as appropriate and increments the processor counter by 1. At the completion of the initialization procedure, the AP executes a CLI instruction and halts itself.
8. When each of the APs has gained access to the semaphore and executed the AP initialization code, the BSP establishes a count for the number of processors connected to the system bus, completes executing the BIOS boot-strap code, and then begins executing operating-system boot-strap and start-up code.
9. While the BSP is executing operating-system boot-strap and start-up code, the APs remain in the halted state. In this state they will respond only to INITs, NMIs, and SMIs. They will also respond to snoops and to assertions of the STPCLK# pin.

The following section gives an example (with code) of the MP initialization protocol for of multiple processors operating in an MP configuration.

Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4 describes how to program the LINT[0:1] pins of the processor’s local APICs after an MP configuration has been completed.

9.4.4 MP Initialization Example

The following example illustrates the use of the MP initialization protocol used to initialize processors in an MP system after the BSP and APs have been established. The code runs on Intel 64 or IA-32 processors that use a protocol. This includes P6 Family processors, Pentium 4 processors, Intel Core Duo, Intel Core 2 Duo and Intel Xeon processors.

The following constants and data definitions are used in the accompanying code examples. They are based on the addresses of the APIC registers defined in Table 11-1.

```

ICR_LOW      EQU OFEE00300H
SVR          EQU OFEE000F0H
APIC_ID      EQU OFEE00020H
LVT3        EQU OFEE00370H
APIC_ENABLED EQU 0100H
BOOT_ID      DD ?
COUNT      EQU 00H
VACANT       EQU 00H

```

9.4.4.1 Typical BSP Initialization Sequence

After the BSP and APs have been selected (by means of a hardware protocol, see Section 9.4.3, “MP Initialization Protocol Algorithm for MP Systems”), the BSP begins executing BIOS boot-strap code (POST) at the normal IA-32 architecture starting address (FFFF FFF0H). The boot-strap code typically performs the following operations:

1. Initializes memory.
2. Loads the microcode update into the processor.
3. Initializes the MTRRs.
4. Enables the caches.

5. Executes the CPUID instruction with a value of 0H in the EAX register, then reads the EBX, ECX, and EDX registers to determine if the BSP is "GenuineIntel."
6. Executes the CPUID instruction with a value of 1H in the EAX register, then saves the values in the EAX, ECX, and EDX registers in a system configuration space in RAM for use later.
7. Loads start-up code for the AP to execute into a 4-KByte page in the lower 1 MByte of memory.
8. Switches to protected mode and ensures that the APIC address space is mapped to the strong uncacheable (UC) memory type.
9. Determine the BSP's APIC ID from the local APIC ID register (default is 0), the code snippet below is an example that applies to logical processors in a system whose local APIC units operate in xAPIC mode that APIC registers are accessed using memory mapped interface:

```

MOV ESI, APIC_ID; Address of local APIC ID register
MOV EAX, [ESI];
AND EAX, 0FF00000H; Zero out all other bits except APIC ID
MOV BOOT_ID, EAX; Save in memory
    
```

Saves the APIC ID in the ACPI and/or MP tables and optionally in the system configuration space in RAM.

10. Converts the base address of the 4-KByte page for the AP's bootup code into 8-bit vector. The 8-bit vector defines the address of a 4-KByte page in the real-address mode address space (1-MByte space). For example, a vector of 0BDH specifies a start-up memory address of 000BD000H.
11. Enables the local APIC by setting bit 8 of the APIC spurious vector register (SVR).

```

MOV ESI, SVR; Address of SVR
MOV EAX, [ESI];
OR EAX, APIC_ENABLED; Set bit 8 to enable (0 on reset)
MOV [ESI], EAX;
    
```

12. Sets up the LVT error handling entry by establishing an 8-bit vector for the APIC error handler.

```

MOV ESI, LVT3;
MOV EAX, [ESI];
AND EAX, 0FFFFFF0H; Clear out previous vector.
OR EAX, 000000xxH; xx is the 8-bit vector the APIC error handler.
MOV [ESI], EAX;
    
```

13. Initializes the Lock Semaphore variable VACANT to 00H. The APs use this semaphore to determine the order in which they execute BIOS AP initialization code.
14. Performs the following operation to set up the BSP to detect the presence of APs in the system and the number of processors (within a finite duration, minimally 100 milliseconds):
 - Sets the value of the COUNT variable to 1.
 - In the AP BIOS initialization code, the AP will increment the COUNT variable to indicate its presence. The finite duration while waiting for the COUNT to be updated can be accomplished with a timer. When the timer expires, the BSP checks the value of the COUNT variable. If the timer expires and the COUNT variable has not been incremented, no APs are present or some error has occurred.

15. Broadcasts an INIT-SIPI-SIPI IPI sequence to the APs to wake them up and initialize them. Alternatively, following a power-up or RESET, since all APs are already in the "wait-for-SIPI state," the BSP can broadcast just a single SIPI IPI to the APs to wake them up and initialize them. If software knows how many logical processors it expects to wake up, it may choose to poll the COUNT variable. If the expected processors show up before the 100 millisecond timer expires, the timer can be canceled and skip to step 16.

The left-hand-side of the procedure illustrated in Table 9-1 provides an algorithm when the expected processor count is unknown. The right-hand-side of Table 9-1 can be used when the expected processor count is known.

Table 9-1. Broadcast INIT-SIPI-SIPI Sequence and Choice of Timeouts

INIT-SIPI-SIPI when the expected processor count is unknown	INIT-SIPI-SIPI when the expected processor count is known
MOV ESI, ICR_LOW; Load address of ICR low dword into ESI. MOV EAX, 000C4500H; Load ICR encoding for broadcast INIT IPI ; to all APs into EAX. MOV [ESI], EAX; Broadcast INIT IPI to all APs ; 10-millisecond delay loop. MOV EAX, 000C46XXH; Load ICR encoding for broadcast SIPI IP ; to all APs into EAX, where xx is the vector computed in step 10. MOV [ESI], EAX; Broadcast SIPI IPI to all APs ; 200-microsecond delay loop MOV [ESI], EAX; Broadcast second SIPI IPI to all APs ; Waits for the timer interrupt until the timer expires	MOV ESI, ICR_LOW; Load address of ICR low dword into ESI. MOV EAX, 000C4500H; Load ICR encoding for broadcast INIT IPI ; to all APs into EAX. MOV [ESI], EAX; Broadcast INIT IPI to all APs ; 10-millisecond delay loop. MOV EAX, 000C46XXH; Load ICR encoding for broadcast SIPI IP ; to all APs into EAX, where xx is the vector computed in step 10. MOV [ESI], EAX; Broadcast SIPI IPI to all APs ; 200 microsecond delay loop with check to see if COUNT has ; reached the expected processor count. If COUNT reaches ; expected processor count, cancel timer and go to step 16. MOV [ESI], EAX; Broadcast second SIPI IPI to all APs ; Wait for the timer interrupt polling COUNT. If COUNT reaches ; expected processor count, cancel timer and go to step 16. ; If timer expires, go to step 16.

16. Reads and evaluates the COUNT variable and establishes a processor count.

17. If necessary, reconfigures the APIC and continues with the remaining system diagnostics as appropriate.

9.4.4.2 Typical AP Initialization Sequence

When an AP receives the SIPI, it begins executing BIOS AP initialization code at the vector encoded in the SIPI. The AP initialization code typically performs the following operations:

1. Waits on the BIOS initialization Lock Semaphore. When control of the semaphore is attained, initialization continues.
2. Loads the microcode update into the processor.
3. Initializes the MTRRs (using the same mapping that was used for the BSP).
4. Enables the cache.
5. Executes the CPUID instruction with a value of 0H in the EAX register, then reads the EBX, ECX, and EDX registers to determine if the AP is "GenuineIntel."
6. Executes the CPUID instruction with a value of 1H in the EAX register, then saves the values in the EAX, ECX, and EDX registers in a system configuration space in RAM for use later.
7. Switches to protected mode and ensures that the APIC address space is mapped to the strong uncacheable (UC) memory type.
8. Determines the AP's APIC ID from the local APIC ID register, and adds it to the MP and ACPI tables and optionally to the system configuration space in RAM.
9. Initializes and configures the local APIC by setting bit 8 in the SVR register and setting up the LVT3 (error LVT) for error handling (as described in steps 9 and 10 in Section 9.4.4.1, "Typical BSP Initialization Sequence").
10. Configures the APs SMI execution environment. (Each AP and the BSP must have a different SMBASE address.)
11. Increments the COUNT variable by 1.
12. Releases the semaphore.
13. Executes one of the following:

- the CLI and HLT instructions (if MONITOR/MWAIT is not supported), or
- the CLI, MONITOR, and MWAIT sequence to enter a deep C-state.

14. Waits for an INIT IPI.

9.4.5 Identifying Logical Processors in an MP System

After the BIOS has completed the MP initialization protocol, each logical processor can be uniquely identified by its local APIC ID. Software can access these APIC IDs in either of the following ways:

- **Read APIC ID for a local APIC** — Code running on a logical processor can read APIC ID in one of two ways depending on the local APIC unit is operating in x2APIC mode (see Intel® 64 Architecture x2APIC Specification) or in xAPIC mode:
 - If the local APIC unit supports x2APIC and is operating in x2APIC mode, 32-bit APIC ID can be read by executing a RDMSR instruction to read the processor's x2APIC ID register. This method is equivalent to executing CPUID leaf 0BH described below.
 - If the local APIC unit is operating in xAPIC mode, 8-bit APIC ID can be read by executing a MOV instruction to read the processor's local APIC ID register (see Section 11.4.6, "Local APIC ID"). This is the ID to use for directing physical destination mode interrupts to the processor.
- **Read ACPI or MP table** — As part of the MP initialization protocol, the BIOS creates an ACPI table and an MP table. These tables are defined in the Multiprocessor Specification Version 1.4 and provide software with a list of the processors in the system and their local APIC IDs. The format of the ACPI table is derived from the ACPI specification, which is an industry standard power management and platform configuration specification for MP systems.
- **Read Initial APIC ID** (If the processor does not support CPUID leaf 0BH) — An APIC ID is assigned to a logical processor during power up. This is the initial APIC ID reported by CPUID.1:EBX[31:24] and may be different from the current value read from the local APIC. The initial APIC ID can be used to determine the topological relationship between logical processors for multi-processor systems that do not support CPUID leaf 0BH.

Bits in the 8-bit initial APIC ID can be interpreted using several bit masks. Each bit mask can be used to extract an identifier to represent a hierarchical **domain** of the multi-threading resource topology in an MP system (See Section 9.9.1, "Hierarchical Mapping of Shared Resources"). The initial APIC ID may consist of up to four bit-fields. In a non-clustered MP system, the field consists of up to three bit fields.
- **Read 32-bit APIC ID from CPUID leaf 0BH** (If the processor supports CPUID leaf 0BH) — A unique APIC ID is assigned to a logical processor during power up. This APIC ID is reported by CPUID.0BH:EDX[31:0] as a 32-bit value. Use the 32-bit APIC ID and CPUID leaf 0BH to determine the topological relationship between logical processors if the processor supports CPUID leaf 0BH.

Bits in the 32-bit x2APIC ID can be extracted into sub-fields using CPUID leaf 0BH parameters. (See Section 9.9.1, "Hierarchical Mapping of Shared Resources").

Figure 9-2 shows two examples of APIC ID bit fields in earlier single-core processors. In single-core Intel Xeon processors, the APIC ID assigned to a logical processor during power-up and initialization is 8 bits. Bits 2:1 form a 2-bit physical package identifier (which can also be thought of as a socket identifier). In systems that configure physical processors in clusters, bits 4:3 form a 2-bit cluster ID. Bit 0 is used in the Intel Xeon processor MP to identify the two logical processors within the package (see Section 9.9.3, "Hierarchical ID of Logical Processors in an MP System"). For Intel Xeon processors that do not support Intel Hyper-Threading Technology, bit 0 is always set to 0; for Intel Xeon processors supporting Intel Hyper-Threading Technology, bit 0 performs the same function as it does for Intel Xeon processor MP.

For more recent multi-core processors, see Section 9.9.1, "Hierarchical Mapping of Shared Resources" for a complete description of the topological relationships between logical processors and bit field locations within an initial APIC ID across Intel 64 and IA-32 processor families.

Note the number of bit fields and the width of bit-fields are dependent on processor and platform hardware capabilities. Software should determine these at runtime. When initial APIC IDs are assigned to logical processors, the value of APIC ID assigned to a logical processor will respect the bit-field boundaries corresponding core, physical package, etc. Additional examples of the bit fields in the initial APIC ID of multi-threading capable systems are shown in Section 9.9.

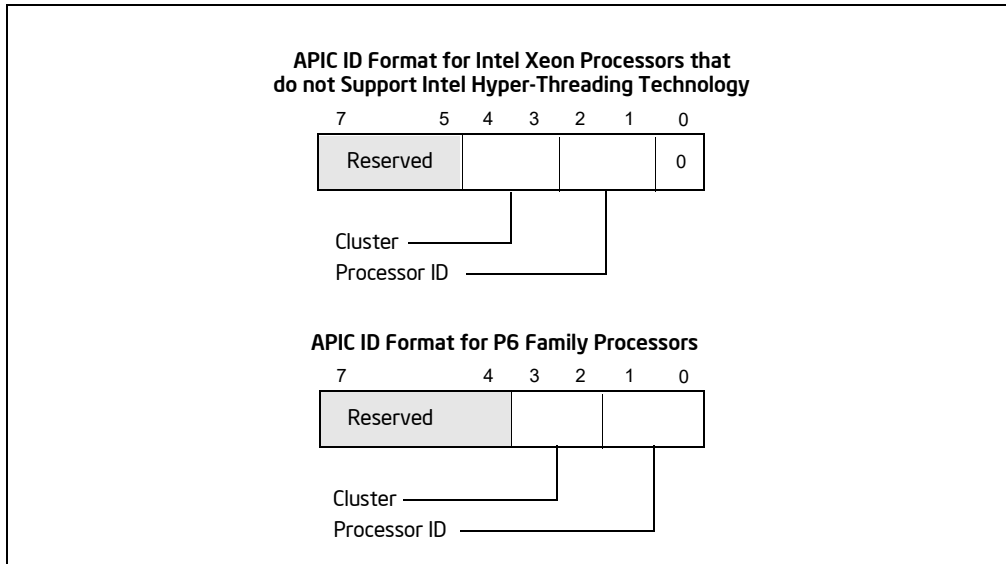


Figure 9-2. Interpretation of APIC ID in Early MP Systems

For P6 family processors, the APIC ID that is assigned to a processor during power-up and initialization is 4 bits (see Figure 9-2). Here, bits 0 and 1 form a 2-bit processor (or socket) identifier and bits 2 and 3 form a 2-bit cluster ID.

9.5 INTEL® HYPER-THREADING TECHNOLOGY AND INTEL® MULTI-CORE TECHNOLOGY

Intel Hyper-Threading Technology and Intel multi-core technology are extensions to Intel 64 and IA-32 architectures that enable a single physical processor to execute two or more separate code streams (called *threads*) concurrently. In Intel Hyper-Threading Technology, a single processor core provides two logical processors that share execution resources (see Section 9.7, “Intel® Hyper-Threading Technology Architecture”). In Intel multi-core technology, a physical processor package provides two or more processor cores. Both configurations require chipsets and a BIOS that support the technologies.

Software should not rely on processor names to determine whether a processor supports Intel Hyper-Threading Technology or Intel multi-core technology. Use the CPUID instruction to determine processor capability (see Section 9.6.2, “Initializing Multi-Core Processors”).

9.6 DETECTING HARDWARE MULTI-THREADING SUPPORT AND TOPOLOGY

Use the CPUID instruction to detect the presence of hardware multi-threading support in a physical processor. Hardware multi-threading can support several varieties of multigrade and/or Intel Hyper-Threading Technology. CPUID instruction provides several sets of parameter information to aid software enumerating topology information. The relevant topology enumeration parameters provided by CPUID include:

- **Hardware Multi-Threading feature flag (CPUID.1:EDX[28] = 1)** — Indicates when set that the physical package is capable of supporting Intel Hyper-Threading Technology and/or multiple cores.
- **Processor topology enumeration parameters for 8-bit APIC ID:**
 - **Addressable IDs for Logical processors in the same Package (CPUID.1:EBX[23:16])** — Indicates the maximum number of addressable ID for logical processors in a physical package. Within a physical package, there may be addressable IDs that are not occupied by any logical processors. This parameter does not represent the hardware capability of the physical processor.¹

- **Addressable IDs for processor cores in the same Package¹ (CPUID.(EAX=4, ECX=0²):EAX[31:26] + 1 = Y)** — Indicates the maximum number of addressable IDs attributable to processor cores (Y) in the physical package.
- **Extended Processor Topology Enumeration parameters for 32-bit APIC ID:** Intel 64 processors supporting CPUID leaf 0BH will assign unique APIC IDs to each logical processor in the system. CPUID leaf 0BH reports the 32-bit APIC ID and provide topology enumeration parameters. See CPUID instruction reference pages in Intel[®] 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

The CPUID feature flag may indicate support for hardware multi-threading when only one logical processor available in the package. In this case, the decimal value represented by bits 16 through 23 in the EBX register will have a value of 1.

Software should note that the number of logical processors enabled by system software may be less than the value of "Addressable IDs for Logical processors". Similarly, the number of cores enabled by system software may be less than the value of "Addressable IDs for processor cores".

Software can detect the availability of the CPUID extended topology enumeration leaf (0BH) by performing two steps:

- Check maximum input value for basic CPUID information by executing CPUID with EAX= 0. If CPUID.0H:EAX is greater than or equal to 11 (0BH), then proceed to next step,
- Check CPUID.EAX=0BH, ECX=0H:EBX is non-zero.

If both of the above conditions are true, extended topology enumeration leaf is available. Note the presence of CPUID leaf 0BH in a processor does not guarantee support that the local APIC supports x2APIC. If CPUID.(EAX=0BH, ECX=0H):EBX returns zero and maximum input value for basic CPUID information is greater than 0BH, then CPUID.0BH leaf is not supported on that processor.

9.6.1 Initializing Processors Supporting Intel[®] Hyper-Threading Technology

The initialization process for an MP system that contains processors supporting Intel Hyper-Threading Technology is the same as for conventional MP systems (see Section 9.4, "Multiple-Processor (MP) Initialization"). One logical processor in the system is selected as the BSP and other processors (or logical processors) are designated as APs. The initialization process is identical to that described in Section 9.4.3, "MP Initialization Protocol Algorithm for MP Systems," and Section 9.4.4, "MP Initialization Example."

During initialization, each logical processor is assigned an APIC ID that is stored in the local APIC ID register for each logical processor. If two or more processors supporting Intel Hyper-Threading Technology are present, each logical processor on the system bus is assigned a unique ID (see Section 9.9.3, "Hierarchical ID of Logical Processors in an MP System"). Once logical processors have APIC IDs, software communicates with them by sending APIC IPI messages.

9.6.2 Initializing Multi-Core Processors

The initialization process for an MP system that contains multi-core Intel 64 or IA-32 processors is the same as for conventional MP systems (see Section 9.4, "Multiple-Processor (MP) Initialization"). A logical processor in one core is selected as the BSP; other logical processors are designated as APs.

During initialization, each logical processor is assigned an APIC ID. Once logical processors have APIC IDs, software may communicate with them by sending APIC IPI messages.

-
1. Operating system and BIOS may implement features that reduce the number of logical processors available in a platform to applications at runtime to less than the number of physical packages times the number of hardware-capable logical processors per package.
 1. Software must check CPUID for its support of leaf 4 when implementing support for multi-core. If CPUID leaf 4 is not available at runtime, software should handle the situation as if there is only one core per package.
 2. Maximum number of cores in the physical package must be queried by executing CPUID with EAX=4 and a valid ECX input value. Valid ECX input values start from 0.

9.6.3 Executing Multiple Threads on an Intel® 64 or IA-32 Processor Supporting Hardware Multi-Threading

Upon completing the operating system boot-up procedure, the bootstrap processor (BSP) executes operating system code. Other logical processors are placed in the halt state. To execute a code stream (thread) on a halted logical processor, the operating system issues an interprocessor interrupt (IPI) addressed to the halted logical processor. In response to the IPI, the processor wakes up and begins executing the code identified by the vector received as part of the IPI.

To manage execution of multiple threads on logical processors, an operating system can use conventional symmetric multiprocessing (SMP) techniques. For example, the operating-system can use a time-slice or load balancing mechanism to periodically interrupt each of the active logical processors. Upon interrupting a logical processor, the operating system checks its run queue for a thread waiting to be executed and dispatches the thread to the interrupted logical processor.

9.6.4 Handling Interrupts on an IA-32 Processor Supporting Hardware Multi-Threading

Interrupts are handled on processors supporting Intel Hyper-Threading Technology as they are on conventional MP systems. External interrupts are received by the I/O APIC, which distributes them as interrupt messages to specific logical processors (see Figure 9-3).

Logical processors can also send IPIs to other logical processors by writing to the ICR register of its local APIC (see Section 11.6, "Issuing Interprocessor Interrupts"). This also applies to dual-core processors.

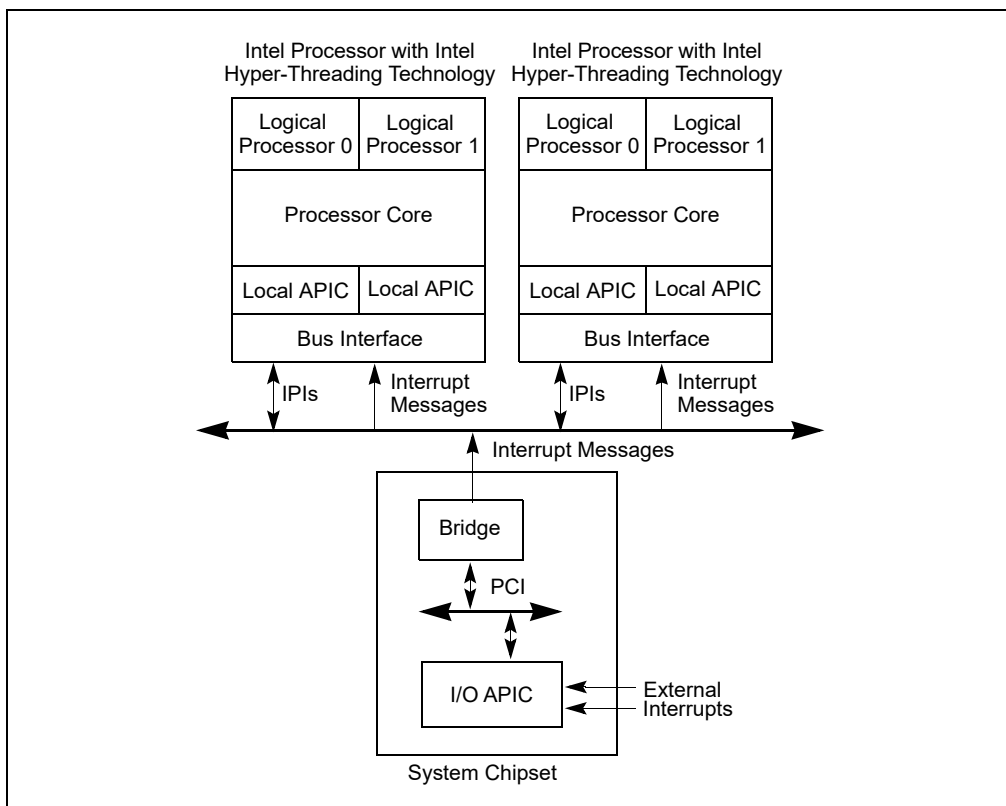


Figure 9-3. Local APICs and I/O APIC in MP System Supporting Intel HT Technology

9.7 INTEL® HYPER-THREADING TECHNOLOGY ARCHITECTURE

Figure 9-4 shows a generalized view of an Intel processor supporting Intel Hyper-Threading Technology, using the original Intel Xeon processor MP as an example. This implementation of the Intel Hyper-Threading Technology

consists of two logical processors (each represented by a separate architectural state) which share the processor’s execution engine and the bus interface. Each logical processor also has its own advanced programmable interrupt controller (APIC).

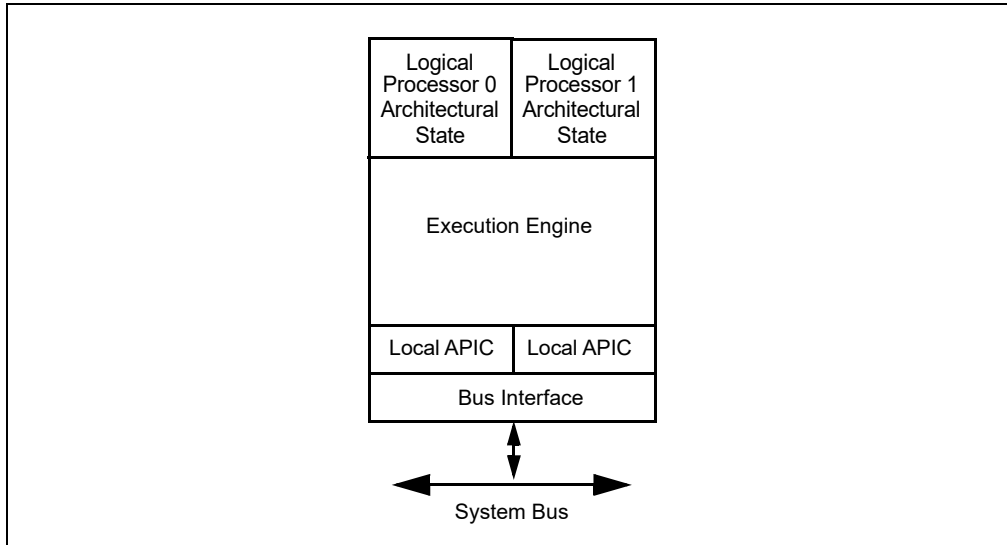


Figure 9-4. IA-32 Processor with Two Logical Processors Supporting Intel HT Technology

9.7.1 State of the Logical Processors

The following features are part of the architectural state of logical processors within Intel 64 or IA-32 processors supporting Intel Hyper-Threading Technology. The features can be subdivided into three groups:

- Duplicated for each logical processor
- Shared by logical processors in a physical processor
- Shared or duplicated, depending on the implementation

The following features are duplicated for each logical processor:

- General purpose registers (EAX, EBX, ECX, EDX, ESI, EDI, ESP, and EBP)
- Segment registers (CS, DS, SS, ES, FS, and GS)
- EFLAGS and EIP registers. Note that the CS and EIP/RIP registers for each logical processor point to the instruction stream for the thread being executed by the logical processor.
- x87 FPU registers (ST0 through ST7, status word, control word, tag word, data operand pointer, and instruction pointer)
- MMX registers (MM0 through MM7)
- XMM registers (XMM0 through XMM7) and the MXCSR register
- Control registers and system table pointer registers (GDTR, LDTR, IDTR, task register)
- Debug registers (DR0, DR1, DR2, DR3, DR6, DR7) and the debug control MSRs
- Machine check global status (IA32_MCG_STATUS) and machine check capability (IA32_MCG_CAP) MSRs
- Thermal clock modulation and ACPI Power management control MSRs
- Time stamp counter MSRs
- Most of the other MSR registers, including the page attribute table (PAT). See the exceptions below.
- Local APIC registers.
- Additional general purpose registers (R8-R15), XMM registers (XMM8-XMM15), control register, IA32_EFER on Intel 64 processors.

The following features are shared by logical processors:

- Memory type range registers (MTRRs)

Whether the following features are shared or duplicated is implementation-specific:

- IA32_MISC_ENABLE MSR (MSR address 1A0H)
- Machine check architecture (MCA) MSRs (except for the IA32_MCG_STATUS and IA32_MCG_CAP MSRs)
- Performance monitoring control and counter MSRs

9.7.2 APIC Functionality

When a processor supporting Intel Hyper-Threading Technology support is initialized, each logical processor is assigned a local APIC ID (see Table 11-1). The local APIC ID serves as an ID for the logical processor and is stored in the logical processor's APIC ID register. If two or more processors supporting Intel Hyper-Threading Technology are present in a dual processor (DP) or MP system, each logical processor on the system bus is assigned a unique local APIC ID (see Section 9.9.3, "Hierarchical ID of Logical Processors in an MP System").

Software communicates with local processors using the APIC's interprocessor interrupt (IPI) messaging facility. Setup and programming for APICs is identical in processors that support and do not support Intel Hyper-Threading Technology. See Chapter 11, "Advanced Programmable Interrupt Controller (APIC)," for a detailed discussion.

9.7.3 Memory Type Range Registers (MTRR)

MTRRs in a processor supporting Intel Hyper-Threading Technology are shared by logical processors. When one logical processor updates the setting of the MTRRs, settings are automatically shared with the other logical processors in the same physical package.

The architectures require that all MP systems based on Intel 64 and IA-32 processors (this includes logical processors) must use an identical MTRR memory map. This gives software a consistent view of memory, independent of the processor on which it is running. See Section 12.11, "Memory Type Range Registers (MTRRs)," for information on setting up MTRRs.

9.7.4 Page Attribute Table (PAT)

Each logical processor has its own PAT MSR (IA32_PAT). However, as described in Section 12.12, "Page Attribute Table (PAT)," the PAT MSR settings must be the same for all processors in a system, including the logical processors.

9.7.5 Machine Check Architecture

In the Intel HT Technology context as implemented by processors based on Intel NetBurst[®] microarchitecture, all of the machine check architecture (MCA) MSRs (except for the IA32_MCG_STATUS and IA32_MCG_CAP MSRs) are duplicated for each logical processor. This permits logical processors to initialize, configure, query, and handle machine-check exceptions simultaneously within the same physical processor. The design is compatible with machine check exception handlers that follow the guidelines given in Chapter 16, "Machine-Check Architecture."

The IA32_MCG_STATUS MSR is duplicated for each logical processor so that its machine check in progress bit field (MCIP) can be used to detect recursion on the part of MCA handlers. In addition, the MSR allows each logical processor to determine that a machine-check exception is in progress independent of the actions of another logical processor in the same physical package.

Because the logical processors within a physical package are tightly coupled with respect to shared hardware resources, both logical processors are notified of machine check errors that occur within a given physical processor. If machine-check exceptions are enabled when a fatal error is reported, all the logical processors within a physical package are dispatched to the machine-check exception handler. If machine-check exceptions are disabled, the logical processors enter the shutdown state and assert the IERR# signal.

When enabling machine-check exceptions, the MCE flag in control register CR4 should be set for each logical processor.

On Intel Atom family processors that support Intel Hyper-Threading Technology, the MCA facilities are shared between all logical processors on the same processor core.

9.7.6 Debug Registers and Extensions

Each logical processor has its own set of debug registers (DR0, DR1, DR2, DR3, DR6, DR7) and its own debug control MSR. These can be set to control and record debug information for each logical processor independently. Each logical processor also has its own last branch records (LBR) stack.

9.7.7 Performance Monitoring Counters

Performance counters and their companion control MSRs are shared between the logical processors within a processor core for processors based on Intel NetBurst microarchitecture. As a result, software must manage the use of these resources. The performance counter interrupts, events, and precise event monitoring support can be set up and allocated on a per thread (per logical processor) basis.

See Section 20.6.4, “Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture,” for a discussion of performance monitoring in the Intel Xeon processor MP.

In Intel Atom processor family that support Intel Hyper-Threading Technology, the performance counters (general-purpose and fixed-function counters) and their companion control MSRs are duplicated for each logical processor.

9.7.8 IA32_MISC_ENABLE MSR

The IA32_MISC_ENABLE MSR (MSR address 1A0H) is generally shared between the logical processors in a processor core supporting Intel Hyper-Threading Technology. However, some bit fields within IA32_MISC_ENABLE MSR may be duplicated per logical processor. The partition of shared or duplicated bit fields within IA32_MISC_ENABLE is implementation dependent. Software should program duplicated fields carefully on all logical processors in the system to ensure consistent behavior.

9.7.9 Memory Ordering

The logical processors in an Intel 64 or IA-32 processor supporting Intel Hyper-Threading Technology obey the same rules for memory ordering as Intel 64 or IA-32 processors without Intel HT Technology (see Section 9.2, “Memory Ordering”). Each logical processor uses a processor-ordered memory model that can be further defined as “write-ordered with store buffer forwarding.” All mechanisms for strengthening or weakening the memory-ordering model to handle special programming situations apply to each logical processor.

9.7.10 Serializing Instructions

As a general rule, when a logical processor in a processor supporting Intel Hyper-Threading Technology executes a serializing instruction, only that logical processor is affected by the operation. An exception to this rule is the execution of the WBINVD, INVD, and WRMSR instructions; and the MOV CR instruction when the state of the CD flag in control register CR0 is modified. Here, both logical processors are serialized.

9.7.11 Microcode Update Resources

In an Intel processor supporting Intel Hyper-Threading Technology, the microcode update facilities are shared between the logical processors; either logical processor can initiate an update. Each logical processor has its own BIOS signature MSR (IA32_BIOS_SIGN_ID at MSR address 8BH). When a logical processor performs an update for the physical processor, the IA32_BIOS_SIGN_ID MSRs for resident logical processors are updated with identical information. If logical processors initiate an update simultaneously, the processor core provides the necessary synchronization needed to ensure that only one update is performed at a time.

NOTE

Some processors (prior to the introduction of Intel 64 Architecture and based on Intel NetBurst microarchitecture) do not support simultaneous loading of microcode update to the sibling logical processors in the same core. All other processors support logical processors initiating an update simultaneously. Intel recommends a common approach that the microcode loader use the sequential technique described in Section 10.11.6.3.

9.7.12 Self Modifying Code

Intel processors supporting Intel Hyper-Threading Technology support self-modifying code, where data writes modify instructions cached or currently in flight. They also support cross-modifying code, where on an MP system writes generated by one processor modify instructions cached or currently in flight on another. See Section 9.1.3, "Handling Self- and Cross-Modifying Code," for a description of the requirements for self- and cross-modifying code in an IA-32 processor.

9.7.13 Implementation-Specific Intel® HT Technology Facilities

The following non-architectural facilities are implementation-specific in IA-32 processors supporting Intel Hyper-Threading Technology:

- Caches.
- Translation lookaside buffers (TLBs).
- Thermal monitoring facilities.

The Intel Xeon processor MP implementation is described in the following sections.

9.7.13.1 Processor Caches

For processors supporting Intel Hyper-Threading Technology, the caches are shared. Any cache manipulation instruction that is executed on one logical processor has a global effect on the cache hierarchy of the physical processor. Note the following:

- **WBINVD instruction** — The entire cache hierarchy is invalidated after modified data is written back to memory. All logical processors are stopped from executing until after the write-back and invalidate operation is completed. A special bus cycle is sent to all caching agents. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.
- **INVD instruction** — The entire cache hierarchy is invalidated without writing back modified data to memory. All logical processors are stopped from executing until after the invalidate operation is completed. A special bus cycle is sent to all caching agents.
- **CLFLUSH and CLFLUSHOPT instructions** — The specified cache line is invalidated from the cache hierarchy after any modified data is written back to memory and a bus cycle is sent to all caching agents, regardless of which logical processor caused the cache line to be filled.
- **CD flag in control register CR0** — Each logical processor has its own CR0 control register, and thus its own CD flag in CR0. The CD flags for the two logical processors are ORed together, such that when any logical processor sets its CD flag, the entire cache is nominally disabled.

9.7.13.2 Processor Translation Lookaside Buffers (TLBs)

In processors supporting Intel Hyper-Threading Technology, data cache TLBs are shared. The instruction cache TLB may be duplicated or shared in each logical processor, depending on implementation specifics of different processor families.

Entries in the TLBs are tagged with an ID that indicates the logical processor that initiated the translation. This tag applies even for translations that are marked global using the page-global feature for memory paging. See Section 4.10, "Caching Translation Information," for information about global translations.

When a logical processor performs a TLB invalidation operation, only the TLB entries that are tagged for that logical processor are guaranteed to be flushed. This protocol applies to all TLB invalidation operations, including writes to control registers CR3 and CR4 and uses of the INVLPG instruction.

9.7.13.3 Thermal Monitor

In a processor that supports Intel Hyper-Threading Technology, logical processors share the catastrophic shutdown detector and the automatic thermal monitoring mechanism (see Section 15.8, “Thermal Monitoring and Protection”). Sharing results in the following behavior:

- If the processor’s core temperature rises above the preset catastrophic shutdown temperature, the processor core halts execution, which causes both logical processors to stop execution.
- When the processor’s core temperature rises above the preset automatic thermal monitor trip temperature, the frequency of the processor core is automatically modulated, which effects the execution speed of both logical processors.

For software controlled clock modulation, each logical processor has its own IA32_CLOCK_MODULATION MSR, allowing clock modulation to be enabled or disabled on a logical processor basis. Typically, if software controlled clock modulation is going to be used, the feature must be enabled for all the logical processors within a physical processor and the modulation duty cycle must be set to the same value for each logical processor. If the duty cycle values differ between the logical processors, the processor clock will be modulated at the highest duty cycle selected.

9.7.13.4 External Signal Compatibility

This section describes the constraints on external signals received through the pins of a processor supporting Intel Hyper-Threading Technology and how these signals are shared between its logical processors.

- **STPCLK#** — A single STPCLK# pin is provided on the physical package of the Intel Xeon processor MP. External control logic uses this pin for power management within the system. When the STPCLK# signal is asserted, the processor core transitions to the stop-grant state, where instruction execution is halted but the processor core continues to respond to snoop transactions. Regardless of whether the logical processors are active or halted when the STPCLK# signal is asserted, execution is stopped on both logical processors and neither will respond to interrupts.

In MP systems, the STPCLK# pins on all physical processors are generally tied together. As a result this signal affects all the logical processors within the system simultaneously.

- **LINT0 and LINT1 pins** — A processor supporting Intel Hyper-Threading Technology has only one set of LINT0 and LINT1 pins, which are shared between the logical processors. When one of these pins is asserted, both logical processors respond unless the pin has been masked in the APIC local vector tables for one or both of the logical processors.

Typically in MP systems, the LINT0 and LINT1 pins are not used to deliver interrupts to the logical processors. Instead all interrupts are delivered to the local processors through the I/O APIC.

- **A20M# pin** — On an IA-32 processor, the A20M# pin is typically provided for compatibility with the Intel 286 processor. Asserting this pin causes bit 20 of the physical address to be masked (forced to zero) for all external bus memory accesses. Processors supporting Intel Hyper-Threading Technology provide one A20M# pin, which affects the operation of both logical processors within the physical processor.

The functionality of A20M# is used primarily by older operating systems and not used by modern operating systems. On newer Intel 64 processors, A20M# may be absent.

9.8 MULTI-CORE ARCHITECTURE

This section describes the architecture of Intel 64 and IA-32 processors supporting dual-core and quad-core technology. The discussion is applicable to the Intel Pentium processor Extreme Edition, Pentium D, Intel Core Duo, Intel Core 2 Duo, Dual-core Intel Xeon processor, Intel Core 2 Quad processors, and quad-core Intel Xeon processors. Features vary across different microarchitectures and are detectable using CPUID.

In general, each processor core has dedicated microarchitectural resources identical to a single-processor implementation of the underlying microarchitecture without hardware multi-threading capability. Each logical processor in a dual-core processor (whether supporting Intel Hyper-Threading Technology or not) has its own APIC functionality, PAT, machine check architecture, debug registers and extensions. Each logical processor handles serialization instructions or self-modifying code on its own. Memory order is handled the same way as in Intel Hyper-Threading Technology.

The topology of the cache hierarchy (with respect to whether a given cache level is shared by one or more processor cores or by all logical processors in the physical package) depends on the processor implementation. Software must use the deterministic cache parameter leaf of CPUID instruction to discover the cache-sharing topology between the logical processors in a multi-threading environment.

9.8.1 Logical Processor Support

The topological composition of processor cores and logical processors in a multi-core processor can be discovered using CPUID. Within each processor core, one or more logical processors may be available.

System software must follow the requirement MP initialization sequences (see Section 9.4, “Multiple-Processor (MP) Initialization”) to recognize and enable logical processors. At runtime, software can enumerate those logical processors enabled by system software to identify the topological relationships between these logical processors. (See Section 9.9.5, “Identifying Topological Relationships in a MP System”).

9.8.2 Memory Type Range Registers (MTRR)

MTRR is shared between two logical processors sharing a processor core if the physical processor supports Intel Hyper-Threading Technology. MTRR is not shared between logical processors located in different cores or different physical packages.

The Intel 64 and IA-32 architectures require that all logical processors in an MP system use an identical MTRR memory map. This gives software a consistent view of memory, independent of the processor on which it is running.

See Section 12.11, “Memory Type Range Registers (MTRRs).”

9.8.3 Performance Monitoring Counters

Performance counters and their companion control MSRs are shared between two logical processors sharing a processor core if the processor core supports Intel Hyper-Threading Technology and is based on Intel NetBurst microarchitecture. They are not shared between logical processors in different cores or different physical packages. As a result, software must manage the use of these resources, based on the topology of performance monitoring resources. Performance counter interrupts, events, and precise event monitoring support can be set up and allocated on a per thread (per logical processor) basis.

See Section 20.6.4, “Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture.”

9.8.4 IA32_MISC_ENABLE MSR

Some bit fields in IA32_MISC_ENABLE MSR (MSR address 1A0H) may be shared between two logical processors sharing a processor core, or may be shared between different cores in a physical processor. See Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

9.8.5 Microcode Update Resources

Microcode update facilities are shared between two logical processors sharing a processor core if the physical package supports Intel Hyper-Threading Technology. They are not shared between logical processors in different

cores or different physical packages. Either logical processor that has access to the microcode update facility can initiate an update.

Each logical processor has its own BIOS signature MSR (IA32_BIOS_SIGN_ID at MSR address 8BH). When a logical processor performs an update for the physical processor, the IA32_BIOS_SIGN_ID MSRs for resident logical processors are updated with identical information.

All microcode update steps during processor initialization should use the same update data on all cores in all physical packages of the same stepping. Any subsequent microcode update must apply consistent update data to all cores in all physical packages of the same stepping. If the processor detects an attempt to load an older microcode update when a newer microcode update had previously been loaded, it may reject the older update to stay with the newer update.

NOTE

Some processors (prior to the introduction of Intel 64 Architecture and based on Intel NetBurst microarchitecture) do not support simultaneous loading of microcode update to the sibling logical processors in the same core. All other processors support logical processors initiating an update simultaneously. Intel recommends a common approach that the microcode loader use the sequential technique described in Section 10.11.6.3.

9.9 PROGRAMMING CONSIDERATIONS FOR HARDWARE MULTI-THREADING CAPABLE PROCESSORS

In a multi-threading environment, there may be certain hardware resources that are physically shared at some level of the hardware topology. In the multi-processor systems, typically bus and memory sub-systems are physically shared between multiple sockets. Within a hardware multi-threading capable processors, certain resources are provided for each processor core, while other resources may be provided for each logical processors (see Section 9.7, “Intel® Hyper-Threading Technology Architecture,” and Section 9.8, “Multi-Core Architecture”).

From a software programming perspective, control transfer of processor operation is managed at the granularity of logical processor (operating systems dispatch a runnable task by allocating an available logical processor on the platform). To manage the topology of shared resources in a multi-threading environment, it may be useful for software to understand and manage resources that are shared by more than one logical processors.

9.9.1 Hierarchical Mapping of Shared Resources

The APIC_ID value associated with each logical processor in a multi-processor system is unique (see Section 9.6, “Detecting Hardware Multi-Threading Support and Topology”). This 8-bit or 32-bit value can be decomposed into sub-fields, where each sub-field corresponds a hierarchical **domain** of the topological mapping of hardware resources.

The decomposition of an APIC_ID may consist of several sub fields representing the topology within a physical processor package, the higher-order bits of an APIC ID may also be used by cluster vendors to represent the topology of cluster nodes of each coherent multiprocessor systems:

- **Cluster** — Some multi-threading environments consists of multiple clusters of multi-processor systems. The CLUSTER_ID sub-field is usually supported by vendor firmware to distinguish different clusters. For non-clustered systems, CLUSTER_ID is usually 0 and system topology is reduced.
- **Package** — A physical processor package mates with a socket. A package may contain one or more software visible die. The PACKAGE_ID sub-field distinguishes different physical packages within a cluster.
- **Die** — A software-visible chip inside a package. The DIE_ID sub-field distinguishes different die within a package. If there are no software visible die, the width of this bit field is 0.
- **DieGrp** — A group of die that share certain resources.
- **Tile** — A set of cores that share certain resources. The TILE_ID sub-field distinguishes different tiles. If there are no software visible tiles, the width of this bit field is 0.

- **Module** — A set of cores that share certain resources. The MODULE_ID sub-field distinguishes different modules. If there are no software visible modules, the width of this bit field is 0.
 - **Core** — Processor cores may be contained within modules, within tiles, on software-visible die, or appear directly at the package domain. The CORE_ID sub-field distinguishes processor cores. For a single-core processor, the width of this bit field is 0.
 - **Logical Processor** — A processor core provides one or more logical processors sharing execution resources. The LOGICAL_PROCESSOR_ID sub-field distinguishes logical processors in a core. The width of this bit field is non-zero if a processor core provides more than one logical processors.
- The LOGICAL_PROCESSOR_ID and CORE_ID sub-fields are bit-wise contiguous in the APIC_ID field (see Figure 9-5).

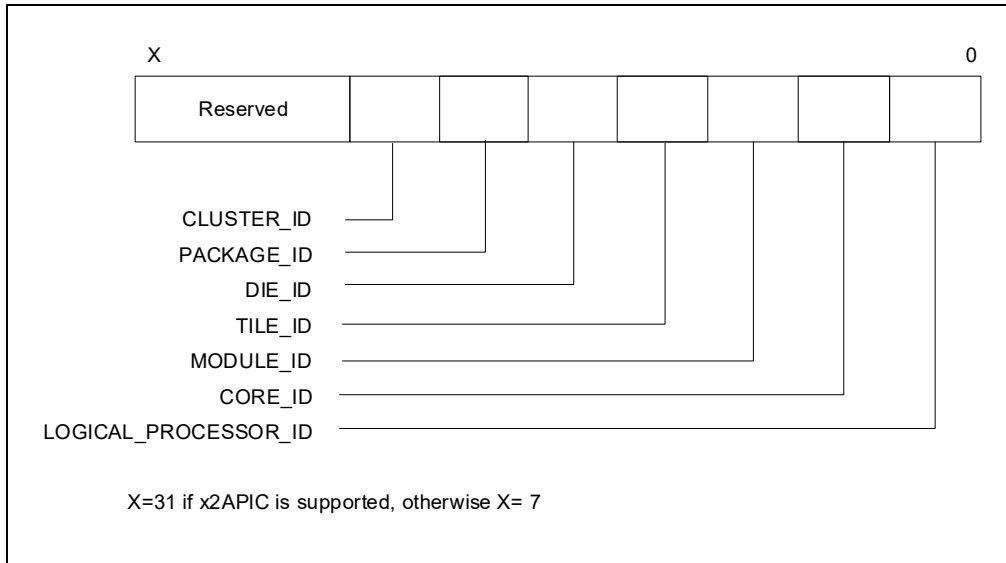


Figure 9-5. Generalized Seven-Domain Interpretation of the APIC ID

If the processor supports CPUID leaf 0BH and leaf 1FH, the 32-bit APIC ID can represent cluster plus several domains of topology within the physical processor package. The exact number of hierarchical domains within a physical processor package must be enumerated through CPUID leaf 0BH and leaf 1FH. Common processor families may employ a topology similar to that represented by the 8-bit Initial APIC ID. In general, CPUID leaf 0BH and leaf 1FH can support a topology enumeration algorithm that decompose a 32-bit APIC ID into more than four sub-fields (see Figure 9-6).

NOTE

CPUID leaf 0BH and leaf 1FH can have differences in the number of domain types reported (CPUID leaf 1FH defines additional domain types). If the processor supports CPUID leaf 1FH, usage of this leaf is preferred over leaf 0BH. CPUID leaf 0BH is available for legacy compatibility going forward.

The width of each sub-field depends on hardware and software configurations. Field widths can be determined at runtime using the algorithm discussed below (Example 9-16 through Example 9-21).

Figure 7-6 depicts the relationships of three of the hierarchical sub-fields in a hypothetical MP system. The value of valid APIC_IDs need not be contiguous across package boundary or core boundaries.

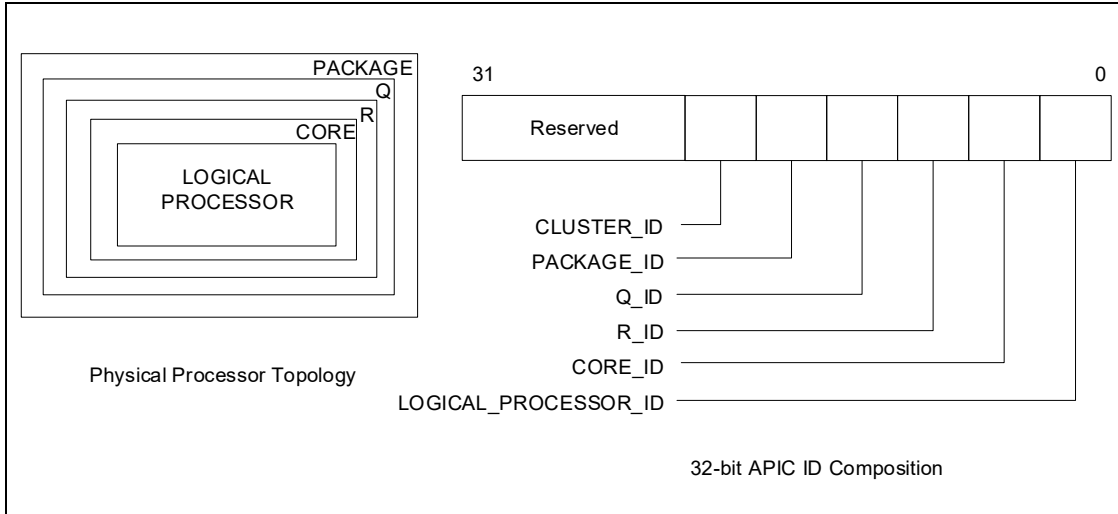


Figure 9-6. Conceptual Six-Domain Topology and 32-bit APIC ID Composition

9.9.2 Hierarchical Mapping of CPUID Extended Topology Leaf

CPUID leaf 0BH and leaf 1FH provide enumeration parameters for software to identify each hierarchy of the processor topology in a deterministic manner. Each hierarchical domain of the topology starting from the Logical Processor domain is represented numerically by a sub-leaf index within the CPUID 0BH leaf and 1FH leaf. Each domain of the topology is mapped to a sub-field in the APIC ID, following the general relationship depicted in Figure 9-6. This mechanism allows software to query the exact number of domains within a physical processor package and the bit-width of each sub-field of x2APIC ID directly. For example,

- Starting from sub-leaf index 0 and incrementing ECX until CPUID.(EAX=0BH or 1FH, ECX=N):ECX[15:8] returns an invalid "domain type" encoding. The number of domains within the physical processor package is "N" (excluding PACKAGE). Using Figure 9-6 as an example, CPUID.(EAX=0BH or 1FH, ECX=4):ECX[15:8] will report 00H, indicating sub leaf 04H is invalid. This is also depicted by a pseudo code example:

Example 9-16. Number of Domains Below the Physical Processor Package

```

Word NumberOfDomainsBelowPackage = 0;
DWord Subleaf = 0;

EAX = 0BH or 1FH; // query each sub leaf of CPUID leaf 0BH or 1FH; CPUID leaf 1FH is preferred over leaf 0BH if available.
ECX = Subleaf;
CPUID;
while(EBX != 0) // Enumerate until EBX reports 0
{
    if(EAX[4:0] != 0) // A Shift Value of 0 indicates this domain does not exist.
                    // (Such as no SMT_ID, which is required entry at sub-leaf 0.)
    {
        NumberOfDomainsBelowPackage++;
    }
    Subleaf++;
    EAX = 0BH or 1FH;
    ECX = Subleaf;
    CPUID;
}
// NumberOfDomainsBelowPackage contains the absolute number of domains that exist below package.
N = Subleaf; // Sub-leaf supplies the number of entries CPUID will return.
    
```


- Sub-leaf index 0 (ECX= 0 as input) provides enumeration parameters to extract the `LOGICAL_PROCESSOR_ID` sub-field of x2APIC ID. If EAX = 0BH or 1FH, and ECX =0 is specified as input when executing CPUID, CPUID.(EAX=0BH or 1FH, ECX=0):EAX[4:0] reports a value (a right-shift count) that allow software to extract part of x2APIC ID to distinguish the next higher topological entities above the `LOGICAL_PROCESSOR_ID domain`. This value also corresponds to the bit-width of the sub-field of x2APIC ID corresponding the hierarchical `domain` with sub-leaf index 0.
- For each subsequent higher sub-leaf index m, CPUID.(EAX=0BH or 1FH, ECX=m):EAX[4:0] reports the right-shift count that will allow software to extract part of x2APIC ID to distinguish higher-`domain` topological entities. This means the right-shift value at of sub-leaf m, corresponds to the least significant (m+1) sub-fields of the 32-bit x2APIC ID.

Example 9-17. BitWidth Determination of x2APIC ID Sub-fields

```
For m = 0, m < N, m ++;
{ cumulative_width[m] = CPUID.(EAX=0BH or 1FH, ECX= m): EAX[4:0]; }
BitWidth[0] = cumulative_width[0];
For m = 1, m < N, m ++;
    BitWidth[m] = cumulative_width[m] - cumulative_width[m-1];
```

NOTE

CPUID leaf 1FH is a preferred superset to leaf 0BH. Leaf 1FH defines additional `domain` types, and it must be parsed by an algorithm that can handle the addition of future `domain` types.

Previously, only the following encoding of hierarchical `domain` types were defined: 0 (invalid), 1 (`logical processor`), and 2 (`core`). With the additional hierarchical `domain` types available (see Section 9.9.1, “Hierarchical Mapping of Shared Resources” and Figure 9-5, “Generalized Seven-Domain Interpretation of the APIC ID”) software must not assume any “`domain type`” encoding value to be related to any sub-leaf index, except sub-leaf 0.

Example 9-18. Support Routines for Identifying Package, Die, Core, and Logical Processors from 32-bit x2APIC ID

- Derive the extraction bitmask for logical processors in a processor core and associated mask offset for different cores.**

```
//
// This example shows how to enumerate CPU topology domain types (domain types may or may not be known/supported by the software)
//
// Below is the list of sample domain types used in the example.
// Refer to the CPUID Leaf 1FH definition for the actual domain type numbers: “V2 Extended Topology Enumeration Leaf” .
//
// LOGICAL_PROCESSOR
// CORE
// MODULE
// TILE
// DIE
// PACKAGE
//
// The example shows how to identify and derive the extraction bitmask for the domains with identify type LOGICAL_PROCESSOR_ID/CORE_ID/DIE_ID/PACKAGE_ID
//
```

```
int DeriveLogical_Processor_Mask_Offsets (void)
{
    IF (!HWMTSupported()) return -1;
```

```

execute cpuid with EAX = 0BH or 1FH, ECX = 0;
IF (returned domain type encoding in ECX[15:8] does not match LOGICAL_PROCESSOR_ID) return -1;
Mask_Logical_Processor_shift = EAX[4:0];    // # bits shift right of APIC ID to distinguish different cores, note this can be a shift
                                             // of zero if there is only one logical processor per core.
Logical Processor Mask = ~( (-1) << Mask_Logical_Processor_shift);    // shift left to derive extraction bitmask for
                                                                    // LOGICAL_PROCESSOR_ID
return 0;
}

```

- b. Derive the extraction bitmask for processor cores in a physical processor package and associated mask offset for different packages.

```

int DeriveCore_Mask_Offsets (void)

```

```

{
    IF (!HWMTSupported()) return -1;
    execute cpuid with EAX = 0BH or 1FH, ECX = 0;
    WHILE( ECX[15:8] ) {                // domain type encoding is valid
        Mask_last_known_shift = EAX[4:0]
        IF (returned domain type encoding in ECX[15:8] matches CORE) {
            Mask_Core_shift = EAX[4:0];
        }
        ELSE IF (returned domain type encoding in ECX[15:8] matches DIE {
            Mask_Die_shift = EAX[4:0];
        }
        //
        // Keep enumerating. Check if the next domain is the desired domain and if not, keep enumerating until you reach a known
        // domain or the invalid domain ("0" domain type). If there are more domains between DIE and PACKAGE, the unknown
        // domains will be ignored and treated as an extension of the last known domain (i.e., DIE in this case).
        //
        ECX++;
        execute cpuid with EAX = 0BH or 1FH;
    }

    COREPlusLogical_Processor_MASK = ~( (-1) << Mask_Core_shift);
    DIEPlusCORE_MASK = ~( (-1) << Mask_Die_shift);

    //
    // Treat domains between DIE and physical package as an extension of DIE for software choosing not to implement or recognize
    // these unknown domains.
    //

    CORE_MASK = COREPlusLogical_Processor_MASK ^ Logical Processor Mask;
    DIE_MASK = DIEPlusCORE_MASK ^ COREPlusLogical_Processor_MASK;
    PACKAGE_MASK = (-1) << Mask_last_known_shift;

    return -1;
}

```

9.9.3 Hierarchical ID of Logical Processors in an MP System

For Intel 64 and IA-32 processors, system hardware establishes an 8-bit initial APIC ID (or 32-bit APIC ID if the processor supports CPUID leaf 0BH) that is unique for each logical processor following power-up or RESET (see

Section 9.6.1). Each logical processor on the system is allocated an initial APIC ID. BIOS may implement features that tell the OS to support less than the total number of logical processors on the system bus. Those logical processors that are not available to applications at runtime are halted during the OS boot process. As a result, the number valid local APIC_IDS that can be queried by `affinitizing-current-thread-context` (See Example 9-23) is limited to the number of logical processors enabled at runtime by the OS boot process.

Table 9-2 shows an example of the 8-bit APIC IDs that are initially reported for logical processors in a system with four Intel Xeon MP processors that support Intel Hyper-Threading Technology (a total of 8 logical processors, each physical package has two processor cores and supports Intel Hyper-Threading Technology). Of the two logical processors within a Intel Xeon processor MP, logical processor 0 is designated the primary logical processor and logical processor 1 as the secondary logical processor.

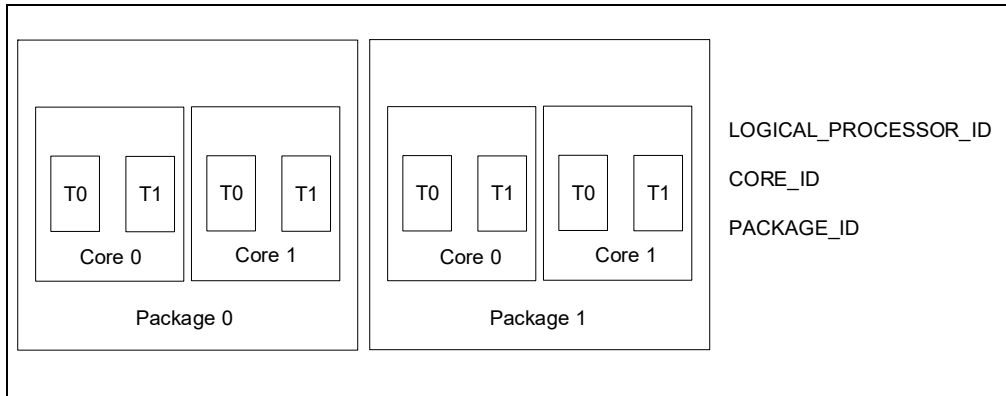


Figure 9-7. Topological Relationships Between Hierarchical IDs in a Hypothetical MP Platform

Table 9-2. Initial APIC IDs for the Logical Processors in a System that has Four Intel Xeon MP Processors Supporting Intel Hyper-Threading Technology¹

Initial APIC ID	PACKAGE_ID	CORE_ID	LOGICAL_PROCESSOR_ID
0H	0H	0H	0H
1H	0H	0H	1H
2H	1H	0H	0H
3H	1H	0H	1H
4H	2H	0H	0H
5H	2H	0H	1H
6H	3H	0H	0H
7H	3H	0H	1H

NOTE:

1. Because information on the number of processor cores in a physical package was not available in early single-core processors supporting Intel Hyper-Threading Technology, the CORE_ID can be treated as 0.

Table 9-3 shows the initial APIC IDs for a hypothetical situation with a dual processor system. Each physical package providing two processor cores, and each processor core also supporting Intel Hyper-Threading Technology.

Table 9-3. Initial APIC IDs for the Logical Processors in a System that has Two Physical Processors Supporting Dual-Core and Intel Hyper-Threading Technology

Initial APIC ID	PACKAGE_ID	CORE_ID	LOGICAL_PROCESSOR_ID
0H	0H	0H	0H

Table 9-3. Initial APIC IDs for the Logical Processors in a System that has Two Physical Processors Supporting Dual-Core and Intel Hyper-Threading Technology

Initial APIC ID	PACKAGE_ID	CORE_ID	LOGICAL_PROCESSOR_ID
1H	0H	0H	1H
2H	0H	1H	0H
3H	0H	1H	1H
4H	1H	0H	0H
5H	1H	0H	1H
6H	1H	1H	0H
7H	1H	1H	1H

9.9.3.1 Hierarchical ID of Logical Processors with x2APIC ID

Table 9-4 shows an example of possible x2APIC ID assignments for a dual processor system that support x2APIC. Each physical package providing four processor cores, and each processor core also supporting Intel Hyper-Threading Technology. Note that the x2APIC ID need not be contiguous in the system.

Table 9-4. Example of Possible x2APIC ID Assignment in a System that has Two Physical Processors Supporting x2APIC and Intel Hyper-Threading Technology

x2APIC ID	PACKAGE_ID	CORE_ID	LOGICAL_PROCESSOR_ID
0H	0H	0H	0H
1H	0H	0H	1H
2H	0H	1H	0H
3H	0H	1H	1H
4H	0H	2H	0H
5H	0H	2H	1H
6H	0H	3H	0H
7H	0H	3H	1H
10H	1H	0H	0H
11H	1H	0H	1H
12H	1H	1H	0H
13H	1H	1H	1H
14H	1H	2H	0H
15H	1H	2H	1H
16H	1H	3H	0H
17H	1H	3H	1H

9.9.4 Algorithm for Three-Domain Mappings of APIC_ID

Software can gather the initial APIC_IDs for each logical processor supported by the operating system at runtime¹ and extract identifiers corresponding to the three domains of sharing topology (package, core, and logical

1. As noted in Section 9.6 and Section 9.9.3, the number of logical processors supported by the OS at runtime may be less than the total number logical processors available in the platform hardware.

processor). The three-domain algorithms below focus on a non-clustered MP system for simplicity. They do not assume APIC IDs are contiguous or that all logical processors on the platform are enabled.

Intel supports multi-threading systems where all physical processors report identical values in CPUID leaf 0BH, CPUID.1:EBX[23:16], CPUID.4¹:EAX[31:26], and CPUID.4²:EAX[25:14]. The algorithms below assume the target system has symmetry across physical package boundaries with respect to the number of logical processors per package, number of cores per package, and cache topology within a package.

Software can choose to assume three-domain hierarchy if it was developed to understand only three domains. However, software implementation needs to ensure it does not break if it runs on systems that have more domains in the hierarchy even if it does not recognize them.

The extraction algorithm (for three-domain mappings from an APIC ID) uses the general procedure depicted in Example 9-19, and is supplemented by more detailed descriptions on the derivation of topology enumeration parameters for extraction bit masks:

1. Detect hardware multi-threading support in the processor.
2. Derive a set of bit masks that can extract the sub ID of each hierarchical domain of the topology. The algorithm to derive extraction bit masks for LOGICAL_PROCESSOR_ID/CORE_ID/PACKAGE_ID differs based on APIC ID is 32-bit (see step 3 below) or 8-bit (see step 4 below).
3. If the processor supports CPUID leaf 0BH, each APIC ID contains a 32-bit value, the topology enumeration parameters needed to derive three-domain extraction bit masks are:
 - a. Query the right-shift value for the LOGICAL_PROCESSOR_ID domain of the topology using CPUID leaf 0BH with ECX = 0H as input. The number of bits to shift-right on x2APIC ID (EAX[4:0]) can distinguish different higher-domain entities above logical processor in the same physical package. This is also the width of the bit mask to extract the LOGICAL_PROCESSOR_ID. The shift value may be 0 and enumerate no logical processor bit mask to create. A platform where cores only have one logical processor are not required to enumerate a separate bit layout for logical processor, and the lowest bits may only identify the core (where core and logical processor are then synonymous).
 - b. Enumerate until the desired domain is found (i.e., processor cores). Determine if the next domain is the expected domain. If the next domain is not known to the software, keep enumerating until the next known or the last domain. Software should use the previous domain before this to represent the last previously known domain (i.e., processor cores). If the software does not recognize or implement certain hierarchical domains, it should assume these unknown domains as an extension of the last known domain.
 - c. Query CPUID leaf 0BH for the amount of bit shift to distinguish next higher-domain entities (e.g., physical processor packages) in the system. This describes an explicit three-domain-topology situation for commonly available processors. Consult Example 9-17 to adapt to situations beyond a three-domain topology of a physical processor. The width of the extraction bit mask can be used to derive the cumulative extraction bitmask to extract the sub IDs of logical processors (including different processor cores) in the same physical package. The extraction bit mask to distinguish merely different processor cores can be derived by xor'ing the logical processor extraction bit mask from the cumulative extraction bit mask.
 - d. Query the 32-bit x2APIC ID for the logical processor where the current thread is executing.
 - e. Derive the extraction bit masks corresponding to LOGICAL_PROCESSOR_ID, CORE_ID, and PACKAGE_ID, starting from LOGICAL_PROCESSOR_ID.
 - f. Apply each extraction bit mask to the 32-bit x2APIC ID to extract sub-field IDs.
4. If the processor does not support CPUID leaf 0BH, each initial APIC ID contains an 8-bit value, the topology enumeration parameters needed to derive extraction bit masks are:
 - a. Query the size of address space for sub IDs that can accommodate logical processors in a physical processor package. This size parameters (CPUID.1:EBX[23:16]) can be used to derive the width of an extraction bitmask to enumerate the sub IDs of different logical processors in the same physical package.

-
1. Maximum number of addressable ID for processor cores in a physical processor is obtained by executing CPUID with EAX=4 and a valid ECX index. The ECX index starts at 0.
 2. Maximum number addressable ID for processor cores sharing the target cache level is obtained by executing CPUID with EAX = 4 and the ECX index corresponding to the target cache level.

- b. Query the size of address space for sub IDs that can accommodate processor cores in a physical processor package. This size parameters can be used to derive the width of an extraction bitmask to enumerate the sub IDs of processor cores in the same physical package.
- c. Query the 8-bit initial APIC ID for the logical processor where the current thread is executing.
- d. Derive the extraction bit masks using respective address sizes corresponding to `LOGICAL_PROCESSOR_ID`, `CORE_ID`, and `PACKAGE_ID`, starting from `LOGICAL_PROCESSOR_ID`.
- e. Apply each extraction bit mask to the 8-bit initial APIC ID to extract sub-field IDs.

Example 9-19. Support Routines for Detecting Hardware Multi-Threading and Identifying the Relationships Between Package, Core, and Logical Processors

1. Detect support for Hardware Multi-Threading Support in a processor.

```
// Returns a non-zero value if CPUID reports the presence of hardware multi-threading
// support in the physical package where the current logical processor is located.
// This does not guarantee BIOS or OS will enable all logical processors in the physical
// package and make them available to applications.
// Returns zero if hardware multi-threading is not present.
```

```
#define HWMT_BIT 10000000H
```

```
unsigned int HWMTSupported(void)
{
    // ensure cpuid instruction is supported
    // execute cpuid with eax = 0 to get vendor string
    // execute cpuid with eax = 1 to get feature flag and signature

    // Check to see if this a Genuine Intel Processor

    if (vendor string EQ GenuineIntel) {
        return (feature_flag_edx & HWMT_BIT); // bit 28
    }
    return 0;
}
```

Example 9-20. Support Routines for Identifying Package, Core, and Logical Processors from 32-bit x2APIC ID

a. Derive the extraction bitmask for logical processors in a processor core and associated mask offset for different cores.

```
int DeriveLogical_Processor_Mask_Offsets (void)
{
    if (!HWMTSupported()) return -1;
    execute cpuid with eax = 11, ECX = 0;
    If (returned domain type encoding in ECX[15:8] does not match logical processor) return -1;
    Mask_Logical_Processor_shift = EAX[4:0]; // # bits shift right of APIC ID to distinguish different cores, note this can be a shift
    // of zero if there is only one logical processor per core.
    Logical_Processor_Mask = ~( (-1) << Mask_Logical_Processor_shift); // shift left to derive extraction bitmask for
    // LOGICAL_PROCESSOR_ID

    return 0;
}
```

- b. Derive the extraction bitmask for processor cores in a physical processor package and associated mask offset for different packages.

```
int DeriveCore_Mask_Offsets (void)
{
    if (!HWMTSupported()) return -1;
    execute cpuid with eax = 11, ECX = 0;
    while( ECX[15:8] ) { // domain type encoding is valid
        Mask_Core_shift = EAX[4:0]; // needed to distinguish different physical packages
        ECX ++;
        execute cpuid with eax = 11;
    }
    COREPlusLogical_Processor_MASK = ~( -1 ) << Mask_Core_shift;
    // treat domains between core and physical package as a core for software choosing not to implement or recognize
    // these unknown domains
    CORE_MASK = COREPlusLogical_Processor_MASK ^ Logical Processor Mask;
    PACKAGE_MASK = (-1) << Mask_Core_shift;
    return -1;
}
```

- c. Query the x2APIC ID of a logical processor.

APIC_IDs for each logical processor.

```
unsigned char Getx2APIC_ID (void)
{
    unsigned reg_edx = 0;
    execute cpuid with eax = 11, ECX = 0
    store returned value of edx
    return (unsigned) (reg_edx);
}
```

Example 9-21. Support Routines for Identifying Package, Core, and Logical Processors from 8-bit Initial APIC ID

- a. Find the size of address space for logical processors in a physical processor package.

```
#define NUM_LOGICAL_BITS 00FF0000H
// Use the mask above and CPUID.1.EBX[23:16] to obtain the max number of addressable IDs
// for logical processors in a physical package,

//Returns the size of address space of logical processors in a physical processor package;
// Software should not assume the value to be a power of 2.

unsigned char MaxLPIDsPerPackage(void)
{
    if (!HWMTSupported()) return 1;
    execute cpuid with eax = 1
    store returned value of ebx
    return (unsigned char) ((reg_ebx & NUM_LOGICAL_BITS) >> 16);
}
```

b. Find the size of address space for processor cores in a physical processor package.

// Returns the max number of addressable IDs for processor cores in a physical processor package;
 // Software should not assume cpuid reports this value to be a power of 2.

```
unsigned MaxCoreIDsPerPackage(void)
{
    if (!HWMTSupported()) return (unsigned char) 1;
    if cpuid supports leaf number 4
    { // we can retrieve multi-core topology info using leaf 4
        execute cpuid with eax = 4, ecx = 0
        store returned value of eax
        return (unsigned) ((reg_eax >> 26) + 1);
    }
    else // must be a single-core processor
        return 1;
}
```

c. Query the initial APIC ID of a logical processor.

#define INITIAL_APIC_ID_BITS FF00000H // CPUID.1.EBX[31:24] initial APIC ID

// Returns the 8-bit unique initial APIC ID for the processor running the code.
 // Software can use OS services to affinitize the current thread to each logical processor
 // available under the OS to gather the initial APIC_IDs for each logical processor.

```
unsigned GetInitAPIC_ID (void)
{
    unsigned int reg_ebx = 0;
    execute cpuid with eax = 1
    store returned value of ebx
    return (unsigned) ((reg_ebx & INITIAL_APIC_ID_BITS) >> 24);
}
```

d. Find the width of an extraction bitmask from the maximum count of the bit-field (address size).

// Returns the mask bit width of a bit field from the maximum count that bit field can represent.
 // This algorithm does not assume 'address size' to have a value equal to power of 2.
 // Address size for LOGICAL_PROCESSOR_ID can be calculated from MaxLPIDsPerPackage()/MaxCoreIDsPerPackage()
 // Then use the routine below to derive the corresponding width of logical processor extraction bitmask
 // Address size for CORE_ID is MaxCoreIDsPerPackage(),
 // Derive the bitwidth for CORE extraction mask similarly

```
unsigned FindMaskWidth(Unsigned Max_Count)
{unsigned int mask_width, cnt = Max_Count;
    __asm {
        mov eax, cnt
        mov ecx, 0
        mov mask_width, ecx
        dec eax
        bsr cx, ax
        jz next
        inc cx
        mov mask_width, ecx
        next:
        mov eax, mask_width
    }
```



```

    }
    return mask_width;
}

```

e. Extract a sub ID from an 8-bit full ID, using address size of the sub ID and shift count.

```

// The routine below can extract LOGICAL_PROCESSOR_ID, CORE_ID, and PACKAGE_ID respectively from the init APIC_ID
// To extract LOGICAL_PROCESSOR_ID, MaxSubIDvalue is set to the address size of LOGICAL_PROCESSOR_ID, Shift_Count = 0
// To extract CORE_ID, MaxSubIDvalue is the address size of CORE_ID, Shift_Count is width of logical processor extraction bitmask.
// Returns the value of the sub ID, this is not a zero-based value

```

```

Unsigned char GetSubID(unsigned char Full_ID, unsigned char MaxSubIDValue, unsigned char Shift_Count)
{
    MaskWidth = FindMaskWidth(MaxSubIDValue);
    MaskBits = ((uchar) (FFH << Shift_Count)) ^ ((uchar) (FFH << Shift_Count + MaskWidth));
    SubID = Full_ID & MaskBits;
    Return SubID;
}

```

Software must not assume local APIC_ID values in an MP system are consecutive. Non-consecutive local APIC_IDs may be the result of hardware configurations or debug features implemented in the BIOS or OS.

An identifier for each hierarchical domain can be extracted from an 8-bit APIC_ID using the support routines illustrated in Example 9-21. The appropriate bit mask and shift value to construct the appropriate bit mask for each domain must be determined dynamically at runtime.

9.9.5 Identifying Topological Relationships in an MP System

To detect the number of physical packages, processor cores, or other topological relationships in a MP system, the following procedures are recommended:

- Extract the three-domain identifiers from the APIC ID of each logical processor enabled by system software. The sequence is as follows (see the pseudo code shown in Example 9-22 and support routines shown in Example 9-19):
 - The extraction start from the right-most bit field, corresponding to LOGICAL_PROCESSOR_ID, the innermost hierarchy in a three-domain topology (See Figure 9-7). For the right-most bit field, the shift value of the working mask is zero. The width of the bit field is determined dynamically using the maximum number of logical processor per core, which can be derived from information provided from CPUID.
 - To extract the next bit-field, the shift value of the working mask is determined from the width of the bit mask of the previous step. The width of the bit field is determined dynamically using the maximum number of cores per package.
 - To extract the remaining bit-field, the shift value of the working mask is determined from the maximum number of logical processor per package. So the remaining bits in the APIC ID (excluding those bits already extracted in the two previous steps) are extracted as the third identifier. This applies to a non-clustered MP system, or if there is no need to distinguish between PACKAGE_ID and CLUSTER_ID.

If there is need to distinguish between PACKAGE_ID and CLUSTER_ID, PACKAGE_ID can be extracted using an algorithm similar to the extraction of CORE_ID, assuming the number of physical packages in each node of a clustered system is symmetric.
- Assemble the three-domain identifiers of LOGICAL_PROCESSOR_ID, CORE_ID, PACKAGE_IDs into arrays for each enabled logical processor. This is shown in Example 9-23a.
- To detect the number of physical packages: use PACKAGE_ID to identify those logical processors that reside in the same physical package. This is shown in Example 9-23b. This example also depicts a technique to construct a mask to represent the logical processors that reside in the same package.

- To detect the number of processor cores: use CORE_ID to identify those logical processors that reside in the same core. This is shown in Example 9-23. This example also depicts a technique to construct a mask to represent the logical processors that reside in the same core.

In Example 9-22, the numerical ID value can be obtained from the value extracted with the mask by shifting it right by shift count. Algorithms below do not shift the value. The assumption is that the SubID values can be compared for equivalence without the need to shift.

Example 9-22. Pseudo Code Depicting Three-Domain Extraction Algorithm

```

For Each local_APIC_ID{
    // Calculate Logical Processor Mask, the bit mask pattern to extract LOGICAL_PROCESSOR_ID,
    // Logical Processor Mask is determined using topology enumeration parameters
    // from CPUID leaf 0BH (Example 9-20);
    // otherwise, Logical Processor Mask is determined using CPUID leaf 01H and leaf 04H (Example 9-21).
    // This algorithm assumes there is symmetry across core boundary, i.e., each core within a
    // package has the same number of logical processors
    // LOGICAL_PROCESSOR_ID always starts from bit 0, corresponding to the right-most bit-field
    LOGICAL_PROCESSOR_ID = APIC_ID & Logical Processor Mask;

// Extract CORE_ID:
    // Core Mask is determined in Example 9-20 or Example 9-21
    CORE_ID = (APIC_ID & Core Mask);

    // Extract PACKAGE_ID:
    // Assume single cluster.
    // Shift out the mask width for maximum logical processors per package
    // Package Mask is determined in Example 9-20 or Example 9-21
    PACKAGE_ID = (APIC_ID & Package Mask);
}
    
```

Example 9-23. Compute the Number of Packages, Cores, and Processor Relationships in a MP System

- a) Assemble lists of PACKAGE_ID, CORE_ID, and LOGICAL_PROCESSOR_ID of each enabled logical processors

```

// The BIOS and/or OS may limit the number of logical processors available to applications after system boot.
// The below algorithm will compute topology for the processors visible to the thread that is computing it.

// Extract the 3-domains of IDs on every processor.
// SystemAffinity is a bitmask of all the processors started by the OS. Use OS specific APIs to obtain it.
// ThreadAffinityMask is used to affinityize the topology enumeration thread to each processor using OS specific APIs.
// Allocate per processor arrays to store the Package_ID, Core_ID, and LOGICAL_PROCESSOR_ID for every started processor.

ThreadAffinityMask = 1;
ProcessorNum = 0;
while (ThreadAffinityMask != 0 && ThreadAffinityMask <= SystemAffinity) {
    // Check to make sure we can utilize this processor first.
    if (ThreadAffinityMask & SystemAffinity){
        Set thread to run on the processor specified in ThreadAffinityMask
        Wait if necessary and ensure thread is running on specified processor

        APIC_ID = GetAPIC_ID(); // 32 bit ID in Example 9-20 or 8-bit ID in Example 9-21
        Extract the Package_ID, Core_ID, and LOGICAL_PROCESSOR_ID as explained in three domain extraction
        algorithm of Example 9-22
        PackageID[ProcessorNUM] = PACKAGE_ID;
        CoreID[ProcessorNum] = CORE_ID;
    }
}
    
```

```

        LOGICAL_PROCESSOR_ID[ProcessorNum] = LOGICAL_PROCESSOR_ID;
        ProcessorNum++;
    }
    ThreadAffinityMask <<= 1;
}
NumStartedLPs = ProcessorNum;

```

b) Using the list of PACKAGE_ID to count the number of physical packages in a MP system and construct, for each package, a multi-bit mask corresponding to those logical processors residing in the same package.

```

// Compute the number of packages by counting the number of processors with unique PACKAGE_IDs in the PackageID array.
// Compute the mask of processors in each package.

// PackageIDBucket is an array of unique PACKAGE_ID values. Allocate an array of NumStartedLPs count of entries in this array.
// PackageProcessorMask is a corresponding array of the bit mask of processors belonging to the same package, these are
// processors with the same PACKAGE_ID.
// The algorithm below assumes there is symmetry across package boundary if more than one socket is populated in an MP
//system.
// Bucket Package IDs and compute processor mask for every package.

PackageNum = 1;
PackageIDBucket[0] = PackageID[0];
ProcessorMask = 1;
PackageProcessorMask[0] = ProcessorMask;
For (ProcessorNum = 1; ProcessorNum < NumStartedLPs; ProcessorNum++) {
    ProcessorMask <<= 1;
    For (i=0; i < PackageNum; i++) {
        // we may be comparing bit-fields of logical processors residing in different
        // packages, the code below assume package symmetry
        If (PackageID[ProcessorNum] = PackageIDBucket[i]) {
            PackageProcessorMask[i] |= ProcessorMask;
            Break; // found in existing bucket, skip to next iteration
        }
    }
    if (i = PackageNum) {
        //PACKAGE_ID did not match any bucket, start new bucket
        PackageIDBucket[i] = PackageID[ProcessorNum];
        PackageProcessorMask[i] = ProcessorMask;
        PackageNum++;
    }
}
// PackageNum has the number of Packages started in OS
// PackageProcessorMask[] array has the processor set of each package

```

c) Using the list of CORE_ID to count the number of cores in a MP system and construct, for each core, a multi-bit mask corresponding to those logical processors residing in the same core.

Processors in the same core can be determined by bucketing the processors with the same PACKAGE_ID and CORE_ID. Note that code below can BIT OR the values of PACKAGE and CORE ID because they have not been shifted right.

The algorithm below assumes there is symmetry across package boundary if more than one socket is populated in an MP system.

```

//Bucketing PACKAGE and CORE IDs and computing processor mask for every core
CoreNum = 1;
CoreIDBucket[0] = PackageID[0] | CoreID[0];
ProcessorMask = 1;

```

```

CoreProcessorMask[0] = ProcessorMask;
For (ProcessorNum = 1; ProcessorNum < NumStartedLPs; ProcessorNum++) {
    ProcessorMask << = 1;
    For (i=0; i < CoreNum; i++) {
        // we may be comparing bit-fields of logical processors residing in different
        // packages, the code below assume package symmetry
        If ((PackageID[ProcessorNum] | CoreID[ProcessorNum]) = CoreIDBucket[i]) {
            CoreProcessorMask[i] = ProcessorMask;
            Break; // found in existing bucket, skip to next iteration
        }
    }
    if (i = CoreNum) {
        //Did not match any bucket, start new bucket
        CoreIDBucket[i] = PackageID[ProcessorNum] | CoreID[ProcessorNum];
        CoreProcessorMask[i] = ProcessorMask;
        CoreNum++;
    }
}
// CoreNum has the number of cores started in the OS
// CoreProcessorMask[] array has the processor set of each core

```

Other processor relationships such as processor mask of sibling cores can be computed from set operations of the PackageProcessorMask[] and CoreProcessorMask[].

The algorithm shown above can be adapted to work with earlier generations of single-core IA-32 processors that support Intel Hyper-Threading Technology and in situations that the deterministic cache parameter leaf is not supported (provided CPUID supports initial APIC ID). A reference code example is available (see Intel® 64 Architecture Processor Topology Enumeration).

9.10 MANAGEMENT OF IDLE AND BLOCKED CONDITIONS

When a logical processor in an MP system (including multi-core processor or processors supporting Intel Hyper-Threading Technology) is idle (no work to do) or blocked (on a lock or semaphore), additional management of the core execution engine resource can be accomplished by using the HLT (halt), PAUSE, or the MONITOR/MWAIT instructions.

9.10.1 HLT Instruction

The HLT instruction stops the execution of the logical processor on which it is executed and places it in a halted state until further notice (see the description of the HLT instruction in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). When a logical processor is halted, active logical processors continue to have full access to the shared resources within the physical package. Here shared resources that were being used by the halted logical processor become available to active logical processors, allowing them to execute at greater efficiency. When the halted logical processor resumes execution, shared resources are again shared among all active logical processors. (See Section 9.10.6.3, "Halt Idle Logical Processors," for more information about using the HLT instruction with processors supporting Intel Hyper-Threading Technology.)

9.10.2 PAUSE Instruction

The PAUSE instruction can improve the performance of processors supporting Intel Hyper-Threading Technology when executing "spin-wait loops" and other routines where one thread is accessing a shared lock or semaphore in a tight polling loop. When executing a spin-wait loop, the processor can suffer a severe performance penalty when exiting the loop because it detects a possible memory order violation and flushes the core processor's pipeline. The PAUSE instruction provides a hint to the processor that the code sequence is a spin-wait loop. The processor uses

this hint to avoid the memory order violation and prevent the pipeline flush. In addition, the PAUSE instruction depipelined the spin-wait loop to prevent it from consuming execution resources excessively and consume power needlessly. (See Section 9.10.6.1, “Use the PAUSE Instruction in Spin-Wait Loops,” for more information about using the PAUSE instruction with IA-32 processors supporting Intel Hyper-Threading Technology.)

9.10.3 Detecting Support MONITOR/MWAIT Instruction

Streaming SIMD Extensions 3 introduced two instructions (MONITOR and MWAIT) to help multithreaded software improve thread synchronization. In the initial implementation, MONITOR and MWAIT are available to software at ring 0. The instructions are conditionally available at levels greater than 0. Use the following steps to detect the availability of MONITOR and MWAIT:

- Use CPUID to query the MONITOR bit (CPUID.1.ECX[3] = 1).
- If CPUID indicates support, execute MONITOR inside a TRY/EXCEPT exception handler and trap for an exception. If an exception occurs, MONITOR and MWAIT are not supported at a privilege level greater than 0. See Example 9-24.

Example 9-24. Verifying MONITOR/MWAIT Support

```
boolean MONITOR_MWAIT_works = TRUE;
try {
    _asm {
        xor ecx, ecx
        xor edx, edx
        mov eax, MemArea
        monitor
    }
    // Use monitor
} except (UNWIND) {
    // if we get here, MONITOR/MWAIT is not supported
    MONITOR_MWAIT_works = FALSE;
}
```

9.10.4 MONITOR/MWAIT Instruction

Operating systems usually implement idle loops to handle thread synchronization. In a typical idle-loop scenario, there could be several “busy loops” and they would use a set of memory locations. An impacted processor waits in a loop and poll a memory location to determine if there is available work to execute. The posting of work is typically a write to memory (the work-queue of the waiting processor). The time for initiating a work request and getting it scheduled is on the order of a few bus cycles.

From a resource sharing perspective (logical processors sharing execution resources), use of the HLT instruction in an OS idle loop is desirable but has implications. Executing the HLT instruction on a idle logical processor puts the targeted processor in a non-execution state. This requires another processor (when posting work for the halted logical processor) to wake up the halted processor using an inter-processor interrupt. The posting and servicing of such an interrupt introduces a delay in the servicing of new work requests.

In a shared memory configuration, exits from busy loops usually occur because of a state change applicable to a specific memory location; such a change tends to be triggered by writes to the memory location by another agent (typically a processor).

MONITOR/MWAIT complement the use of HLT and PAUSE to allow for efficient partitioning and un-partitioning of shared resources among logical processors sharing physical resources. MONITOR sets up an effective address range that is monitored for write-to-memory activities; MWAIT places the processor in an optimized state (this may vary between different implementations) until a write to the monitored address range occurs.

In the initial implementation of MONITOR and MWAIT, they are available at CPL = 0 only.

Both instructions rely on the state of the processor's monitor hardware. The monitor hardware can be either armed (by executing the MONITOR instruction) or triggered (due to a variety of events, including a store to the monitored memory region). If upon execution of MWAIT, monitor hardware is in a triggered state: MWAIT behaves as a NOP and execution continues at the next instruction in the execution stream. The state of monitor hardware is not architecturally visible except through the behavior of MWAIT.

Multiple events other than a write to the triggering address range can cause a processor that executed MWAIT to wake up. These include events that would lead to voluntary or involuntary context switches, such as:

- External interrupts, including NMI, SMI, INIT, BINIT, MCERR, A20M#
- Faults, Aborts (including Machine Check)
- Architectural TLB invalidations including writes to CR0, CR3, CR4, and certain MSR writes; execution of LMSW (occurring prior to issuing MWAIT but after setting the monitor)
- Voluntary transitions due to fast system call and far calls (occurring prior to issuing MWAIT but after setting the monitor)

Power management related events (such as Thermal Monitor 2 or chipset driven STPCLK# assertion) will not cause the monitor event pending flag to be cleared. Faults will not cause the monitor event pending flag to be cleared.

Software should not allow for voluntary context switches in between MONITOR/MWAIT in the instruction flow. Note that execution of MWAIT does not re-arm the monitor hardware. This means that MONITOR/MWAIT need to be executed in a loop. Also note that exits from the MWAIT state could be due to a condition other than a write to the triggering address; software should explicitly check the triggering data location to determine if the write occurred. Software should also check the value of the triggering address following the execution of the monitor instruction (and prior to the execution of the MWAIT instruction). This check is to identify any writes to the triggering address that occurred during the course of MONITOR execution.

The address range provided to the MONITOR instruction must be of write-back caching type. Only write-back memory type stores to the monitored address range will trigger the monitor hardware. If the address range is not in memory of write-back type, the address monitor hardware may not be set up properly or the monitor hardware may not be armed. Software is also responsible for ensuring that

- Writes that are not intended to cause the exit of a busy loop do not write to a location within the address region being monitored by the monitor hardware,
- Writes intended to cause the exit of a busy loop are written to locations within the monitored address region.

Not doing so will lead to more false wakeups (an exit from the MWAIT state not due to a write to the intended data location). These have negative performance implications. It might be necessary for software to use padding to prevent false wakeups. CPUID provides a mechanism for determining the size data locations for monitoring as well as a mechanism for determining the size of a the pad.

9.10.5 Monitor/Mwait Address Range Determination

To use the MONITOR/MWAIT instructions, software should know the length of the region monitored by the MONITOR/MWAIT instructions and the size of the coherence line size for cache-snoop traffic in a multiprocessor system. This information can be queried using the CPUID monitor leaf function (EAX = 05H). You will need the smallest and largest monitor line size:

- To avoid missed wake-ups: make sure that the data structure used to monitor writes fits within the smallest monitor line-size. Otherwise, the processor may not wake up after a write intended to trigger an exit from MWAIT.
- To avoid false wake-ups; use the largest monitor line size to pad the data structure used to monitor writes. Software must make sure that beyond the data structure, no unrelated data variable exists in the triggering area for MWAIT. A pad may be needed to avoid this situation.

These above two values bear no relationship to cache line size in the system and software should not make any assumptions to that effect. Within a single-cluster system, the two parameters should default to be the same (the size of the monitor triggering area is the same as the system coherence line size).

Based on the monitor line sizes returned by the CPUID, the OS should dynamically allocate structures with appropriate padding. If static data structures must be used by an OS, attempt to adapt the data structure and use a

dynamically allocated data buffer for thread synchronization. When the latter technique is not possible, consider not using MONITOR/MWAIT when using static data structures.

To set up the data structure correctly for MONITOR/MWAIT on multi-clustered systems: interaction between processors, chipsets, and the BIOS is required (system coherence line size may depend on the chipset used in the system; the size could be different from the processor's monitor triggering area). The BIOS is responsible to set the correct value for system coherence line size using the IA32_MONITOR_FILTER_LINE_SIZE MSR. Depending on the relative magnitude of the size of the monitor triggering area versus the value written into the IA32_MONITOR_FILTER_LINE_SIZE MSR, the smaller of the parameters will be reported as the *Smallest Monitor Line Size*. The larger of the parameters will be reported as the *Largest Monitor Line Size*.

9.10.6 Required Operating System Support

This section describes changes that must be made to an operating system to run on processors supporting Intel Hyper-Threading Technology. It also describes optimizations that can help an operating system make more efficient use of the logical processors sharing execution resources. The required changes and suggested optimizations are representative of the types of modifications that appear in Windows* XP and Linux* kernel 2.4.0 operating systems for Intel processors supporting Intel Hyper-Threading Technology. Additional optimizations for processors supporting Intel Hyper-Threading Technology are described in the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

9.10.6.1 Use the PAUSE Instruction in Spin-Wait Loops

Intel recommends that a PAUSE instruction be placed in all spin-wait loops that run on Intel processors supporting Intel Hyper-Threading Technology and multi-core processors.

Software routines that use spin-wait loops include multiprocessor synchronization primitives (spin-locks, semaphores, and mutex variables) and idle loops. Such routines keep the processor core busy executing a load-compare-branch loop while a thread waits for a resource to become available. Including a PAUSE instruction in such a loop greatly improves efficiency (see Section 9.10.2, "PAUSE Instruction"). The following routine gives an example of a spin-wait loop that uses a PAUSE instruction:

```
Spin_Lock:
    CMP lockvar, 0    ;Check if lock is free
    JE Get_Lock
    PAUSE            ;Short delay
    JMP Spin_Lock
Get_Lock:
    MOV EAX, 1
    XCHG EAX, lockvar ;Try to get lock
    CMP EAX, 0      ;Test if successful
    JNE Spin_Lock
Critical_Section:
    <critical section code>
    MOV lockvar, 0
    ...
```

Continue:

The spin-wait loop above uses a "test, test-and-set" technique for determining the availability of the synchronization variable. This technique is recommended when writing spin-wait loops.

In IA-32 processor generations earlier than the Pentium 4 processor, the PAUSE instruction is treated as a NOP instruction.

9.10.6.2 Potential Usage of MONITOR/MWAIT in C0 Idle Loops

An operating system may implement different handlers for different idle states. A typical OS idle loop on an ACPI-compatible OS is shown in Example 9-25:

Example 9-25. A Typical OS Idle Loop

// WorkQueue is a memory location indicating there is a thread
 // ready to run. A non-zero value for WorkQueue is assumed to
 // indicate the presence of work to be scheduled on the processor.
 // The idle loop is entered with interrupts disabled.

```
WHILE (1) {
  IF (WorkQueue) THEN {
    // Schedule work at WorkQueue.
  }
  ELSE {
    // No work to do - wait in appropriate C-state handler depending
    // on Idle time accumulated
    IF (IdleTime >= IdleTimeThreshold) THEN {
      // Call appropriate C1, C2, C3 state handler, C1 handler
      // shown below
    }
  }
}
// C1 handler uses a Halt instruction
VOID C1Handler()
{ STI
  HLT
}
```

The MONITOR and MWAIT instructions may be considered for use in the C0 idle state loops, if MONITOR and MWAIT are supported.

Example 9-26. An OS Idle Loop with MONITOR/MWAIT in the C0 Idle Loop

// WorkQueue is a memory location indicating there is a thread
 // ready to run. A non-zero value for WorkQueue is assumed to
 // indicate the presence of work to be scheduled on the processor.
 // The following example assumes that the necessary padding has been
 // added surrounding WorkQueue to eliminate false wakeups
 // The idle loop is entered with interrupts disabled.

```
WHILE (1) {
  IF (WorkQueue) THEN {
    // Schedule work at WorkQueue.
  }
  ELSE {
    // No work to do - wait in appropriate C-state handler depending
    // on Idle time accumulated.
    IF (IdleTime >= IdleTimeThreshold) THEN {
      // Call appropriate C1, C2, C3 state handler, C1
      // handler shown below
      MONITOR WorkQueue // Setup of eax with WorkQueue
                        // LinearAddress,
                        // ECX, EDX = 0
      IF (WorkQueue = 0) THEN {
        MWAIT
      }
    }
  }
}
```



```

}
// C1 handler uses a Halt instruction.
VOID C1Handler()
{ STI
  HLT
}

```

9.10.6.3 Halt Idle Logical Processors

If one of two logical processors is idle or in a spin-wait loop of long duration, explicitly halt that processor by means of a HLT instruction.

In an MP system, operating systems can place idle processors into a loop that continuously checks the run queue for runnable software tasks. Logical processors that execute idle loops consume a significant amount of core's execution resources that might otherwise be used by the other logical processors in the physical package. For this reason, halting idle logical processors optimizes the performance.¹ If all logical processors within a physical package are halted, the processor will enter a power-saving state.

9.10.6.4 Potential Usage of MONITOR/MWAIT in C1 Idle Loops

An operating system may also consider replacing HLT with MONITOR/MWAIT in its C1 idle loop. An example is shown in Example 9-27:

Example 9-27. An OS Idle Loop with MONITOR/MWAIT in the C1 Idle Loop

```

// WorkQueue is a memory location indicating there is a thread
// ready to run. A non-zero value for WorkQueue is assumed to
// indicate the presence of work to be scheduled on the processor.
// The following example assumes that the necessary padding has been
// added surrounding WorkQueue to eliminate false wakeups
// The idle loop is entered with interrupts disabled.

```

```

WHILE (1) {
  IF (WorkQueue) THEN {
    // Schedule work at WorkQueue
  }
  ELSE {
    // No work to do - wait in appropriate C-state handler depending
    // on Idle time accumulated
    IF (IdleTime >= IdleTimeThreshold) THEN {
      // Call appropriate C1, C2, C3 state handler, C1
      // handler shown below
    }
  }
}

```

```

VOID C1Handler()

{ MONITOR WorkQueue // Setup of eax with WorkQueue LinearAddress,
  // ECX, EDX = 0
  IF (WorkQueue = 0) THEN {
    STI
  }
}

```

1. Excessive transitions into and out of the HALT state could also incur performance penalties. Operating systems should evaluate the performance trade-offs for their operating system.

```

    MWAIT    // EAX, ECX = 0
  }
}

```

9.10.6.5 Guidelines for Scheduling Threads on Logical Processors Sharing Execution Resources

Because the logical processors, the order in which threads are dispatched to logical processors for execution can affect the overall efficiency of a system. The following guidelines are recommended for scheduling threads for execution.

- Dispatch threads to one logical processor per processor core before dispatching threads to the other logical processor sharing execution resources in the same processor core.
- In an MP system with two or more physical packages, distribute threads out over all the physical processors, rather than concentrate them in one or two physical processors.
- Use processor affinity to assign a thread to a specific processor core or package, depending on the cache-sharing topology. The practice increases the chance that the processor's caches will contain some of the thread's code and data when it is dispatched for execution after being suspended.

9.10.6.6 Eliminate Execution-Based Timing Loops

Intel discourages the use of timing loops that depend on a processor's execution speed to measure time. There are several reasons:

- Timing loops cause problems when they are calibrated on a IA-32 processor running at one frequency and then executed on a processor running at another frequency.
- Routines for calibrating execution-based timing loops produce unpredictable results when run on an IA-32 processor supporting Intel Hyper-Threading Technology. This is due to the sharing of execution resources between the logical processors within a physical package.

To avoid the problems described, timing loop routines must use a timing mechanism for the loop that does not depend on the execution speed of the logical processors in the system. The following sources are generally available:

- A high resolution system timer (for example, an Intel 8254).
- A high resolution timer within the processor (such as, the local APIC timer or the time-stamp counter).

For additional information, see the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

9.10.6.7 Place Locks and Semaphores in Aligned, 128-Byte Blocks of Memory

When software uses locks or semaphores to synchronize processes, threads, or other code sections; Intel recommends that only one lock or semaphore be present within a cache line (or 128 byte sector, if 128-byte sector is supported). In processors based on Intel NetBurst microarchitecture (which support 128-byte sector consisting of two cache lines), following this recommendation means that each lock or semaphore should be contained in a 128-byte block of memory that begins on a 128-byte boundary. The practice minimizes the bus traffic required to service locks.

9.11 MP INITIALIZATION FOR P6 FAMILY PROCESSORS

This section describes the MP initialization process for systems that use multiple P6 family processors. This process uses the MP initialization protocol that was introduced with the Pentium Pro processor (see Section 9.4, "Multiple-Processor (MP) Initialization"). For P6 family processors, this protocol is typically used to boot 2 or 4 processors that reside on single system bus; however, it can support from 2 to 15 processors in a multi-clustered system when the APIC buses are tied together. Larger systems are not supported.

9.11.1 Overview of the MP Initialization Process for P6 Family Processors

During the execution of the MP initialization protocol, one processor is selected as the bootstrap processor (BSP) and the remaining processors are designated as application processors (APs), see Section 9.4.1, “BSP and AP Processors.” Thereafter, the BSP manages the initialization of itself and the APs. This initialization includes executing BIOS initialization code and operating-system initialization code.

The MP protocol imposes the following requirements and restrictions on the system:

- An APIC clock (APICLK) must be provided.
- The MP protocol will be executed only after a power-up or RESET. If the MP protocol has been completed and a BSP has been chosen, subsequent INITs (either to a specific processor or system wide) do not cause the MP protocol to be repeated. Instead, each processor examines its BSP flag (in the APIC_BASE MSR) to determine whether it should execute the BIOS boot-strap code (if it is the BSP) or enter a wait-for-SIPI state (if it is an AP).
- All devices in the system that are capable of delivering interrupts to the processors must be inhibited from doing so for the duration of the MP initialization protocol. The time during which interrupts must be inhibited includes the window between when the BSP issues an INIT-SIPI-SIPI sequence to an AP and when the AP responds to the last SIPI in the sequence.

The following special-purpose interprocessor interrupts (IPIs) are used during the boot phase of the MP initialization protocol. These IPIs are broadcast on the APIC bus.

- Boot IPI (BIPI)—Initiates the arbitration mechanism that selects a BSP from the group of processors on the system bus and designates the remainder of the processors as APs. Each processor on the system bus broadcasts a BIPI to all the processors following a power-up or RESET.
- Final Boot IPI (FIPI)—Initiates the BIOS initialization procedure for the BSP. This IPI is broadcast to all the processors on the system bus, but only the BSP responds to it. The BSP responds by beginning execution of the BIOS initialization code at the reset vector.
- Startup IPI (SIPI)—Initiates the initialization procedure for an AP. The SIPI message contains a vector to the AP initialization code in the BIOS.

Table 9-5 describes the various fields of the boot phase IPIs.

Table 9-5. Boot Phase IPI Message Format

Type	Destination Field	Destination Shorthand	Trigger Mode	Level	Destination Mode	Delivery Mode	Vector (Hex)
BIPI	Not used	All including self	Edge	Deassert	Don't Care	Fixed (000)	40 to 4E*
FIPI	Not used	All including self	Edge	Deassert	Don't Care	Fixed (000)	10
SIPI	Used	All excluding self	Edge	Assert	Physical	StartUp (110)	00 to FF

NOTE:

* For all P6 family processors.

For BIPI messages, the lower 4 bits of the vector field contain the APIC ID of the processor issuing the message and the upper 4 bits contain the “generation ID” of the message. All P6 family processor will have a generation ID of 4H. BIPIs will therefore use vector values ranging from 40H to 4EH (4FH can not be used because FH is not a valid APIC ID).

9.11.2 MP Initialization Protocol Algorithm

Following a power-up or RESET of a system, the P6 family processors in the system execute the MP initialization protocol algorithm to initialize each of the processors on the system bus. In the course of executing this algorithm, the following boot-up and initialization operations are carried out:

MULTIPLE-PROCESSOR MANAGEMENT

1. Each processor on the system bus is assigned a unique APIC ID, based on system topology (see Section 9.4.5, "Identifying Logical Processors in an MP System"). This ID is written into the local APIC ID register for each processor.
2. Each processor executes its internal BIST simultaneously with the other processors on the system bus. Upon completion of the BIST (at T0), each processor broadcasts a BIPI to "all including self" (see Figure 9-1).
3. APIC arbitration hardware causes all the APICs to respond to the BIPIs one at a time (at T1, T2, T3, and T4).
4. When the first BIPI is received (at time T1), each APIC compares the four least significant bits of the BIPI's vector field with its APIC ID. If the vector and APIC ID match, the processor selects itself as the BSP by setting the BSP flag in its IA32_APIC_BASE MSR. If the vector and APIC ID do not match, the processor selects itself as an AP by entering the "wait for SIPI" state. (Note that in Figure 9-1, the BIPI from processor 1 is the first BIPI to be handled, so processor 1 becomes the BSP.)
5. The newly established BSP broadcasts an FIPI message to "all including self." The FIPI is guaranteed to be handled only after the completion of the BIPIs that were issued by the non-BSP processors.

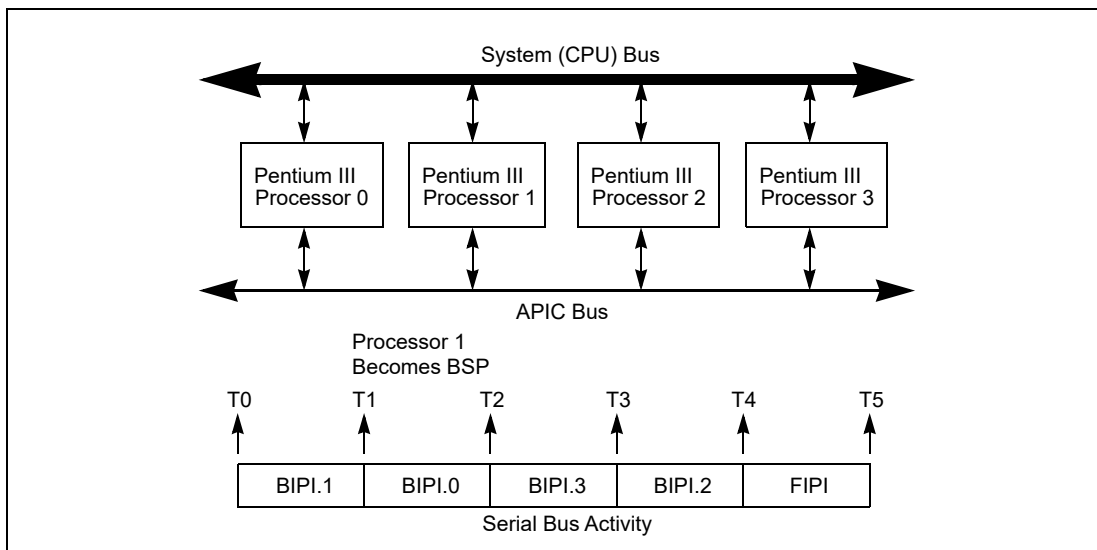


Figure 9-1. MP System With Multiple Pentium III Processors

6. After the BSP has been established, the outstanding BIPIs are received one at a time (at T2, T3, and T4) and ignored by all processors.
7. When the FIPI is finally received (at T5), only the BSP responds to it. It responds by fetching and executing BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).
8. As part of the boot-strap code, the BSP creates an ACPI table and an MP table and adds its initial APIC ID to these tables as appropriate.
9. At the end of the boot-strap procedure, the BSP broadcasts a SIPI message to all the APs in the system. Here, the SIPI message contains a vector to the BIOS AP initialization code (at 000V V000H, where VV is the vector contained in the SIPI message).
10. All APs respond to the SIPI message by racing to a BIOS initialization semaphore. The first one to the semaphore begins executing the initialization code. (See MP init code for semaphore implementation details.) As part of the AP initialization procedure, the AP adds its APIC ID number to the ACPI and MP tables as appropriate. At the completion of the initialization procedure, the AP executes a CLI instruction (to clear the IF flag in the EFLAGS register) and halts itself.
11. When each of the APs has gained access to the semaphore and executed the AP initialization code and all written their APIC IDs into the appropriate places in the ACPI and MP tables, the BSP establishes a count for the number of processors connected to the system bus, completes executing the BIOS boot-strap code, and then begins executing operating-system boot-strap and start-up code.

12. While the BSP is executing operating-system boot-strap and start-up code, the APs remain in the halted state. In this state they will respond only to INITs, NMIs, and SMIs. They will also respond to snoops and to assertions of the STPCLK# pin.

See Section 9.4.4, “MP Initialization Example,” for an annotated example the use of the MP protocol to boot IA-32 processors in an MP. This code should run on any IA-32 processor that used the MP protocol.

9.11.2.1 Error Detection and Handling During the MP Initialization Protocol

Errors may occur on the APIC bus during the MP initialization phase. These errors may be transient or permanent and can be caused by a variety of failure mechanisms (for example, broken traces, soft errors during bus usage, etc.). All serial bus related errors will result in an APIC checksum or acceptance error.

The MP initialization protocol makes the following assumptions regarding errors that occur during initialization:

- If errors are detected on the APIC bus during execution of the MP initialization protocol, the processors that detect the errors are shut down.
- The MP initialization protocol will be executed by processors even if they fail their BIST sequences.

7. Updates to Chapter 10, Volume 3A

Change bars and green text show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Update to the IA32_PERF_GLOBAL_CTRL entry and footnote in Table 10-1, "IA-32 and Intel® 64 Processor States Following Power-up, Reset, or INIT."

This chapter describes the facilities provided for managing processor wide functions and for initializing the processor. The subjects covered include: processor initialization, x87 FPU initialization, processor configuration, feature determination, mode switching, the MSRs (in the Pentium, P6 family, Pentium 4, and Intel Xeon processors), and the MTRRs (in the P6 family, Pentium 4, and Intel Xeon processors).

10.1 INITIALIZATION OVERVIEW

Following power-up or an assertion of the RESET# pin, each processor on the system bus performs a hardware initialization of the processor (known as a hardware reset) and an optional built-in self-test (BIST). A hardware reset sets each processor's registers to a known state and places the processor in real-address mode. It also invalidates the internal caches, translation lookaside buffers (TLBs) and the branch target buffer (BTB). At this point, the action taken depends on the processor family:

- **Pentium 4 processors (CPUID DisplayFamily 0FH)** — All the processors on the system bus (including a single processor in a uniprocessor system) execute the multiple processor (MP) initialization protocol. The processor that is selected through this protocol as the bootstrap processor (BSP) then immediately starts executing software-initialization code in the current code segment beginning at the offset in the EIP register. The application (non-BSP) processors (APs) go into a Wait For Startup IPI (SIPI) state while the BSP is executing initialization code. See Section 9.4, "Multiple-Processor (MP) Initialization," for more details. Note that in a uniprocessor system, the single Pentium 4 or Intel Xeon processor automatically becomes the BSP.
- **IA-32 and Intel 64 processors (CPUID DisplayFamily 06H)** — The action taken is the same as for the Pentium 4 processors (as described in the previous paragraph).
- **Pentium processors** — In either a single- or dual- processor system, a single Pentium processor is always pre-designated as the primary processor. Following a reset, the primary processor behaves as follows in both single- and dual-processor systems. Using the dual-processor (DP) ready initialization protocol, the primary processor immediately starts executing software-initialization code in the current code segment beginning at the offset in the EIP register. The secondary processor (if there is one) goes into a halt state.
- **Intel486 processor** — The primary processor (or single processor in a uniprocessor system) immediately starts executing software-initialization code in the current code segment beginning at the offset in the EIP register. (The Intel486 does not automatically execute a DP or MP initialization protocol to determine which processor is the primary processor.)

The software-initialization code performs all system-specific initialization of the BSP or primary processor and the system logic.

At this point, for MP (or DP) systems, the BSP (or primary) processor wakes up each AP (or secondary) processor to enable those processors to execute self-configuration code.

When all processors are initialized, configured, and synchronized, the BSP or primary processor begins executing an initial operating-system or executive task.

The x87 FPU is also initialized to a known state during hardware reset. x87 FPU software initialization code can then be executed to perform operations such as setting the precision of the x87 FPU and the exception masks. No special initialization of the x87 FPU is required to switch operating modes.

Asserting the INIT# pin on the processor invokes a similar response to a hardware reset. The major difference is that during an INIT, the internal caches, MSRs, MTRRs, and x87 FPU state are left unchanged (although, the TLBs and BTB are invalidated as with a hardware reset). An INIT provides a method for switching from protected to real-address mode while maintaining the contents of the internal caches.

10.1.1 Processor State After Reset

Following power-up, The state of control register CR0 is 60000010H (see Figure 10-1). This places the processor in real-address mode with paging disabled.

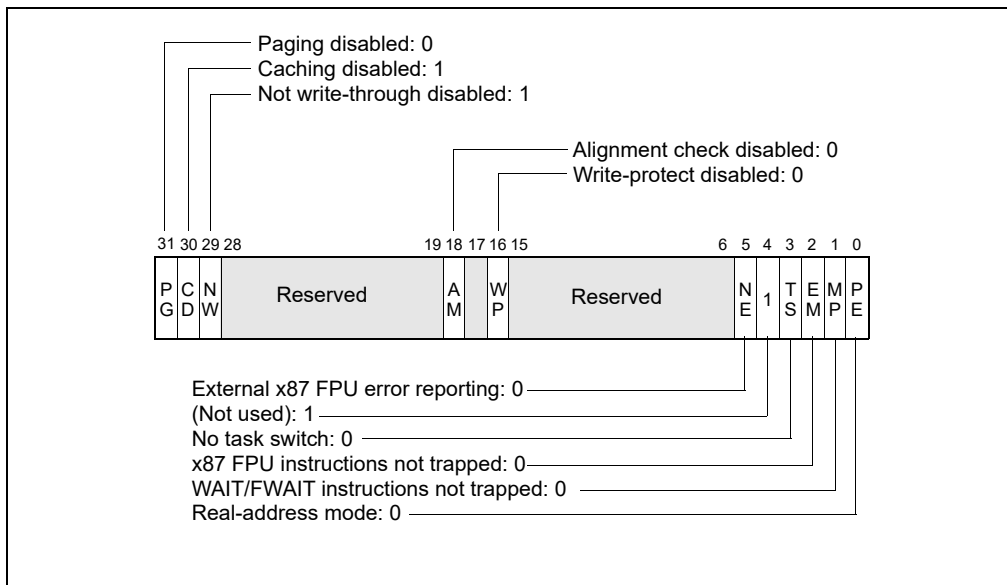


Figure 10-1. Contents of CR0 Register after Reset

The state of the flags and other registers following power-up for the Pentium 4, Pentium Pro, and Pentium processors are shown in Section 23.39, "Initial State of Pentium, Pentium Pro and Pentium 4 Processors," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

Table 10-1 shows processor states of IA-32 and Intel 64 processors with CPUID DisplayFamily signature of 06H at the following events: power-up, RESET, and INIT. In a few cases, the behavior of some registers behave slightly different across warm RESET, the variant cases are marked in Table 10-1 and described in more detail in Table 10-2.

Table 10-1. IA-32 and Intel® 64 Processor States Following Power-up, Reset, or INIT

Register	Power up	Reset	INIT
EFLAGS ¹	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H ²	60000010H ²	60000010H ²
CR2, CR3, CR4	00000000H	00000000H	00000000H
CS	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed
SS, DS, ES, FS, GS	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed
EDX	000n06xxH ³	000n06xxH ³	000n06xxH ³
EAX	0 ⁴	0 ⁴	0 ⁴
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H

Table 10-1. IA-32 and Intel® 64 Processor States Following Power-up, Reset, or INIT (Contd.)

Register	Power up	Reset	INIT
ST0 through ST7 ⁵	+0.0	+0.0	FINIT/FNINIT: Unchanged
x87 FPU Control Word ⁵	0040H	0040H	FINIT/FNINIT: 037FH
x87 FPU Status Word ⁵	0000H	0000H	FINIT/FNINIT: 0000H
x87 FPU Tag Word ⁵	5555H	5555H	FINIT/FNINIT: FFFFH
x87 FPU Data Operand and CS Seg. Selectors ⁵	0000H	0000H	FINIT/FNINIT: 0000H
x87 FPU Data Operand and Inst. Pointers ⁵	00000000H	00000000H	FINIT/FNINIT: 00000000H
MM0 through MM7 ⁵	0000000000000000H	0000000000000000H	INIT or FINIT/FNINIT: Unchanged
XMM0 through XMM7	0H	0H	Unchanged
MXCSR	1F80H	1F80H	Unchanged
GDTR, IDTR	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W
LDTR, Task Register	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W
DR0, DR1, DR2, DR3	00000000H	00000000H	00000000H
DR6	FFFF0FF0H	FFFF0FF0H	FFFF0FF0H
DR7	00000400H	00000400H	00000400H
R8-R15	0000000000000000H	0000000000000000H	0000000000000000H
XMM8-XMM15	0H	0H	Unchanged
XCRO	1H	1H	Unchanged
IA32_XSS	0H	0H	Unchanged
YMM_H[255:128]	0H	0H	Unchanged
BNDCFGU	0H	0H	0H
BND0-BND3	0H	0H	0H
IA32_BNDCFGS	0H	0H	0H
OPMASK	0H	0H	Unchanged
ZMM_H[511:256]	0H	0H	Unchanged
ZMMHi16[511:0]	0H	0H	Unchanged
PKRU	0H	0H	Unchanged
Intel Processor Trace MSRs	0H	0H ^w	Unchanged
Time-Stamp Counter	0H	0H ^w	Unchanged
IA32_TSC_AUX	0H	0H	Unchanged
IA32_TSC_ADJUST	0H	0H	Unchanged
IA32_TSC_DEADLINE	0H	0H	Unchanged
IA32_SYSENTER_CS/ESP/EIP	0H	0H	Unchanged
IA32_EFER	0000000000000000H	0000000000000000H	0000000000000000H
IA32_STAR/LSTAR	0H	0H	Unchanged

Table 10-1. IA-32 and Intel® 64 Processor States Following Power-up, Reset, or INIT (Contd.)

Register	Power up	Reset	INIT
IA32_FS_BASE/GS_BASE	0H	0H	0H
IA32_PMCx, IA32_PERFEVTSELx	0H	0H	Unchanged
IA32_PERF_GLOBAL_CTRL	Sets bits n-1:0 and clears the upper bits. ⁷	Sets bits n-1:0 and clears the upper bits. ⁷	Unchanged
IA32_FIXED_CTRx, IA32_FIXED_CTR_CTRL	0H	0H	Unchanged
Data and Code Cache, TLBs	Invalid ⁶	Invalid ⁶	Unchanged
Fixed MTRRs	Disabled	Disabled	Unchanged
Variable MTRRs	Disabled	Disabled	Unchanged
Machine-Check Banks	Undefined	Undefined ^w	Unchanged
Last Branch Record Stack	0	0 ^w	Unchanged
APIC	Enabled	Enabled	Unchanged
X2APIC	Disabled	Disabled	Unchanged
IA32_DEBUG_INTERFACE	0	0 ^w	Unchanged

NOTES:

1. The 10 most-significant bits of the EFLAGS register are undefined following a reset. Software should not depend on the states of any of these bits.
2. The CD and NW flags are unchanged, bit 4 is set to 1, all other bits are cleared.
3. Where “n” is the Extended Model Value for the respective processor, and “xx” = don’t care.
4. If Built-In Self-Test (BIST) is invoked on power up or reset, EAX is 0 only if all tests passed. (BIST cannot be invoked during an INIT.)
5. The state of the x87 FPU and MMX registers is not changed by the execution of an INIT.
6. Internal caches are invalid after power-up and RESET, but left unchanged with an INIT.
7. Where “n” is the number of general-purpose counters available in the processor. See Chapter 20, “Performance Monitoring,” for additional details.

w: Warm RESET behavior differs from power-on RESET with details listed in Table 10-2.

Table 10-2. Variance of RESET Values in Selected Intel Architecture Processors

State	XREF	Value	Feature Flag or DisplayFamily_DisplayModel Signatures
Time-Stamp Counter	Warm RESET	Unmodified across warm Reset	06_2DH, 06_3EH
Machine-Check Banks	Warm RESET	IA32_MCi_Status banks are unmodified across warm Reset	06_2DH, 06_3EH, 06_3FH, 06_4FH, 06_56H
Last Branch Record Stack	Warm RESET	LBR stack MSRs are unmodified across warm Reset	06_1AH, 06_1CH, DisplayFamiy= 06 and DisplayModel >1DH
Intel Processor Trace MSRs	Warm RESET	Clears IA32_RTIT_CTL.TraceEn, the rest of MSRs are unmodified	If CPUID.(EAX=14H, ECX=0H):EBX[bit 2] = 1
IA32_DEBUG_INTERFACE	Warm RESET	Unmodified across warm Reset	If CPUID.01H:ECX.[11] = 1

10.1.2 Processor Built-In Self-Test (BIST)

Hardware may request that the BIST be performed at power-up. The EAX register is cleared (0H) if the processor passes the BIST. A nonzero value in the EAX register after the BIST indicates that a processor fault was detected. If the BIST is not requested, the contents of the EAX register after a hardware reset is 0H.

The overhead for performing a BIST varies between processor families. For example, the BIST takes approximately 30 million processor clock periods to execute on the Pentium 4 processor. This clock count is model-specific; Intel reserves the right to change the number of periods for any Intel 64 or IA-32 processor, without notification.

10.1.3 Model and Stepping Information

Following a hardware reset, the EDX register contains component identification and revision information (see Figure 10-2). For example, the model, family, and processor type returned for the first processor in the Intel Pentium 4 family is as follows: model (0000B), family (1111B), and processor type (00B).

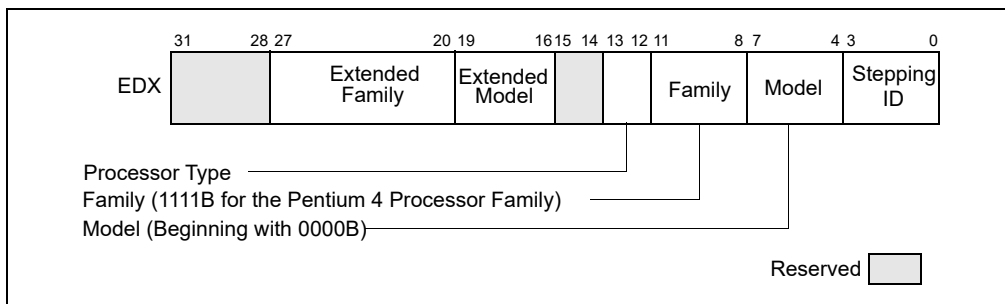


Figure 10-2. Version Information in the EDX Register after Reset

The stepping ID field contains a unique identifier for the processor's stepping ID or revision level. The extended family and extended model fields were added to the IA-32 architecture in the Pentium 4 processors.

10.1.4 First Instruction Executed

The first instruction that is fetched and executed following a hardware reset is located at physical address FFFFFFF0H. This address is 16 bytes below the processor's uppermost physical address. The EPROM containing the software-initialization code must be located at this address.

The address FFFFFFF0H is beyond the 1-MByte addressable range of the processor while in real-address mode. The processor is initialized to this starting address as follows. The CS register has two parts: the visible segment selector part and the hidden base address part. In real-address mode, the base address is normally formed by shifting the 16-bit segment selector value 4 bits to the left to produce a 20-bit base address. However, during a hardware reset, the segment selector in the CS register is loaded with F000H and the base address is loaded with FFFF0000H. The starting address is thus formed by adding the base address to the value in the EIP register (that is, FFFF0000 + FFF0H = FFFFFFF0H).

The first time the CS register is loaded with a new value after a hardware reset, the processor will follow the normal rule for address translation in real-address mode (that is, [CS base address = CS segment selector * 16]). To ensure that the base address in the CS register remains unchanged until the EPROM based software-initialization code is completed, the code must not contain a far jump or far call or allow an interrupt to occur (which would cause the CS selector value to be changed).

10.2 X87 FPU INITIALIZATION

Software-initialization code can determine whether the processor contains an x87 FPU by using the CPUID instruction. The code must then initialize the x87 FPU and set flags in control register CR0 to reflect the state of the x87 FPU environment.

A hardware reset places the x87 FPU in the state shown in Table 10-1. This state is different from the state the x87 FPU is placed in following the execution of a FINIT or FNINIT instruction (also shown in Table 10-1). If the x87 FPU is to be used, the software-initialization code should execute a FINIT/FNINIT instruction following a hardware reset. These instructions, tag all data registers as empty, clear all the exception masks, set the TOP-of-stack value to 0, and select the default rounding and precision controls setting (round to nearest and 64-bit precision).

If the processor is reset by asserting the INIT# pin, the x87 FPU state is not changed.

10.2.1 Configuring the x87 FPU Environment

Initialization code must load the appropriate values into the MP, EM, and NE flags of control register CR0. These bits are cleared on hardware reset of the processor. Figure 10-3 shows the suggested settings for these flags, depending on the IA-32 processor being initialized. Initialization code can test for the type of processor present before setting or clearing these flags.

Table 10-3. Recommended Settings of EM and MP Flags on IA-32 Processors

EM	MP	NE	IA-32 processor
1	0	1	Intel486™ SX, Intel386™ DX, and Intel386™ SX processors only, without the presence of a math coprocessor.
0	1	1 or 0*	Pentium 4, Intel Xeon, P6 family, Pentium, Intel486™ DX, and Intel 487 SX processors, and Intel386 DX and Intel386 SX processors when a companion math coprocessor is present.
0	1	1 or 0*	More recent Intel 64 or IA-32 processors

NOTE:

* The setting of the NE flag depends on the operating system being used.

The EM flag determines whether floating-point instructions are executed by the x87 FPU (EM is cleared) or a device-not-available exception (#NM) is generated for all floating-point instructions so that an exception handler can emulate the floating-point operation (EM = 1). Ordinarily, the EM flag is cleared when an x87 FPU or math coprocessor is present and set if they are not present. If the EM flag is set and no x87 FPU, math coprocessor, or floating-point emulator is present, the processor will hang when a floating-point instruction is executed.

The MP flag determines whether WAIT/FWAIT instructions react to the setting of the TS flag. If the MP flag is clear, WAIT/FWAIT instructions ignore the setting of the TS flag; if the MP flag is set, they will generate a device-not-available exception (#NM) if the TS flag is set. Generally, the MP flag should be set for processors with an integrated x87 FPU and clear for processors without an integrated x87 FPU and without a math coprocessor present. However, an operating system can choose to save the floating-point context at every context switch, in which case there would be no need to set the MP bit.

Table 2-2 shows the actions taken for floating-point and WAIT/FWAIT instructions based on the settings of the EM, MP, and TS flags.

The NE flag determines whether unmasked floating-point exceptions are handled by generating a floating-point error exception internally (NE is set, native mode) or through an external interrupt (NE is cleared). In systems where an external interrupt controller is used to invoke numeric exception handlers (such as MS-DOS-based systems), the NE bit should be cleared.

10.2.2 Setting the Processor for x87 FPU Software Emulation

Setting the EM flag causes the processor to generate a device-not-available exception (#NM) and trap to a software exception handler whenever it encounters a floating-point instruction. (Table 10-3 shows when it is appropriate to use this flag.) Setting this flag has two functions:

- It allows x87 FPU code to run on an IA-32 processor that has neither an integrated x87 FPU nor is connected to an external math coprocessor, by using a floating-point emulator.
- It allows floating-point code to be executed using a special or nonstandard floating-point emulator, selected for a particular application, regardless of whether an x87 FPU or math coprocessor is present.

To emulate floating-point instructions, the EM, MP, and NE flag in control register CR0 should be set as shown in Table 10-4.

Table 10-4. Software Emulation Settings of EM, MP, and NE Flags

CRO Bit	Value
EM	1
MP	0
NE	1

Regardless of the value of the EM bit, the Intel486 SX processor generates a device-not-available exception (#NM) upon encountering any floating-point instruction.

10.3 CACHE ENABLING

IA-32 processors (beginning with the Intel486 processor) and Intel 64 processors contain internal instruction and data caches. These caches are enabled by clearing the CD and NW flags in control register CR0. (They are set during a hardware reset.) Because all internal cache lines are invalid following reset initialization, it is not necessary to invalidate the cache before enabling caching. Any external caches may require initialization and invalidation using a system-specific initialization and invalidation code sequence.

Depending on the hardware and operating system or executive requirements, additional configuration of the processor's caching facilities will probably be required. Beginning with the Intel486 processor, page-level caching can be controlled with the PCD and PWT flags in page-directory and page-table entries. Beginning with the P6 family processors, the memory type range registers (MTRRs) control the caching characteristics of the regions of physical memory. (For the Intel486 and Pentium processors, external hardware can be used to control the caching characteristics of regions of physical memory.) See Chapter 12, "Memory Cache Control," for detailed information on configuration of the caching facilities in the Pentium 4, Intel Xeon, and P6 family processors and system memory.

10.4 MODEL-SPECIFIC REGISTERS (MSRS)

Most IA-32 processors (starting from Pentium processors) and Intel 64 processors contain a model-specific registers (MSRs). A given MSR may not be supported across all families and models for Intel 64 and IA-32 processors. Some MSRs are designated as architectural to simplify software programming; a feature introduced by an architectural MSR is expected to be supported in future processors. Non-architectural MSRs are not guaranteed to be supported or to have the same functions on future processors.

MSRs that provide control for a number of hardware and software-related features, include:

- Performance-monitoring counters (see Chapter 20, "Performance Monitoring").
- Debug extensions (see Chapter 18, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features").
- Machine-check exception capability and its accompanying machine-check architecture (see Chapter 16, "Machine-Check Architecture").
- MTRRs (see Section 12.11, "Memory Type Range Registers (MTRRs)").
- Thermal and power management.
- Instruction-specific support (for example: SYSENTER, SYSEXIT, SWAPGS, etc.).
- Processor feature/mode support (for example: IA32_EFER, IA32_FEATURE_CONTROL).

The MSRs can be read and written to using the RDMSR and WRMSR instructions, respectively.

When performing software initialization of an IA-32 or Intel 64 processor, many of the MSRs will need to be initialized to set up things like performance-monitoring events, run-time machine checks, and memory types for physical memory.

Lists of available performance-monitoring events can be found at: <https://perfmon-events.intel.com/>, and lists of available MSRs are given in Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4. The references earlier in this section show where the functions of the various groups of MSRs are described in this manual.

10.5 MEMORY TYPE RANGE REGISTERS (MTRRS)

Memory type range registers (MTRRs) were introduced into the IA-32 architecture with the Pentium Pro processor. They allow the type of caching (or no caching) to be specified in system memory for selected physical address ranges. They allow memory accesses to be optimized for various types of memory such as RAM, ROM, frame buffer memory, and memory-mapped I/O devices.

In general, initializing the MTRRs is normally handled by the software initialization code or BIOS and is not an operating system or executive function. At the very least, all the MTRRs must be cleared to 0, which selects the uncached (UC) memory type. See Section 12.11, “Memory Type Range Registers (MTRRs),” for detailed information on the MTRRs.

10.6 INITIALIZING SSE/SSE2/SSE3/SSSE3 EXTENSIONS

For processors that contain SSE/SSE2/SSE3/SSSE3 extensions, steps must be taken when initializing the processor to allow execution of these instructions.

1. Check the CPUID feature flags for the presence of the SSE/SSE2/SSE3/SSSE3 extensions (respectively: EDX bits 25 and 26, ECX bit 0 and 9) and support for the FXSAVE and FXRSTOR instructions (EDX bit 24). Also check for support for the CLFLUSH instruction (EDX bit 19). The CPUID feature flags are loaded in the EDX and ECX registers when the CPUID instruction is executed with a 1 in the EAX register.
2. Set the OSFXSR flag (bit 9 in control register CR4) to indicate that the operating system supports saving and restoring the SSE/SSE2/SSE3/SSSE3 execution environment (XMM and MXCSR registers) with the FXSAVE and FXRSTOR instructions, respectively. See Section 2.5, “Control Registers,” for a description of the OSFXSR flag.
3. Set the OSXMMEXCPT flag (bit 10 in control register CR4) to indicate that the operating system supports the handling of SSE/SSE2/SSE3 SIMD floating-point exceptions (#XM). See Section 2.5, “Control Registers,” for a description of the OSXMMEXCPT flag.
4. Set the mask bits and flags in the MXCSR register according to the mode of operation desired for SSE/SSE2/SSE3 SIMD floating-point instructions. See “MXCSR Control and Status Register” in Chapter 10, “Programming with Intel® Streaming SIMD Extensions (Intel® SSE),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for a detailed description of the bits and flags in the MXCSR register.

10.7 SOFTWARE INITIALIZATION FOR REAL-ADDRESS MODE OPERATION

Following a hardware reset (either through a power-up or the assertion of the RESET# pin) the processor is placed in real-address mode and begins executing software initialization code from physical address FFFFFFF0H. Software initialization code must first set up the necessary data structures for handling basic system functions, such as a real-mode IDT for handling interrupts and exceptions. If the processor is to remain in real-address mode, software must then load additional operating-system or executive code modules and data structures to allow reliable execution of application programs in real-address mode.

If the processor is going to operate in protected mode, software must load the necessary data structures to operate in protected mode and then switch to protected mode. The protected-mode data structures that must be loaded are described in Section 10.8, “Software Initialization for Protected-Mode Operation.”

10.7.1 Real-Address Mode IDT

In real-address mode, the only system data structure that must be loaded into memory is the IDT (also called the “interrupt vector table”). By default, the address of the base of the IDT is physical address 0H. This address can be

changed by using the LIDT instruction to change the base address value in the IDTR. Software initialization code needs to load interrupt- and exception-handler pointers into the IDT before interrupts can be enabled.

The actual interrupt- and exception-handler code can be contained either in EPROM or RAM; however, the code must be located within the 1-MByte addressable range of the processor in real-address mode. If the handler code is to be stored in RAM, it must be loaded along with the IDT.

10.7.2 NMI Interrupt Handling

The NMI interrupt is always enabled (except when multiple NMIs are nested). If the IDT and the NMI interrupt handler need to be loaded into RAM, there will be a period of time following hardware reset when an NMI interrupt cannot be handled. During this time, hardware must provide a mechanism to prevent an NMI interrupt from halting code execution until the IDT and the necessary NMI handler software is loaded. Here are two examples of how NMIs can be handled during the initial states of processor initialization:

- A simple IDT and NMI interrupt handler can be provided in EPROM. This allows an NMI interrupt to be handled immediately after reset initialization.
- The system hardware can provide a mechanism to enable and disable NMIs by passing the NMI# signal through an AND gate controlled by a flag in an I/O port. Hardware can clear the flag when the processor is reset, and software can set the flag when it is ready to handle NMI interrupts.

10.8 SOFTWARE INITIALIZATION FOR PROTECTED-MODE OPERATION

The processor is placed in real-address mode following a hardware reset. At this point in the initialization process, some basic data structures and code modules must be loaded into physical memory to support further initialization of the processor, as described in Section 10.7, "Software Initialization for Real-Address Mode Operation." Before the processor can be switched to protected mode, the software initialization code must load a minimum number of protected mode data structures and code modules into memory to support reliable operation of the processor in protected mode. These data structures include the following:

- A IDT.
- A GDT.
- A TSS.
- (Optional) An LDT.
- If paging is to be used, at least one page directory and one page table.
- A code segment that contains the code to be executed when the processor switches to protected mode.
- One or more code modules that contain the necessary interrupt and exception handlers.

Software initialization code must also initialize the following system registers before the processor can be switched to protected mode:

- The GDTR.
- (Optional.) The IDTR. This register can also be initialized immediately after switching to protected mode, prior to enabling interrupts.
- Control registers CR1 through CR4.
- (Pentium 4, Intel Xeon, and P6 family processors only.) The memory type range registers (MTRRs).

With these data structures, code modules, and system registers initialized, the processor can be switched to protected mode by loading control register CR0 with a value that sets the PE flag (bit 0).

10.8.1 Protected-Mode System Data Structures

The contents of the protected-mode system data structures loaded into memory during software initialization, depend largely on the type of memory management the protected-mode operating-system or executive is going to support: flat, flat with paging, segmented, or segmented with paging.

To implement a flat memory model without paging, software initialization code must at a minimum load a GDT with one code and one data-segment descriptor. A null descriptor in the first GDT entry is also required. The stack can be placed in a normal read/write data segment, so no dedicated descriptor for the stack is required. A flat memory model with paging also requires a page directory and at least one page table (unless all pages are 4 MBytes in which case only a page directory is required). See Section 10.8.3, "Initializing Paging."

Before the GDT can be used, the base address and limit for the GDT must be loaded into the GDTR register using an LGDT instruction.

A multi-segmented model may require additional segments for the operating system, as well as segments and LDTs for each application program. LDTs require segment descriptors in the GDT. Some operating systems allocate new segments and LDTs as they are needed. This provides maximum flexibility for handling a dynamic programming environment. However, many operating systems use a single LDT for all tasks, allocating GDT entries in advance. An embedded system, such as a process controller, might pre-allocate a fixed number of segments and LDTs for a fixed number of application programs. This would be a simple and efficient way to structure the software environment of a real-time system.

10.8.2 Initializing Protected-Mode Exceptions and Interrupts

Software initialization code must at a minimum load a protected-mode IDT with gate descriptor for each exception vector that the processor can generate. If interrupt or trap gates are used, the gate descriptors can all point to the same code segment, which contains the necessary exception handlers. If task gates are used, one TSS and accompanying code, data, and task segments are required for each exception handler called with a task gate.

If hardware allows interrupts to be generated, gate descriptors must be provided in the IDT for one or more interrupt handlers.

Before the IDT can be used, the base address and limit for the IDT must be loaded into the IDTR register using an LIDT instruction. This operation is typically carried out immediately after switching to protected mode.

10.8.3 Initializing Paging

Paging is controlled by the PG flag in control register CR0. When this flag is clear (its state following a hardware reset), the paging mechanism is turned off; when it is set, paging is enabled. Before setting the PG flag, the following data structures and registers must be initialized:

- Software must load at least one page directory and one page table into physical memory. The page table can be eliminated if the page directory contains a directory entry pointing to itself (here, the page directory and page table reside in the same page), or if only 4-MByte pages are used.
- Control register CR3 (also called the PDBR register) is loaded with the physical base address of the page directory.
- (Optional) Software may provide one set of code and data descriptors in the GDT or in an LDT for supervisor mode and another set for user mode.

With this paging initialization complete, paging is enabled and the processor is switched to protected mode at the same time by loading control register CR0 with an image in which the PG and PE flags are set. (Paging cannot be enabled before the processor is switched to protected mode.)

10.8.4 Initializing Multitasking

If the multitasking mechanism is not going to be used and changes between privilege levels are not allowed, it is not necessary to load a TSS into memory or to initialize the task register.

If the multitasking mechanism is going to be used and/or changes between privilege levels are allowed, software initialization code must load at least one TSS and an accompanying TSS descriptor. (A TSS is required to change privilege levels because pointers to the privileged-level 0, 1, and 2 stack segments and the stack pointers for these stacks are obtained from the TSS.) TSS descriptors must not be marked as busy when they are created; they should be marked busy by the processor only as a side-effect of performing a task switch. As with descriptors for LDTs, TSS descriptors reside in the GDT.

After the processor has switched to protected mode, the LTR instruction can be used to load a segment selector for a TSS descriptor into the task register. This instruction marks the TSS descriptor as busy, but does not perform a task switch. The processor can, however, use the TSS to locate pointers to privilege-level 0, 1, and 2 stacks. The segment selector for the TSS must be loaded before software performs its first task switch in protected mode, because a task switch copies the current task state into the TSS.

After the LTR instruction has been executed, further operations on the task register are performed by task switching. As with other segments and LDTs, TSSs and TSS descriptors can be either pre-allocated or allocated as needed.

10.8.5 Initializing IA-32e Mode

On Intel 64 processors, the IA32_EFER MSR is cleared on system reset. The operating system must be in protected mode with paging enabled before attempting to initialize IA-32e mode. IA-32e mode operation also requires physical-address extensions with four or five levels of enhanced paging structures (see Section 4.5, “4-Level Paging and 5-Level Paging”).

Operating systems should follow this sequence to initialize IA-32e mode:

1. Starting from protected mode, disable paging by setting CR0.PG = 0. Use the MOV CR0 instruction to disable paging (the instruction must be located in an identity-mapped page).
2. Enable physical-address extensions (PAE) by setting CR4.PAE = 1. Failure to enable PAE will result in a #GP fault when an attempt is made to initialize IA-32e mode.
3. Load CR3 with the physical base address of the Level 4 page map table (PML4) or Level 5 page map table (PML5).
4. Enable IA-32e mode by setting IA32_EFER.LME = 1.
5. Enable paging by setting CR0.PG = 1. This causes the processor to set the IA32_EFER.LMA bit to 1. The MOV CR0 instruction that enables paging and the following instructions must be located in an identity-mapped page (until such time that a branch to non-identity mapped pages can be effected).

64-bit mode paging structures must be located in the first 4 GBytes of physical-address space prior to activating IA-32e mode. This is necessary because the MOV CR3 instruction used to initialize the page-directory base must be executed in legacy mode prior to activating IA-32e mode (setting CR0.PG = 1 to enable paging). Because MOV CR3 is executed in protected mode, only the lower 32 bits of the register are written, limiting the table location to the low 4 GBytes of memory. Software can relocate the page tables anywhere in physical memory after IA-32e mode is activated.

The processor performs 64-bit mode consistency checks whenever software attempts to modify any of the enable bits directly involved in activating IA-32e mode (IA32_EFER.LME, CR0.PG, and CR4.PAE). It will generate a general protection fault (#GP) if consistency checks fail. 64-bit mode consistency checks ensure that the processor does not enter an undefined mode or state with unpredictable behavior.

64-bit mode consistency checks fail in the following circumstances:

- An attempt is made to enable or disable IA-32e mode while paging is enabled.
- IA-32e mode is enabled and an attempt is made to enable paging prior to enabling physical-address extensions (PAE).
- IA-32e mode is active and an attempt is made to disable physical-address extensions (PAE).
- If the current CS has the L-bit set on an attempt to activate IA-32e mode.
- If the TR contains a 16-bit TSS on an attempt to activate IA-32e mode.

10.8.5.1 IA-32e Mode System Data Structures

After activating IA-32e mode, the system-descriptor-table registers (GDTR, LDTR, IDTR, TR) continue to reference legacy protected-mode descriptor tables. Tables referenced by the descriptors all reside in the lower 4 GBytes of linear-address space. After activating IA-32e mode, 64-bit operating-systems should use the LGDT, LLDT, LIDT, and LTR instructions to load the system-descriptor-table registers with references to 64-bit descriptor tables.

10.8.5.2 IA-32e Mode Interrupts and Exceptions

Software must not allow exceptions or interrupts to occur between the time IA-32e mode is activated and the update of the interrupt-descriptor-table register (IDTR) that establishes references to a 64-bit interrupt-descriptor table (IDT). This is because the IDT remains in legacy form immediately after IA-32e mode is activated.

If an interrupt or exception occurs prior to updating the IDTR, a legacy 32-bit interrupt gate will be referenced and interpreted as a 64-bit interrupt gate with unpredictable results. External interrupts can be disabled by using the CLI instruction.

Non-maskable interrupts (NMI) must be disabled using external hardware.

10.8.5.3 64-bit Mode and Compatibility Mode Operation

IA-32e mode uses two code segment-descriptor bits (CS.L and CS.D, see Figure 3-8) to control the operating modes after IA-32e mode is initialized. If CS.L = 1 and CS.D = 0, the processor is running in 64-bit mode. With this encoding, the default operand size is 32 bits and default address size is 64 bits. Using instruction prefixes, operand size can be changed to 64 bits or 16 bits; address size can be changed to 32 bits.

When IA-32e mode is active and CS.L = 0, the processor operates in compatibility mode. In this mode, CS.D controls default operand and address sizes exactly as it does in the IA-32 architecture. Setting CS.D = 1 specifies default operand and address size as 32 bits. Clearing CS.D to 0 specifies default operand and address size as 16 bits (the CS.L = 1, CS.D = 1 bit combination is reserved).

Compatibility mode execution is selected on a code-segment basis. This mode allows legacy applications to coexist with 64-bit applications running in 64-bit mode. An operating system running in IA-32e mode can execute existing 16-bit and 32-bit applications by clearing their code-segment descriptor's CS.L bit to 0.

In compatibility mode, the following system-level mechanisms continue to operate using the IA-32e-mode architectural semantics:

- Linear-to-physical address translation uses the 64-bit mode extended page-translation mechanism.
- Interrupts and exceptions are handled using the 64-bit mode mechanisms.
- System calls (calls through call gates and SYSENTER/SYSEXIT) are handled using the IA-32e mode mechanisms.

10.8.5.4 Switching Out of IA-32e Mode Operation

To return from IA-32e mode to paged-protected mode operation operating systems must use the following sequence:

1. Switch to compatibility mode.
2. Deactivate IA-32e mode by clearing CR0.PG = 0. This causes the processor to set IA32_EFER.LMA = 0. The MOV CR0 instruction used to disable paging and subsequent instructions must be located in an identity-mapped page.
3. Load CR3 with the physical base address of the legacy page-table-directory base address.
4. Disable IA-32e mode by setting IA32_EFER.LME = 0.
5. Enable legacy paged-protected mode by setting CR0.PG = 1
6. A branch instruction must follow the MOV CR0 that enables paging. Both the MOV CR0 and the branch instruction must be located in an identity-mapped page.

Registers only available in 64-bit mode (R8-R15 and XMM8-XMM15) are preserved across transitions from 64-bit mode into compatibility mode then back into 64-bit mode. However, values of R8-R15 and XMM8-XMM15 are undefined after transitions from 64-bit mode through compatibility mode to legacy or real mode and then back through compatibility mode to 64-bit mode.

10.9 MODE SWITCHING

To use the processor in protected mode after hardware or software reset, a mode switch must be performed from real-address mode. Once in protected mode, software generally does not need to return to real-address mode. To run software written to run in real-address mode (8086 mode), it is generally more convenient to run the software in virtual-8086 mode, than to switch back to real-address mode.

10.9.1 Switching to Protected Mode

Before switching to protected mode from real mode, a minimum set of system data structures and code modules must be loaded into memory, as described in Section 10.8, “Software Initialization for Protected-Mode Operation.” Once these tables are created, software initialization code can switch into protected mode.

Protected mode is entered by executing a MOV CR0 instruction that sets the PE flag in the CR0 register. (In the same instruction, the PG flag in register CR0 can be set to enable paging.) Execution in protected mode begins with a CPL of 0.

Intel 64 and IA-32 processors have slightly different requirements for switching to protected mode. To ensure upwards and downwards code compatibility with Intel 64 and IA-32 processors, we recommend that you follow these steps:

1. Disable interrupts. A CLI instruction disables maskable hardware interrupts. NMI interrupts can be disabled with external circuitry. (Software must guarantee that no exceptions or interrupts are generated during the mode switching operation.)
2. Execute the LGDT instruction to load the GDTR register with the base address of the GDT.
3. Execute a MOV CR0 instruction that sets the PE flag (and optionally the PG flag) in control register CR0.
4. Immediately following the MOV CR0 instruction, execute a far JMP or far CALL instruction. (This operation is typically a far jump or call to the next instruction in the instruction stream.)
5. The JMP or CALL instruction immediately after the MOV CR0 instruction changes the flow of execution and serializes the processor.
6. If paging is enabled, the code for the MOV CR0 instruction and the JMP or CALL instruction must come from a page that is identity mapped (that is, the linear address before the jump is the same as the physical address after paging and protected mode is enabled). The target instruction for the JMP or CALL instruction does not need to be identity mapped.
7. If a local descriptor table is going to be used, execute the LLDT instruction to load the segment selector for the LDT in the LDTR register.
8. Execute the LTR instruction to load the task register with a segment selector to the initial protected-mode task or to a writable area of memory that can be used to store TSS information on a task switch.
9. After entering protected mode, the segment registers continue to hold the contents they had in real-address mode. The JMP or CALL instruction in step 4 resets the CS register. Perform one of the following operations to update the contents of the remaining segment registers.
 - Reload segment registers DS, SS, ES, FS, and GS. If the ES, FS, and/or GS registers are not going to be used, load them with a null selector.
 - Perform a JMP or CALL instruction to a new task, which automatically resets the values of the segment registers and branches to a new code segment.
10. Execute the LIDT instruction to load the IDTR register with the address and limit of the protected-mode IDT.
11. Execute the STI instruction to enable maskable hardware interrupts and perform the necessary hardware operation to enable NMI interrupts.

Random failures can occur if other instructions exist between steps 3 and 4 above. Failures will be readily seen in some situations, such as when instructions that reference memory are inserted between steps 3 and 4 while in system management mode.

10.9.2 Switching Back to Real-Address Mode

The processor switches from protected mode back to real-address mode if software clears the PE bit in the CR0 register with a MOV CR0 instruction. A procedure that re-enters real-address mode should perform the following steps:

1. Disable interrupts. A CLI instruction disables maskable hardware interrupts. NMI interrupts can be disabled with external circuitry.
2. If paging is enabled, perform the following operations:
 - Transfer program control to linear addresses that are identity mapped to physical addresses (that is, linear addresses equal physical addresses).
 - Ensure that the GDT and IDT are in identity mapped pages.
 - Clear the PG bit in the CR0 register.
 - Move 0H into the CR3 register to flush the TLB.
3. Transfer program control to a readable segment that has a limit of 64 KBytes (FFFFH). This operation loads the CS register with the segment limit required in real-address mode.
4. Load segment registers SS, DS, ES, FS, and GS with a selector for a descriptor containing the following values, which are appropriate for real-address mode:
 - Limit = 64 KBytes (0FFFFH)
 - Byte granular (G = 0)
 - Expand up (E = 0)
 - Writable (W = 1)
 - Present (P = 1)
 - Base = any value

The segment registers must be loaded with non-null segment selectors or the segment registers will be unusable in real-address mode. Note that if the segment registers are not reloaded, execution continues using the descriptor attributes loaded during protected mode.

5. Execute an LIDT instruction to point to a real-address mode interrupt table that is within the 1-MByte real-address mode address range.
6. Clear the PE flag in the CR0 register to switch to real-address mode.
7. Execute a far JMP instruction to jump to a real-address mode program. This operation flushes the instruction queue and loads the appropriate base-address value in the CS register.
8. Load the SS, DS, ES, FS, and GS registers as needed by the real-address mode code. If any of the registers are not going to be used in real-address mode, write 0s to them.
9. Execute the STI instruction to enable maskable hardware interrupts and perform the necessary hardware operation to enable NMI interrupts.

NOTE

All the code that is executed in steps 1 through 9 must be in a single page and the linear addresses in that page must be identity mapped to physical addresses.

10.10 INITIALIZATION AND MODE SWITCHING EXAMPLE

This section provides an initialization and mode switching example that can be incorporated into an application. This code was originally written to initialize the Intel386 processor, but it will execute successfully on the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The code in this example is intended to reside in EPROM and to run following a hardware reset of the processor. The function of the code is to do the following:

- Establish a basic real-address mode operating environment.

- Load the necessary protected-mode system data structures into RAM.
- Load the system registers with the necessary pointers to the data structures and the appropriate flag settings for protected-mode operation.
- Switch the processor to protected mode.

Figure 10-3 shows the physical memory layout for the processor following a hardware reset and the starting point of this example. The EPROM that contains the initialization code resides at the upper end of the processor's physical memory address range, starting at address FFFFFFFFH and going down from there. The address of the first instruction to be executed is at FFFFFFF0H, the default starting address for the processor following a hardware reset.

The main steps carried out in this example are summarized in Table 10-5. The source listing for the example (with the filename STARTUP.ASM) is given in Example 10-1. The line numbers given in Table 10-5 refer to the source listing.

The following are some additional notes concerning this example:

- When the processor is switched into protected mode, the original code segment base-address value of FFFF0000H (located in the hidden part of the CS register) is retained and execution continues from the current offset in the EIP register. The processor will thus continue to execute code in the EPROM until a far jump or call is made to a new code segment, at which time, the base address in the CS register will be changed.
- Maskable hardware interrupts are disabled after a hardware reset and should remain disabled until the necessary interrupt handlers have been installed. The NMI interrupt is not disabled following a reset. The NMI# pin must thus be inhibited from being asserted until an NMI handler has been loaded and made available to the processor.
- The use of a temporary GDT allows simple transfer of tables from the EPROM to anywhere in the RAM area. A GDT entry is constructed with its base pointing to address 0 and a limit of 4 GBytes. When the DS and ES registers are loaded with this descriptor, the temporary GDT is no longer needed and can be replaced by the application GDT.
- This code loads one TSS and no LDTs. If more TSSs exist in the application, they must be loaded into RAM. If there are LDTs they may be loaded as well.

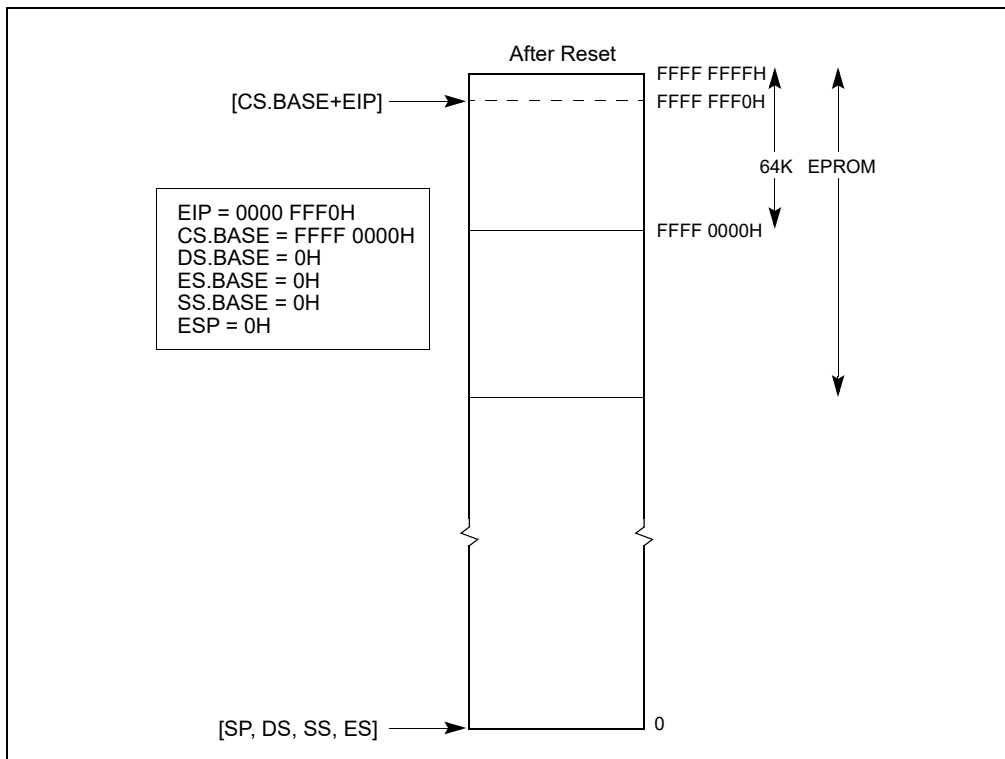


Figure 10-3. Processor State After Reset

Table 10-5. Main Initialization Steps in STARTUP.ASM Source Listing

STARTUP.ASM Line Numbers		Description
From	To	
157	157	Jump (short) to the entry code in the EPROM
162	169	Construct a temporary GDT in RAM with one entry: 0 - null 1 - R/W data segment, base = 0, limit = 4 GBytes
171	172	Load the GDTR to point to the temporary GDT
174	177	Load CRO with PE flag set to switch to protected mode
179	181	Jump near to clear real mode instruction queue
184	186	Load DS, ES registers with GDT[1] descriptor, so both point to the entire physical memory space
188	195	Perform specific board initialization that is imposed by the new protected mode
196	218	Copy the application's GDT from ROM into RAM
220	238	Copy the application's IDT from ROM into RAM
241	243	Load application's GDTR
244	245	Load application's IDTR
247	261	Copy the application's TSS from ROM into RAM
263	267	Update TSS descriptor and other aliases in GDT (GDT alias or IDT alias)
277	277	Load the task register (without task switch) using LTR instruction
282	286	Load SS, ESP with the value found in the application's TSS
287	287	Push EFLAGS value found in the application's TSS
288	288	Push CS value found in the application's TSS
289	289	Push EIP value found in the application's TSS
290	293	Load DS, ES with the value found in the application's TSS
296	296	Perform IRET; pop the above values and enter the application code

10.10.1 Assembler Usage

In this example, the Intel assembler ASM386 and build tools BLD386 are used to assemble and build the initialization code module. The following assumptions are used when using the Intel ASM386 and BLD386 tools.

- The ASM386 will generate the right operand size opcodes according to the code-segment attribute. The attribute is assigned either by the ASM386 invocation controls or in the code-segment definition.
- If a code segment that is going to run in real-address mode is defined, it must be set to a USE 16 attribute. If a 32-bit operand is used in an instruction in this code segment (for example, MOV EAX, EBX), the assembler automatically generates an operand prefix for the instruction that forces the processor to execute a 32-bit operation, even though its default code-segment attribute is 16-bit.
- Intel's ASM386 assembler allows specific use of the 16- or 32-bit instructions, for example, LGDTW, LGDTD, IRETD. If the generic instruction LGDT is used, the default- segment attribute will be used to generate the right opcode.

10.10.2 STARTUP.ASM Listing

Example 10-1 provides high-level sample code designed to move the processor into protected mode. This listing does not include any opcode and offset information.

Example 10-1. STARTUP.ASM

MS-DOS* 5.0(045-N) 386(TM) MACRO ASSEMBLER STARTUP 09:44:51 08/19/92 PAGE 1

MS-DOS 5.0(045-N) 386(TM) MACRO ASSEMBLER V4.0, ASSEMBLY OF MODULE STARTUP
 OBJECT MODULE PLACED IN startup.obj
 ASSEMBLER INVOKED BY: f:\386tools\ASM386.EXE startup.a58 pw (132)

```

LINE      SOURCE

1         NAME      STARTUP
2
3         ;;;;;;;;;;;;;;
4         ;
5         ;   ASSUMPTIONS:
6         ;
7         ;   1.  The bottom 64K of memory is ram, and can be used for
8         ;       scratch space by this module.
9         ;
10        ;   2.  The system has sufficient free usable ram to copy the
11        ;       initial GDT, IDT, and TSS
12        ;
13        ;;;;;;;;;;;;;;
14
15        ; configuration data - must match with build definition
16
17        CS_BASE      EQU      0FFFF0000H
18
19        ; CS_BASE is the linear address of the segment STARTUP_CODE
20        ; - this is specified in the build language file
21
22        RAM_START     EQU      400H
23
24        ; RAM_START is the start of free, usable ram in the linear
25        ; memory space.  The GDT, IDT, and initial TSS will be
26        ; copied above this space, and a small data segment will be
27        ; discarded at this linear address.  The 32-bit word at
28        ; RAM_START will contain the linear address of the first
29        ; free byte above the copied tables - this may be useful if
30        ; a memory manager is used.
31
32        TSS_INDEX     EQU      10
33
34        ; TSS_INDEX is the index of the TSS of the first task to
35        ; run after startup
36
37
38        ;;;;;;;;;;;;;;
39
40        ; ----- STRUCTURES and EQU -----
41        ; structures for system data
42
43        ; TSS structure
44        TASK_STATE     STRUC
45        link          DW ?

```


PROCESSOR MANAGEMENT AND INITIALIZATION

```
46     link_h     DW ?
47     ESP0      DD ?
48     SS0       DW ?
49     SS0_h     DW ?
50     ESP1      DD ?
51     SS1       DW ?
52     SS1_h     DW ?
53     ESP2      DD ?
54     SS2       DW ?
55     SS2_h     DW ?
56     CR3_reg   DD ?
57     EIP_reg   DD ?
58     EFLAGS_reg DD ?
59     EAX_reg   DD ?
60     ECX_reg   DD ?
61     EDX_reg   DD ?
62     EBX_reg   DD ?
63     ESP_reg   DD ?
64     EBP_reg   DD ?
65     ESI_reg   DD ?
66     EDI_reg   DD ?
67     ES_reg    DW ?
68     ES_h     DW ?
69     CS_reg    DW ?
70     CS_h     DW ?
71     SS_reg    DW ?
72     SS_h     DW ?
73     DS_reg    DW ?
74     DS_h     DW ?
75     FS_reg    DW ?
76     FS_h     DW ?
77     GS_reg    DW ?
78     GS_h     DW ?
79     LDT_reg   DW ?
80     LDT_h     DW ?
81     TRAP_reg  DW ?
82     IO_map_base DW ?
83 TASK_STATE ENDS
84
85 ; basic structure of a descriptor
86 DESC     STRUC
87     lim_0_15 DW ?
88     bas_0_15 DW ?
89     bas_16_23 DB ?
90     access   DB ?
91     gran     DB ?
92     bas_24_31 DB ?
93 DESC     ENDS
94
95 ; structure for use with LGDT and LIDT instructions
96 TABLE_REG STRUC
97     table_lim DW ?
98     table_linear DD ?
99 TABLE_REG ENDS
```

```

100
101 ; offset of GDT and IDT descriptors in builder generated GDT
102 GDT_DESC_OFF EQU 1*SIZE(DESC)
103 IDT_DESC_OFF EQU 2*SIZE(DESC)
104
105 ; equates for building temporary GDT in RAM
106 LINEAR_SEL EQU 1*SIZE(DESC)
107 LINEAR_PROTO_LO EQU 00000FFFFH ; LINEAR_ALIAS
108 LINEAR_PROTO_HI EQU 000CF9200H
109
110 ; Protection Enable Bit in CR0
111 PE_BIT EQU 1B
112
113 ; -----
114
115 ; ----- DATA SEGMENT-----
116
117 ; Initially, this data segment starts at linear 0, according
118 ; to the processor's power-up state.
119
120 STARTUP_DATA SEGMENT RW
121
122 free_mem_linear_base LABEL DWORD
123 TEMP_GDT LABEL BYTE ; must be first in segment
124 TEMP_GDT_NULL_DESC DESC <>
125 TEMP_GDT_LINEAR_DESC DESC <>
126
127 ; scratch areas for LGDT and LIDT instructions
128 TEMP_GDT_SCRATCH TABLE_REG <>
129 APP_GDT_RAM TABLE_REG <>
130 APP_IDT_RAM TABLE_REG <>
131 ; align end_data
132 fill DW ?
133
134 ; last thing in this segment - should be on a dword boundary
135 end_data LABEL BYTE
136
137 STARTUP_DATA ENDS
138 ; -----
139
140
141 ; ----- CODE SEGMENT-----
142 STARTUP_CODE SEGMENT ER PUBLIC USE16
143
144 ; filled in by builder
145 PUBLIC GDT_EPROM
146 GDT_EPROM TABLE_REG <>
147
148 ; filled in by builder
149 PUBLIC IDT_EPROM
150 IDT_EPROM TABLE_REG <>
151
152 ; entry point into startup code - the bootstrap will vector
153 ; here with a near JMP generated by the builder. This

```

PROCESSOR MANAGEMENT AND INITIALIZATION

```
154 ; label must be in the top 64K of linear memory.
155
156     PUBLIC  STARTUP
157 STARTUP:
158
159 ; DS,ES address the bottom 64K of flat linear memory
160     ASSUME  DS:STARTUP_DATA, ES:STARTUP_DATA
161 ; See Figure 10-4
162 ; load GDTR with temporary GDT
163     LEA    EBX,TEMP_GDT ; build the TEMP_GDT in low ram,
164     MOV    DWORD PTR [EBX],0 ; where we can address
165     MOV    DWORD PTR [EBX]+4,0
166     MOV    DWORD PTR [EBX]+8, LINEAR_PROTO_LO
167     MOV    DWORD PTR [EBX]+12, LINEAR_PROTO_HI
168     MOV    TEMP_GDT_scratch.table_linear,EBX
169     MOV    TEMP_GDT_scratch.table_lim,15
170
171     DB 66H; execute a 32 bit LGDT
172     LGDT  TEMP_GDT_scratch
173
174 ; enter protected mode
175     MOV    EBX,CR0
176     OR    EBX,PE_BIT
177     MOV    CR0,EBX
178
179 ; clear prefetch queue
180     JMP    CLEAR_LABEL
181 CLEAR_LABEL:
182
183 ; make DS and ES address 4G of linear memory
184     MOV    CX,LINEAR_SEL
185     MOV    DS,CX
186     MOV    ES,CX
187
188 ; do board specific initialization
189 ;
190     ;
191     ; .....
192     ;
193
194
195     ; See Figure 10-5
196     ; copy EPROM GDT to ram at:
197     ;             RAM_START + size (STARTUP_DATA)
198     MOV    EAX,RAM_START
199     ADD    EAX,OFFSET (end_data)
200     MOV    EBX,RAM_START
201     MOV    ECX, CS_BASE
202     ADD    ECX, OFFSET (GDT_EPROM)
203     MOV    ESI, [ECX].table_linear
204     MOV    EDI,EAX
205     MOVZX  ECX, [ECX].table_lim
206     MOV    APP_GDT_ram[EBX].table_lim,CX
```

```

207     INC     ECX
208     MOV     EDX,EAX
209     MOV     APP_GDT_ram[EBX].table_linear,EAX
210     ADD     EAX,ECX
211     REP MOVSB     BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
212
213     ; fixup GDT base in descriptor
214     MOV     ECX,EDX
215     MOV     [EDX].bas_0_15+GDT_DESC_OFF,CX
216     ROR     ECX,16
217     MOV     [EDX].bas_16_23+GDT_DESC_OFF,CL
218     MOV     [EDX].bas_24_31+GDT_DESC_OFF,CH
219
220     ; copy EPROM IDT to ram at:
221     ; RAM_START+size(STARTUP_DATA)+SIZE (EPROM GDT)
222     MOV     ECX, CS_BASE
223     ADD     ECX, OFFSET (IDT_EPROM)
224     MOV     ESI, [ECX].table_linear
225     MOV     EDI,EAX
226     MOVZX  ECX, [ECX].table_lim
227     MOV     APP_IDT_ram[EBX].table_lim,CX
228     INC     ECX
229     MOV     APP_IDT_ram[EBX].table_linear,EAX
230     MOV     EBX,EAX
231     ADD     EAX,ECX
232     REP MOVSB     BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
233
234     ; fixup IDT pointer in GDT
235     MOV     [EDX].bas_0_15+IDT_DESC_OFF,BX
236     ROR     EBX,16
237     MOV     [EDX].bas_16_23+IDT_DESC_OFF,BL
238     MOV     [EDX].bas_24_31+IDT_DESC_OFF,BH
239
240     ; load GDTR and IDTR
241     MOV     EBX,RAM_START
242     DB     66H           ; execute a 32 bit LGDT
243     LGDT   APP_GDT_ram[EBX]
244     DB     66H           ; execute a 32 bit LIDT
245     LIDT   APP_IDT_ram[EBX]
246
247     ; move the TSS
248     MOV     EDI,EAX
249     MOV     EBX,TSS_INDEX*SIZE(DESC)
250     MOV     ECX,GDT_DESC_OFF ;build linear address for TSS
251     MOV     GS,CX
252     MOV     DH,GS:[EBX].bas_24_31
253     MOV     DL,GS:[EBX].bas_16_23
254     ROL     EDX,16
255     MOV     DX,GS:[EBX].bas_0_15
256     MOV     ESI,EDX
257     LSL     ECX,EBX
258     INC     ECX
259     MOV     EDX,EAX
260     ADD     EAX,ECX

```

PROCESSOR MANAGEMENT AND INITIALIZATION

```
261     REP MOVS     BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
262
263         ; fixup TSS pointer
264     MOV     GS:[EBX].bas_0_15,DX
265     ROL     EDX,16
266     MOV     GS:[EBX].bas_24_31,DH
267     MOV     GS:[EBX].bas_16_23,DL
268     ROL     EDX,16
269     ;save start of free ram at linear location RAMSTART
270     MOV     free_mem_linear_base+RAM_START,EAX
271
272     ;assume no LDT used in the initial task - if necessary,
273     ;code to move the LDT could be added, and should resemble
274     ;that used to move the TSS
275
276     ; load task register
277     LTR     BX ; No task switch, only descriptor loading
278     ; See Figure 10-6
279     ; load minimal set of registers necessary to simulate task
280     ; switch
281
282
283     MOV     AX,[EDX].SS_reg ; start loading registers
284     MOV     EDI,[EDX].ESP_reg
285     MOV     SS,AX
286     MOV     ESP,EDI ; stack now valid
287     PUSH   DWORD PTR [EDX].EFLAGS_reg
288     PUSH   DWORD PTR [EDX].CS_reg
289     PUSH   DWORD PTR [EDX].EIP_reg
290     MOV     AX,[EDX].DS_reg
291     MOV     BX,[EDX].ES_reg
292     MOV     DS,AX ; DS and ES no longer linear memory
293     MOV     ES,BX
294
295     ; simulate far jump to initial task
296     IRETD
297
298     STARTUP_CODE ENDS
*** WARNING #377 IN 298, (PASS 2) SEGMENT CONTAINS PRIVILEGED INSTRUCTION(S)
299
300     END STARTUP, DS:STARTUP_DATA, SS:STARTUP_DATA
301
302
```

ASSEMBLY COMPLETE, 1 WARNING, NO ERRORS.

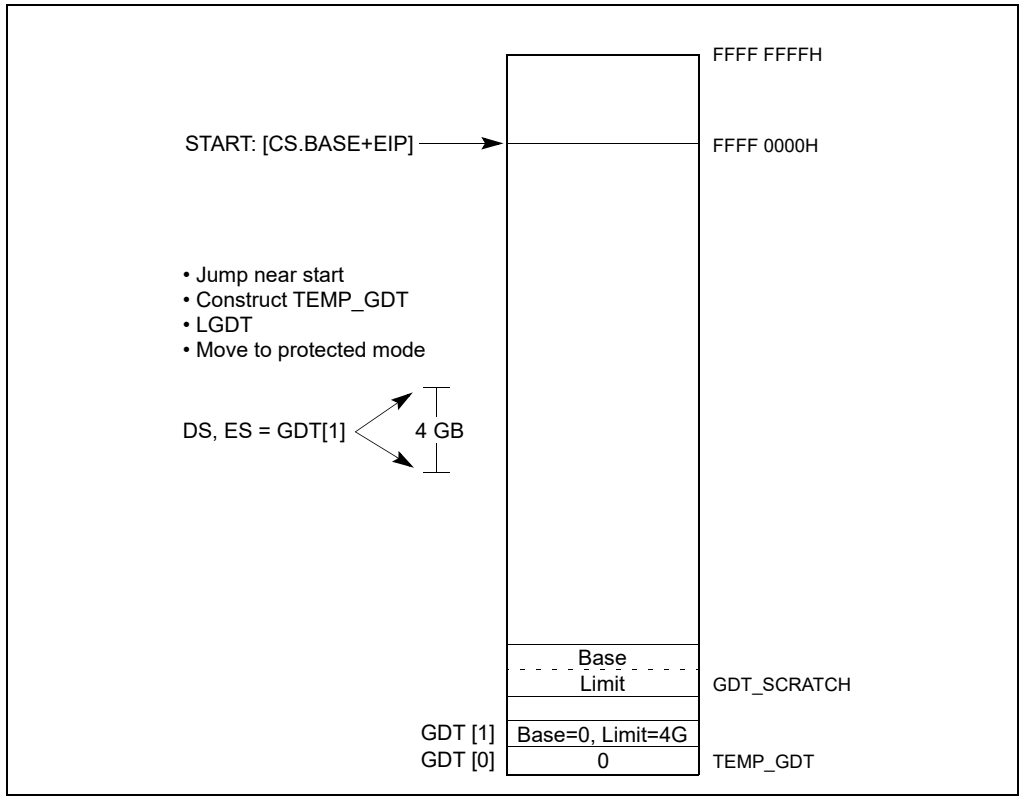


Figure 10-4. Constructing Temporary GDT and Switching to Protected Mode (Lines 162-172 of List File)

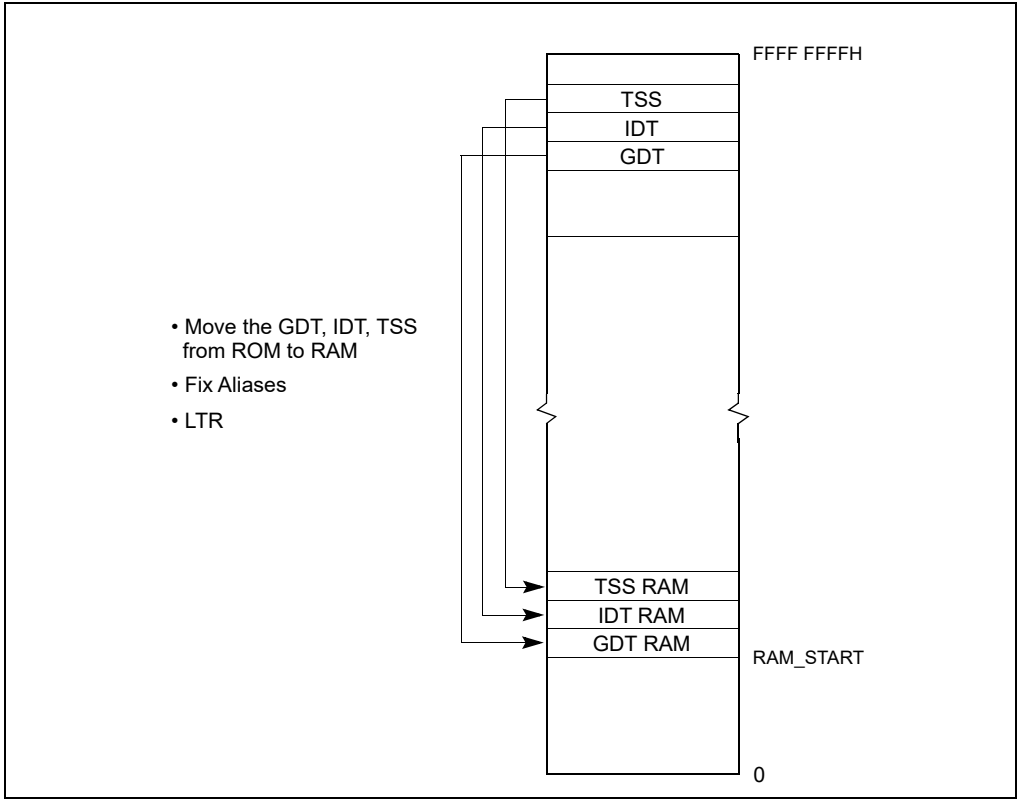


Figure 10-5. Moving the GDT, IDT, and TSS from ROM to RAM (Lines 196-261 of List File)

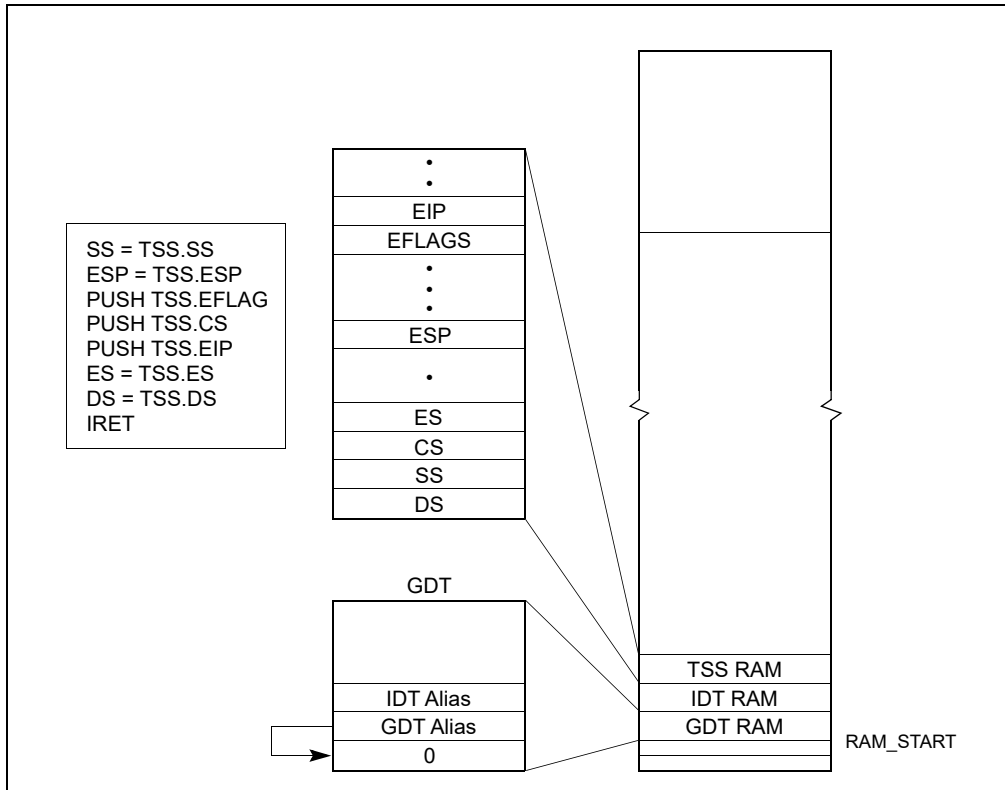


Figure 10-6. Task Switching (Lines 282-296 of List File)

10.10.3 MAIN.ASM Source Code

The file MAIN.ASM shown in Example 10-2 defines the data and stack segments for this application and can be substituted with the main module task written in a high-level language that is invoked by the IRET instruction executed by STARTUP.ASM.

Example 10-2. MAIN.ASM

```

NAME    main_module
data    SEGMENT RW
        dw 1000 dup(?)
DATA    ENDS
stack   stackseg 800
CODE    SEGMENT ER use32 PUBLIC
main_start:
        nop
        nop
        nop
CODE    ENDS
END     main_start, ds:data, ss:stack

```

10.10.4 Supporting Files

The batch file shown in Example 10-3 can be used to assemble the source code files STARTUP.ASM and MAIN.ASM and build the final application.

Example 10-3. Batch File to Assemble and Build the Application

```
ASM386 STARTUP.ASM
ASM386 MAIN.ASM
BLD386 STARTUP.OBJ, MAIN.OBJ buildfile(EPROM.BLD) bootstrap(STARTUP) Bootload
```

BLD386 performs several operations in this example:
 It allocates physical memory location to segments and tables.
 It generates tables using the build file and the input files.
 It links object files and resolves references.
 It generates a boot-loadable file to be programmed into the EPROM.

Example 10-4 shows the build file used as an input to BLD386 to perform the above functions.

Example 10-4. Build File

```
INIT_BLD_EXAMPLE;

SEGMENT
    *SEGMENTS(DPL = 0)
    , startup.startup_code(BASE = 0FFFF0000H)
    ;

TASK
    BOOT_TASK(OBJECT = startup, INITIAL,DPL = 0,
              NOT INTENABLED)
    , PROTECTED_MODE_TASK(OBJECT = main_module,DPL = 0,
                          NOT INTENABLED)
    ;

TABLE
    GDT (
        LOCATION = GDT_EPROM
        , ENTRY = (
            10: PROTECTED_MODE_TASK
            , startup.startup_code
            , startup.startup_data
            , main_module.data
            , main_module.code
            , main_module.stack
            )
        ),
    IDT (
        LOCATION = IDT_EPROM
        );

MEMORY
    (
        RESERVE = (0..3FFFH
                  -- Area for the GDT, IDT, TSS copied from ROM
                  , 60000H..0FFFFFFFHH)
        , RANGE = (ROM_AREA = ROM (0FFFF0000H..0FFFFFFFHH)
                  -- Eprom size 64K
                  , RANGE = (RAM_AREA = RAM (4000H..05FFFFH))
```

);

END

Table 10-6 shows the relationship of each build item with an ASM source file.

Table 10-6. Relationship Between BLD Item and ASM Source File

Item	ASM386 and Startup.A58	BLD386 Controls and BLD file	Effect
Bootstrap	public startup startup:	bootstrap start(startup)	Near jump at OFFFFFFFF0H to start.
GDT location	public GDT_EEPROM GDT_EEPROM TABLE_REG <>	TABLE GDT(location = GDT_EEPROM)	The location of the GDT will be programmed into the GDT_EEPROM location.
IDT location	public IDT_EEPROM IDT_EEPROM TABLE_REG <>	TABLE IDT(location = IDT_EEPROM)	The location of the IDT will be programmed into the IDT_EEPROM location.
RAM start	RAM_START equ 400H	memory (reserve = (0..3FFFH))	RAM_START is used as the ram destination for moving the tables. It must be excluded from the application's segment area.
Location of the application TSS in the GDT	TSS_INDEX EQU 10	TABLE GDT(ENTRY = (10: PROTECTED_MODE_ TASK))	Put the descriptor of the application TSS in GDT entry 10.
EPROM size and location	size and location of the initialization code	SEGMENT startup.code (base = OFFFF0000H) ...memory (RANGE(ROM_AREA = ROM(x..y))	Initialization code size must be less than 64K and resides at upper most 64K of the 4-GByte memory space.

10.11 MICROCODE UPDATE FACILITIES

The P6 family and later processors have the capability to correct errata by loading an Intel-supplied data block into the processor. The data block is called a microcode update. This section describes the mechanisms the BIOS needs to provide in order to use this feature during system initialization. It also describes a specification that permits the incorporation of future updates into a system BIOS.

Intel considers the release of a microcode update for a silicon revision to be the equivalent of a processor stepping and completes a full-stepping level validation for releases of microcode updates.

A microcode update is used to correct errata in the processor. The BIOS, which has an update loader, is responsible for loading the update on processors during system initialization (Figure 10-7). There are two steps to this process: the first is to incorporate the necessary update data blocks into the BIOS; the second is to load update data blocks into the processor.

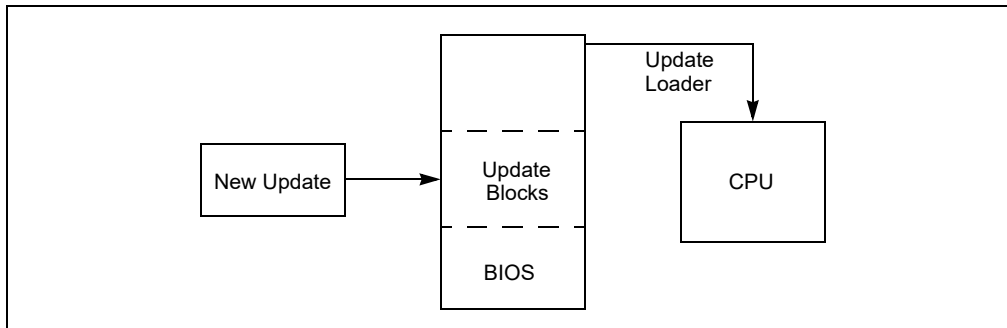


Figure 10-7. Applying Microcode Updates

10.11.1 Microcode Update

A microcode update consists of an Intel-supplied binary that contains a descriptive header and data. No executable code resides within the update. Each microcode update is tailored for a specific list of processor signatures. A mismatch of the processor’s signature with the signature contained in the update will result in a failure to load. A processor signature includes the extended family, extended model, type, family, model, and stepping of the processor (starting with processor family 0FH, model 03H, a given microcode update may be associated with one of multiple processor signatures; see Section 10.11.2 for details).

Microcode updates are composed of a multi-byte header, followed by encrypted data and then by an optional extended signature table. Table 10-7 provides a definition of the fields; Table 10-8 shows the format of an update.

The header is 48 bytes. The first 4 bytes of the header contain the header version. The update header and its reserved fields are interpreted by software based upon the header version. An encoding scheme guards against tampering and provides a means for determining the authenticity of any given update. For microcode updates with a data size field equal to 00000000H, the size of the microcode update is 2048 bytes. The first 48 bytes contain the microcode update header. The remaining 2000 bytes contain encrypted data.

For microcode updates with a data size not equal to 00000000H, the total size field specifies the size of the microcode update. The first 48 bytes contain the microcode update header. The second part of the microcode update is the encrypted data. The data size field of the microcode update header specifies the encrypted data size, its value must be a multiple of the size of DWORD. The total size field of the microcode update header specifies the encrypted data size plus the header size; its value must be in multiples of 1024 bytes (1 KBytes). The optional extended signature table if implemented follows the encrypted data, and its size is calculated by (Total Size – (Data Size + 48)).

NOTE

The optional extended signature table is supported starting with processor family 0FH, model 03H.

Table 10-7. Microcode Update Field Definitions

Field Name	Offset (bytes)	Length (bytes)	Description
Header Version	0	4	Version number of the update header.
Update Revision	4	4	Unique version number for the update, the basis for the update signature provided by the processor to indicate the current update functioning within the processor. Used by the BIOS to authenticate the update and verify that the processor loads successfully. The value in this field cannot be used for processor stepping identification alone. This is a signed 32-bit number.
Date	8	4	Date of the update creation in binary format: mmddyyyy (e.g., 07/18/98 is 07181998H).

Table 10-7. Microcode Update Field Definitions (Contd.)

Field Name	Offset (bytes)	Length (bytes)	Description
Processor Signature	12	4	<i>Extended family, extended model, type, family, model, and stepping</i> of processor that requires this particular update revision (e.g., 00000650H). Each microcode update is designed specifically for a given extended family, extended model, <i>type, family, model, and stepping</i> of the processor. Software should use the processor signature field in conjunction with the CPUID instruction to determine whether or not an update is appropriate to load on a processor. The information encoded within this field exactly corresponds to the bit representations returned by the CPUID instruction.
Checksum	16	4	Checksum of Update Data and Header. Used to verify the integrity of the update header and data. Checksum is correct when the summation of all the DWORDs (including the extended Processor Signature Table) that comprise the microcode update result in 00000000H.
Loader Revision	20	4	Version number of the loader program needed to correctly load this update. The initial version is 00000001H.
Processor Flags	24	4	Platform type information is encoded in the lower 8 bits of this 4-byte field. Each bit represents a particular platform type for a given CPUID. Software should use the processor flags field in conjunction with the platform Id bits in MSR (17H) to determine whether or not an update is appropriate to load on a processor. Multiple bits may be set representing support for multiple platform IDs.
Data Size	28	4	Specifies the size of the encrypted data in bytes, and must be a multiple of DWORDs. If this value is 00000000H, then the microcode update encrypted data is 2000 bytes (or 500 DWORDs).
Total Size	32	4	Specifies the total size of the microcode update in bytes. It is the summation of the header size, the encrypted data size and the size of the optional extended signature table. This value is always a multiple of 1024.
Reserved	36	12	Reserved fields for future expansion.
Update Data	48	Data Size or 2000	Update data.
Extended Signature Count	Data Size + 48	4	Specifies the number of extended signature structures (Processor Signature[n], processor flags[n] and checksum[n]) that exist in this microcode update.
Extended Checksum	Data Size + 52	4	Checksum of update extended processor signature table. Used to verify the integrity of the extended processor signature table. Checksum is correct when the summation of the DWORDs that comprise the extended processor signature table results in 00000000H.
Reserved	Data Size + 56	12	Reserved fields.

Table 10-7. Microcode Update Field Definitions (Contd.)

Field Name	Offset (bytes)	Length (bytes)	Description
Processor Signature[n]	Data Size + 68 + (n * 12)	4	<p><i>Extended family, extended model, type, family, model, and stepping</i> of processor that requires this particular update revision (e.g., 00000650H). Each microcode update is designed specifically for a given extended family, extended model, <i>type, family, model, and stepping</i> of the processor.</p> <p>Software should use the processor signature field in conjunction with the CPUID instruction to determine whether or not an update is appropriate to load on a processor. The information encoded within this field exactly corresponds to the bit representations returned by the CPUID instruction.</p>
Processor Flags[n]	Data Size + 72 + (n * 12)	4	Platform type information is encoded in the lower 8 bits of this 4-byte field. Each bit represents a particular platform type for a given CPUID. Software should use the processor flags field in conjunction with the platform Id bits in MSR (17H) to determine whether or not an update is appropriate to load on a processor. Multiple bits may be set representing support for multiple platform IDs.
Checksum[n]	Data Size + 76 + (n * 12)	4	<p>Used by utility software to decompose a microcode update into multiple microcode updates where each of the new updates is constructed without the optional Extended Processor Signature Table.</p> <p>To calculate the Checksum, substitute the Primary Processor Signature entry and the Processor Flags entry with the corresponding Extended Patch entry. Delete the Extended Processor Signature Table entries. The Checksum is correct when the summation of all DWORDs that comprise the created Extended Processor Patch results in 00000000H.</p>

Table 10-8. Microcode Update Format

31	24	16	8	0	Bytes
Header Version					0
Update Revision					4
Month: 8		Day: 8		Year: 16	8
Processor Signature (CPUID)					12
Res: 4		Extended Mode: 4		Reserved: 2	Type: 2
Extended Family: 8		Family: 4		Model: 4	Stepping: 4
Checksum					16
Loader Revision					20
Processor Flags					24
Reserved (24 bits)					P7 P6 P5 P4 P3 P2 P1 P0
Data Size					28
Total Size					32
Reserved (12 Bytes)					36

Table 10-8. Microcode Update Format (Contd.)

31	24	16	8	0	Bytes
Update Data (Data Size bytes, or 2000 Bytes if Data Size = 00000000H)					48
Extended Signature Count 'n'					Data Size + 48
Extended Processor Signature Table Checksum					Data Size + 52
Reserved (12 Bytes)					Data Size + 56
Processor Signature[n]					Data Size + 68 + (n * 12)
Processor Flags[n]					Data Size + 72 + (n * 12)
Checksum[n]					Data Size + 76 + (n * 12)

10.11.2 Optional Extended Signature Table

The extended signature table is a structure that may be appended to the end of the encrypted data when the encrypted data only supports a single processor signature (optional case). The extended signature table will always be present when the encrypted data supports multiple processor steppings and/or models (required case).

The extended signature table consists of a 20-byte extended signature header structure, which contains the extended signature count, the extended processor signature table checksum, and 12 reserved bytes (Table 10-9). Following the extended signature header structure, the extended signature table contains 0-to-n extended processor signature structures.

Each processor signature structure consist of the processor signature, processor flags, and a checksum (Table 10-10).

The extended signature count in the extended signature header structure indicates the number of processor signature structures that exist in the extended signature table.

The extended processor signature table checksum is a checksum of all DWORDs that comprise the extended signature table. That includes the extended signature count, extended processor signature table checksum, 12 reserved bytes and the n processor signature structures. A valid extended signature table exists when the result of a DWORD checksum is 00000000H.

Table 10-9. Extended Processor Signature Table Header Structure

Extended Signature Count 'n'	Data Size + 48
Extended Processor Signature Table Checksum	Data Size + 52
Reserved (12 Bytes)	Data Size + 56

Table 10-10. Processor Signature Structure

Processor Signature[n]	Data Size + 68 + (n * 12)
Processor Flags[n]	Data Size + 72 + (n * 12)
Checksum[n]	Data Size + 76 + (n * 12)

10.11.3 Processor Identification

Each microcode update is designed to for a specific processor or set of processors. To determine the correct microcode update to load, software must ensure that one of the processor signatures embedded in the microcode update matches the 32-bit processor signature returned by the CPUID instruction when executed by the target processor with EAX = 1. Attempting to load a microcode update that does not match a processor signature embedded in the microcode update with the processor signature returned by CPUID will cause the BIOS to reject the update.

Example 10-5 shows how to check for a valid processor signature match between the processor and microcode update.

Example 10-5. Pseudo Code to Validate the Processor Signature

```
ProcessorSignature ← CPUID(1):EAX

If (Update.HeaderVersion = 00000001h)
{
    // first check the ProcessorSignature field
    If (ProcessorSignature = Update.ProcessorSignature)
        Success

    // if extended signature is present
    Else If (Update.TotalSize > (Update.DataSize + 48))
    {

        //
        // Assume the Data Size has been used to calculate the
        // location of Update.ProcessorSignature[0].
        //

        For (N ← 0; ((N < Update.ExtendedSignatureCount) AND
            (ProcessorSignature ≠ Update.ProcessorSignature[N])); N++);

            // if the loops ended when the iteration count is
            // less than the number of processor signatures in
            // the table, we have a match
            If (N < Update.ExtendedSignatureCount)
                Success
            Else
                Fail
    }
    Else
        Fail
Else
    Fail
```

10.11.4 Platform Identification

In addition to verifying the processor signature, the intended processor platform type must be determined to properly target the microcode update. The intended processor platform type is determined by reading the IA32_PLATFORM_ID register, (MSR 17H). This 64-bit register must be read using the RDMSR instruction.

The three platform ID bits, when read as a binary coded decimal (BCD) number, indicate the bit position in the microcode update header's processor flags field associated with the installed processor. The processor flags in the 48-byte header and the processor flags field associated with the extended processor signature structures may have multiple bits set. Each set bit represents a different platform ID that the update supports.

Register Name: IA32_PLATFORM_ID
MSR Address: 017H

Access: Read Only

IA32_PLATFORM_ID is a 64-bit register accessed only when referenced as a Qword through a RDMSR instruction.

Table 10-11. Processor Flags

Bit	Descriptions																																				
63:53	Reserved																																				
52:50	Platform Id Bits (RO). The field gives information concerning the intended platform for the processor. See also Table 10-8. <table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="padding-right: 10px;">52</td> <td style="padding-right: 10px;">51</td> <td style="padding-right: 10px;">50</td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>Processor Flag 0</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>Processor Flag 1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>Processor Flag 2</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>Processor Flag 3</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>Processor Flag 4</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>Processor Flag 5</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>Processor Flag 6</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>Processor Flag 7</td> </tr> </table>	52	51	50		0	0	0	Processor Flag 0	0	0	1	Processor Flag 1	0	1	0	Processor Flag 2	0	1	1	Processor Flag 3	1	0	0	Processor Flag 4	1	0	1	Processor Flag 5	1	1	0	Processor Flag 6	1	1	1	Processor Flag 7
52	51	50																																			
0	0	0	Processor Flag 0																																		
0	0	1	Processor Flag 1																																		
0	1	0	Processor Flag 2																																		
0	1	1	Processor Flag 3																																		
1	0	0	Processor Flag 4																																		
1	0	1	Processor Flag 5																																		
1	1	0	Processor Flag 6																																		
1	1	1	Processor Flag 7																																		
49:0	Reserved																																				

To validate the platform information, software may implement an algorithm similar to the algorithms in Example 10-6.

Example 10-6. Pseudo Code Example of Processor Flags Test

```

Flag ← 1 << IA32_PLATFORM_ID[52:50]

If (Update.HeaderVersion = 00000001h)
{
  If (Update.ProcessorFlags & Flag)
  {
    Load Update
  }
  Else
  {
    //
    // Assume the Data Size has been used to calculate the
    // location of Update.ProcessorSignature[N] and a match
    // on Update.ProcessorSignature[N] has already succeeded
    //

    If (Update.ProcessorFlags[n] & Flag)
    {
      Load Update
    }
  }
}

```

10.11.5 Microcode Update Checksum

Each microcode update contains a DWORD checksum located in the update header. It is software's responsibility to ensure that a microcode update is not corrupt. To check for a corrupt microcode update, software must perform a

unsigned DWORD (32-bit) checksum of the microcode update. Even though some fields are signed, the checksum procedure treats all DWORDs as unsigned. Microcode updates with a header version equal to 00000001H must sum all DWORDs that comprise the microcode update. A valid checksum check will yield a value of 00000000H. Any other value indicates the microcode update is corrupt and should not be loaded.

The checksum algorithm shown by the pseudo code in Example 10-7 treats the microcode update as an array of unsigned DWORDs. If the data size DWORD field at byte offset 32 equals 00000000H, the size of the encrypted data is 2000 bytes, resulting in 500 DWORDs. Otherwise the microcode update size in DWORDs = $(Total\ Size / 4)$, where the total size is a multiple of 1024 bytes (1 KBytes).

Example 10-7. Pseudo Code Example of Checksum Test

```
N ← 512

If (Update.DataSize ≠ 00000000H)
    N ← Update.TotalSize / 4

ChkSum ← 0
For (I ← 0; I < N; I++)
{
    ChkSum ← ChkSum + MicrocodeUpdate[I]
}

If (ChkSum = 00000000H)
    Success
Else
    Fail
```

10.11.6 Microcode Update Loader

This section describes an update loader used to load an update into a P6 family or later processors. It also discusses the requirements placed on the BIOS to ensure proper loading. The update loader described contains the minimal instructions needed to load an update. The specific instruction sequence that is required to load an update is dependent upon the loader revision field contained within the update header. This revision is expected to change infrequently (potentially, only when new processor models are introduced).

Example 10-8 below represents the update loader with a loader revision of 00000001H. Note that the microcode update must be aligned on a 16-byte boundary and the size of the microcode update must be 1-KByte granular.

Example 10-8. Assembly Code Example of Simple Microcode Update Loader

```
mov  ecx,79h           ; MSR to write in ECX
xor  eax,eax          ; clear EAX
xor  ebx,ebx          ; clear EBX
mov  ax,cs            ; Segment of microcode update
shl  eax,4
mov  bx,offset Update ; Offset of microcode update
add  eax,ebx          ; Linear Address of Update in EAX
add  eax,48d          ; Offset of the Update Data within the Update
xor  edx,edx          ; Zero in EDX
WRMSR                  ; microcode update trigger
```

The loader shown in Example 10-8 assumes that *update* is the address of a microcode update (header and data) embedded within the code segment of the BIOS. It also assumes that the processor is operating in real mode. The data may reside anywhere in memory, aligned on a 16-byte boundary, that is accessible by the processor within its current operating mode.

Before the BIOS executes the microcode update trigger (WRMSR) instruction, the following must be true:

- In 64-bit mode, EAX contains the lower 32-bits of the microcode update linear address. In protected mode, EAX contains the full 32-bit linear address of the microcode update.
- In 64-bit mode, EDX contains the upper 32-bits of the microcode update linear address. In protected mode, EDX equals zero.
- ECX contains 79H (address of IA32_BIOS_UPDT_TRIG).

Other requirements are:

- The addresses for the microcode update data must be in canonical form.
- If paging is enabled, the microcode update data must map that data as present.
- The microcode update data must start at a 16-byte aligned linear address.

10.11.6.1 Hard Resets in Update Loading

The effects of a loaded update are cleared from the processor upon a hard reset. Therefore, each time a hard reset is asserted during the BIOS POST, the update must be reloaded on all processors that observed the reset. The effects of a loaded update are, however, maintained across a processor INIT. There are no side effects caused by loading an update into a processor multiple times.

10.11.6.2 Update in a Multiprocessor System

A multiprocessor (MP) system requires loading each processor with update data appropriate for its CPUID and platform ID bits. The BIOS is responsible for ensuring that this requirement is met and that the loader is located in a module executed by all processors in the system. If a system design permits multiple steppings of Pentium 4, Intel Xeon, and P6 family processors to exist concurrently; then the BIOS must verify individual processors against the update header information to ensure appropriate loading. Given these considerations, it is most practical to load the update during MP initialization.

10.11.6.3 Update in a System Supporting Intel Hyper-Threading Technology

Intel Hyper-Threading Technology has implications on the loading of the microcode update. The update must be loaded for each core in a physical processor. Thus, for a processor supporting Intel Hyper-Threading Technology, only one logical processor per core is required to load the microcode update. Each individual logical processor can independently load the update. However, MP initialization must provide some mechanism (e.g., a software semaphore) to force serialization of microcode update loads and to prevent simultaneous load attempts to the same core.

10.11.6.4 Update in a System Supporting Dual-Core Technology

Dual-core technology has implications on the loading of the microcode update. The microcode update facility is not shared between processor cores in the same physical package. The update must be loaded for each core in a physical processor.

If processor core supports Intel Hyper-Threading Technology, the guideline described in Section 10.11.6.3 also applies.

10.11.6.5 Update Loader Enhancements

The update loader presented in Section 10.11.6, "Microcode Update Loader," is a minimal implementation that can be enhanced to provide additional functionality. Potential enhancements are described below:

- BIOS can incorporate multiple updates to support multiple steppings of the Pentium 4, Intel Xeon, and P6 family processors. This feature provides for operating in a mixed stepping environment on an MP system and enables a user to upgrade to a later version of the processor. In this case, modify the loader to check the CPUID and platform ID bits of the processor that it is running on against the available headers before loading a particular update. The number of updates is only limited by available BIOS space.

- A loader can load the update and test the processor to determine if the update was loaded correctly. See Section 10.11.7, “Update Signature and Verification.”
- A loader can verify the integrity of the update data by performing a checksum on the double words of the update summing to zero. See Section 10.11.5, “Microcode Update Checksum.”
- A loader can provide power-on messages indicating successful loading of an update.

10.11.7 Update Signature and Verification

The P6 family and later processors provide capabilities to verify the authenticity of a particular update and to identify the current update revision. This section describes the model-specific extensions of processors that support this feature. The update verification method below assumes that the BIOS will only verify an update that is more recent than the revision currently loaded in the processor.

CPUID returns a value in a model specific register in addition to its usual register return values. The semantics of CPUID cause it to deposit an update ID value in the 64-bit model-specific register at address 08BH (IA32_BIOS_SIGN_ID). If no update is present in the processor, the value in the MSR remains unmodified. The BIOS must pre-load a zero into the MSR before executing CPUID. If a read of the MSR at 8BH still returns zero after executing CPUID, this indicates that no update is present.

The update ID value returned in the EDX register after RDMSR executes indicates the revision of the update loaded in the processor. This value, in combination with the CPUID value returned in the EAX register, uniquely identifies a particular update. The signature ID can be directly compared with the update revision field in a microcode update header for verification of a correct load. No consecutive updates released for a given stepping of a processor may share the same signature. The processor signature returned by CPUID differentiates updates for different step-pings.

10.11.7.1 Determining the Signature

An update that is successfully loaded into the processor provides a signature that matches the update revision of the currently functioning revision. This signature is available any time after the actual update has been loaded. Requesting the signature does not have a negative impact upon a loaded update.

The procedure for determining this signature shown in Example 10-9.

Example 10-9. Assembly Code to Retrieve the Update Revision

```

MOV    ECX, 08BH           ;IA32_BIOS_SIGN_ID
XOR    EAX, EAX           ;clear EAX
XOR    EDX, EDX          ;clear EDX
WRMSR                ;Load 0 to MSR at 8BH
MOV    EAX, 1
cpuid
MOV    ECX, 08BH           ;IA32_BIOS_SIGN_ID
rdmsr                ;Read Model Specific Register
    
```

If there is an update active in the processor, its revision is returned in the EDX register after the RDMSR instruction executes.

IA32_BIOS_SIGN_ID	Microcode Update Signature Register
MSR Address:	08BH Accessed as a Qword
Default Value:	XXXX XXXX XXXX XXXXh
Access:	Read/Write

The IA32_BIOS_SIGN_ID register is used to report the microcode update signature when CPUID executes. The signature is returned in the upper DWORD (Table 10-12).

Table 10-12. Microcode Update Signature

Bit	Description
63:32	Microcode update signature. This field contains the signature of the currently loaded microcode update when read following the execution of the CPUID instruction, function 1. It is required that this register field be pre-loaded with zero prior to executing the CPUID, function 1. If the field remains equal to zero, then there is no microcode update loaded. Another non-zero value will be the signature.
31:0	Reserved.

10.11.7.2 Authenticating the Update

An update may be authenticated by the BIOS using the signature primitive, described above, and the algorithm in Example 10-10.

Example 10-10. Pseudo Code to Authenticate the Update

```
Z ← Obtain Update Revision from the Update Header to be authenticated;
X ← Obtain Current Update Signature from MSR 8BH;

If (Z > X)
{
  Load Update that is to be authenticated;
  Y ← Obtain New Signature from MSR 8BH;

  If (Z = Y)
    Success
  Else
    Fail
}
Else
  Fail
```

Example 10-10 requires that the BIOS only authenticate updates that contain a numerically larger revision than the currently loaded revision, where Current Signature (X) < New Update Revision (Z). A processor with no loaded update is considered to have a revision equal to zero.

This authentication procedure relies upon the decoding provided by the processor to verify an update from a potentially hostile source. As an example, this mechanism in conjunction with other safeguards provides security for dynamically incorporating field updates into the BIOS.

10.11.8 Optional Processor Microcode Update Specifications

This section an interface that an OEM-BIOS may provide to its client system software to manage processor microcode updates. System software may choose to build its own facility to manage microcode updates (e.g., similar to the facility described in Section 9.11.6) or rely on a facility provided by the BIOS to perform microcode updates.

Sections 10.11.8.1-10.11.8.9 describes an extension (Function 0D042H) to the real mode INT 15H service. INT 15H 0D042H function is one of several alternatives that a BIOS may choose to implement microcode update facility and offer to its client application (e.g., an OS). Other alternative microcode update facility that BIOS can choose are dependent on platform-specific capabilities, including the Capsule Update mechanism from the UEFI specification (www.uefi.org). In this discussion, the application is referred to as the calling program or caller.

The real mode INT15 call specification described here is an Intel extension to an OEM BIOS. This extension allows an application to read and modify the contents of the microcode update data in NVRAM. The update loader, which is part of the system BIOS, cannot be updated by the interface. All of the functions defined in the specification must be implemented for a system to be considered compliant with the specification. The INT15 functions are accessible only from real mode.

10.11.8.1 Responsibilities of the BIOS

If a BIOS passes the presence test (INT 15H, AX = 0D042H, BL = 0H), it must implement all of the sub-functions defined in the INT 15H, AX = 0D042H specification. There are no optional functions. BIOS must load the appropriate update for each processor during system initialization.

A Header Version of an update block containing the value 0FFFFFFFH indicates that the update block is unused and available for storing a new update.

The BIOS is responsible for providing a region of non-volatile storage (NVRAM) for each potential processor stepping within a system. This storage unit consists of one or more update blocks. An update block is a contiguous 2048-byte block of memory. The BIOS for a single processor system need only provide update blocks to store one microcode update. If the BIOS for a multiple processor system is intended to support mixed processor steppings, then the BIOS needs to provide enough update blocks to store each unique microcode update or for each processor socket on the OEM's system board.

The BIOS is responsible for managing the NVRAM update blocks. This includes garbage collection, such as removing microcode updates that exist in NVRAM for which a corresponding processor does not exist in the system. This specification only provides the mechanism for ensuring security, the uniqueness of an entry, and that stale entries are not loaded. The actual update block management is implementation specific on a per-BIOS basis.

As an example, the BIOS may use update blocks sequentially in ascending order with CPU signatures sorted versus the first available block. In addition, garbage collection may be implemented as a setup option to clear all NVRAM slots or as BIOS code that searches and eliminates unused entries during boot.

NOTES

For IA-32 processors starting with family 0FH and model 03H and Intel 64 processors, the microcode update may be as large as 16 KBytes. Thus, BIOS must allocate 8 update blocks for each microcode update. In a MP system, a common microcode update may be sufficient for each socket in the system.

For IA-32 processors earlier than family 0FH and model 03H, the microcode update is 2 KBytes. An MP-capable BIOS that supports multiple steppings must allocate a block for each socket in the system.

A single-processor BIOS that supports variable-sized microcode update and fixed-sized microcode update must allocate one 16-KByte region and a second region of at least 2 KBytes.

The following algorithm (Example 10-11) describes the steps performed during BIOS initialization used to load the updates into the processor(s). The algorithm assumes:

- The BIOS ensures that no update contained within NVRAM has a header version or loader version that does not match one currently supported by the BIOS.
- The update contains a correct checksum.
- The BIOS ensures that (at most) one update exists for each processor stepping.
- Older update revisions are not allowed to overwrite more recent ones.

These requirements are checked by the BIOS during the execution of the write update function of this interface. The BIOS sequentially scans through all of the update blocks in NVRAM starting with index 0. The BIOS scans until it finds an update where the processor fields in the header match the processor signature (extended family, extended model, type, family, model, and stepping) as well as the platform bits of the current processor.

Example 10-11. Pseudo Code, Checks Required Prior to Loading an Update

```

For each processor in the system
{
    Determine the Processor Signature via CPUID function 1;
    Determine the Platform Bits ← 1 << IA32_PLATFORM_ID[52:50];

    For (I ← UpdateBlock 0, I < NumOfBlocks; I++)
    {
        If (Update.Header_Version = 00000001H)
        {

```

```

If ((Update.ProcessorSignature = Processor Signature) &&
    (Update.ProcessorFlags & Platform Bits))
{
    Load Update.UpdateData into the Processor;
    Verify update was correctly loaded into the processor
    Go on to next processor
    Break;
}
Else If (Update.TotalSize > (Update.DataSize + 48))
{
    N ← 0
    While (N < Update.ExtendedSignatureCount)
    {
        If ((Update.ProcessorSignature[N] =
            Processor Signature) &&
            (Update.ProcessorFlags[N] & Platform Bits))
        {
            Load Update.UpdateData into the Processor;
            Verify update correctly loaded into the processor
            Go on to next processor
            Break;
        }
        N ← N + 1
    }
    I ← I + (Update.TotalSize / 2048)
    If ((Update.TotalSize MOD 2048) = 0)
        I ← I + 1
    }
}
}
}

```

NOTES

The platform Id bits in IA32_PLATFORM_ID are encoded as a three-bit binary coded decimal field. The platform bits in the microcode update header are individually bit encoded. The algorithm must do a translation from one format to the other prior to doing a check.

When performing the INT 15H, 0D042H functions, the BIOS must assume that the caller has no knowledge of platform specific requirements. It is the responsibility of BIOS calls to manage all chipset and platform specific prerequisites for managing the NVRAM device. When writing the update data using the Write Update sub-function, the BIOS must maintain implementation specific data requirements (such as the update of NVRAM checksum). The BIOS should also attempt to verify the success of write operations on the storage device used to record the update.

10.11.8.2 Responsibilities of the Calling Program

This section of the document lists the responsibilities of a calling program using the interface specifications to load microcode update(s) into BIOS NVRAM.

- The calling program should call the INT 15H, 0D042H functions from a pure real mode program and should be executing on a system that is running in pure real mode.
- The caller should issue the presence test function (sub function 0) and verify the signature and return codes of that function.
- It is important that the calling program provides the required scratch RAM buffers for the BIOS and the proper stack size as specified in the interface definition.
- The calling program should read any update data that already exists in the BIOS in order to make decisions about the appropriateness of loading the update. The BIOS must refuse to overwrite a newer update with an

older version. The update header contains information about version and processor specifics for the calling program to make an intelligent decision about loading.

- There can be no ambiguous updates. The BIOS must refuse to allow multiple updates for the same CPU to exist at the same time; it also must refuse to load updates for processors that don't exist on the system.
- The calling application should implement a verify function that is run after the update write function successfully completes. This function reads back the update and verifies that the BIOS returned an image identical to the one that was written.

Example 10-12 represents a calling program.

Example 10-12. INT 15 D042 Calling Program Pseudo-code

```
//
// We must be in real mode
//
If the system is not in Real mode exit
//
// Detect presence of Genuine Intel processor(s) that can be updated
// using(CPUID)
//
If no Intel processors exist that can be updated exit
//
// Detect the presence of the Intel microcode update extensions
//
If the BIOS fails the PresenceTestexit
//
// If the APIC is enabled, see if any other processors are out there
//
Read IA32_APICBASE
If APIC enabled
{
    Send Broadcast Message to all processors except self via APIC
    Have all processors execute CPUID, record the Processor Signature
    (i.e.,Extended Family, Extended Model, Type, Family, Model, Stepping)
    Have all processors read IA32_PLATFORM_ID[52:50], record Platform
    Id Bits

    If current processor cannot be updated
        exit
}
//
// Determine the number of unique update blocks needed for this system
//
NumBlocks = 0
For each processor
{
    If ((this is a unique processor stepping) AND
        (we have a unique update in the database for this processor))
    {
        Checksum the update from the database;
        If Checksum fails
            exit
        NumBlocks ← NumBlocks + size of microcode update / 2048
    }
}
//
// Do we have enough update slots for all CPUs?
//
```

```

If there are more blocks required to support the unique processor steppings than update blocks
provided by the BIOS exit
//
// Do we need any update blocks at all?  If not, we are done
//
If (NumBlocks = 0)
    exit
//
// Record updates for processors in NVRAM.
//
For (I=0; I<NumBlocks; I++)
{
    //
    // Load each Update
    //
    Issue the WriteUpdate function

    If (STORAGE_FULL) returned
    {
        Display Error -- BIOS is not managing NVRAM appropriately
        exit
    }

    If (INVALID_REVISION) returned
    {
        Display Message: More recent update already loaded in NVRAM for
        this stepping
        continue
    }

    If any other error returned
    {
        Display Diagnostic
        exit
    }

    //
    // Verify the update was loaded correctly
    //
    Issue the ReadUpdate function

    If an error occurred
    {
        Display Diagnostic
        exit
    }
    //
    // Compare the Update read to that written
    //
    If (Update read ≠ Update written)
    {
        Display Diagnostic
        exit
    }

    I ← I + (size of microcode update / 2048)
}
//
// Enable Update Loading, and inform user

```


//
Issue the Update Control function with Task = Enable.

10.11.8.3 Microcode Update Functions

Table 10-13 defines the processor microcode update functions that implementations of INT 15H 0D042H must support.

Table 10-13. Microcode Update Functions

Microcode Update Function	Function Number	Description	Required/Optional
Presence test	00H	Returns information about the supported functions.	Required
Write update data	01H	Writes one of the update data areas (slots).	Required
Update control	02H	Globally controls the loading of updates.	Required
Read update data	03H	Reads one of the update data areas (slots).	Required

10.11.8.4 INT 15H-based Interface

If an OEM-BIOS is implementing INT 15H 0D042H interface and offer to its client, the BIOS should allow additional microcode updates to be added to system flash.

The program that calls this interface is responsible for providing three 64-kilobyte RAM areas for BIOS use during calls to the read and write functions. These RAM scratch pads can be used by the BIOS for any purpose, but only for the duration of the function call. The calling routine places real mode segments pointing to the RAM blocks in the CX, DX, and SI registers. Calls to functions in this interface must be made with a minimum of 32 kilobytes of stack available to the BIOS.

In general, each function returns with CF cleared and AH contains the returned status. The general return codes and other constant definitions are listed in Section 10.11.8.9, "Return Codes."

The OEM error field (AL) is provided for the OEM to return additional error information specific to the platform. If the BIOS provides no additional information about the error, OEM error must be set to SUCCESS. The OEM error field is undefined if AH contains either SUCCESS (00H) or NOT_IMPLEMENTED (86H). In all other cases, it must be set with either SUCCESS or a value meaningful to the OEM.

The following sections describe functions provided by the INT15H-based interface.

10.11.8.5 Function 00H—Presence Test

This function verifies that the BIOS has implemented required microcode update functions. Table 10-14 lists the parameters and return codes for the function.

Table 10-14. Parameters for the Presence Test

Input		
AX	Function Code	0D042H
BL	Sub-function	00H - Presence test
Output		
CF	Carry Flag	Carry Set - Failure - AH contains status Carry Clear - All return values valid
AH	Return Code	
AL	OEM Error	Additional OEM information.
EBX	Signature Part 1	'INTE' - Part one of the signature
ECX	Signature Part 2	'LPEP' - Part two of the signature
EDX	Loader Version	Version number of the microcode update loader

Table 10-14. Parameters for the Presence Test (Contd.)

Input		
SI	Update Count	Number of 2048 update blocks in NVRAM the BIOS allocated to storing microcode updates
Return Codes (see Table 10-19 for code definitions)		
SUCCESS		The function completed successfully.
NOT_IMPLEMENTED		The function is not implemented.

In order to assure that the BIOS function is present, the caller must verify the carry flag, the return code, and the 64-bit signature. The update count reflects the number of 2048-byte blocks available for storage within one non-volatile RAM.

The loader version number refers to the revision of the update loader program that is included in the system BIOS image.

10.11.8.6 Function 01H—Write Microcode Update Data

This function integrates a new microcode update into the BIOS storage device. Table 10-15 lists the parameters and return codes for the function.

Table 10-15. Parameters for the Write Update Data Function

Input		
AX	Function Code	0D042H
BL	Sub-function	01H - Write update
ES:DI	Update Address	Real Mode pointer to the Intel Update structure. This buffer is 2048 bytes in length if the processor supports only fixed-size microcode update or... Real Mode pointer to the Intel Update structure. This buffer is 64 KBytes in length if the processor supports a variable-size microcode update.
CX	Scratch Pad1	Real mode segment address of 64 KBytes of RAM block
DX	Scratch Pad2	Real mode segment address of 64 KBytes of RAM block
SI	Scratch Pad3	Real mode segment address of 64 KBytes of RAM block
SS:SP	Stack pointer	32 KBytes of stack minimum
Output		
CF	Carry Flag	Carry Set - Failure - AH Contains status Carry Clear - All return values valid
AH	Return Code	Status of the call
AL	OEM Error	Additional OEM information
Return Codes (see Table 10-19 for code definitions)		
SUCCESS		The function completed successfully.
NOT_IMPLEMENTED		The function is not implemented.
WRITE_FAILURE		A failure occurred because of the inability to write the storage device.
ERASE_FAILURE		A failure occurred because of the inability to erase the storage device.
READ_FAILURE		A failure occurred because of the inability to read the storage device.

Table 10-15. Parameters for the Write Update Data Function (Contd.)

Input	
STORAGE_FULL	The BIOS non-volatile storage area is unable to accommodate the update because all available update blocks are filled with updates that are needed for processors in the system.
CPU_NOT_PRESENT	The processor stepping does not currently exist in the system.
INVALID_HEADER	The update header contains a header or loader version that is not recognized by the BIOS.
INVALID_HEADER_CS	The update does not checksum correctly.
SECURITY_FAILURE	The processor rejected the update.
INVALID_REVISION	The same or more recent revision of the update exists in the storage device.

Description

The BIOS is responsible for selecting an appropriate update block in the non-volatile storage for storing the new update. This BIOS is also responsible for ensuring the integrity of the information provided by the caller, including authenticating the proposed update before incorporating it into storage.

Before writing the update block into NVRAM, the BIOS should ensure that the update structure meets the following criteria in the following order:

1. The update header version should be equal to an update header version recognized by the BIOS.
2. The update loader version in the update header should be equal to the update loader version contained within the BIOS image.
3. The update block must checksum. This checksum is computed as a 32-bit summation of all double words in the structure, including the header, data, and processor signature table.

The BIOS selects update block(s) in non-volatile storage for storing the candidate update. The BIOS can select any available update block as long as it guarantees that only a single update exists for any given processor stepping in non-volatile storage. If the update block selected already contains an update, the following additional criteria apply to overwrite it:

- The processor signature in the proposed update must be equal to the processor signature in the header of the current update in NVRAM (Processor Signature + platform ID bits).
- The update revision in the proposed update should be greater than the update revision in the header of the current update in NVRAM.

If no unused update blocks are available and the above criteria are not met, the BIOS can overwrite update block(s) for a processor stepping that is no longer present in the system. This can be done by scanning the update blocks and comparing the processor steppings, identified in the MP Specification table, to the processor steppings that currently exist in the system.

Finally, before storing the proposed update in NVRAM, the BIOS must verify the authenticity of the update via the mechanism described in Section 10.11.6, "Microcode Update Loader." This includes loading the update into the current processor, executing the CPUID instruction, reading MSR 08Bh, and comparing a calculated value with the update revision in the proposed update header for equality.

When performing the write update function, the BIOS must record the entire update, including the header, the update data, and the extended processor signature table (if applicable). When writing an update, the original contents may be overwritten, assuming the above criteria have been met. It is the responsibility of the BIOS to ensure that more recent updates are not overwritten through the use of this BIOS call, and that only a single update exists within the NVRAM for any processor stepping and platform ID.

Figure 10-8 and Figure 10-9 show the process the BIOS follows to choose an update block and ensure the integrity of the data when it stores the new microcode update.

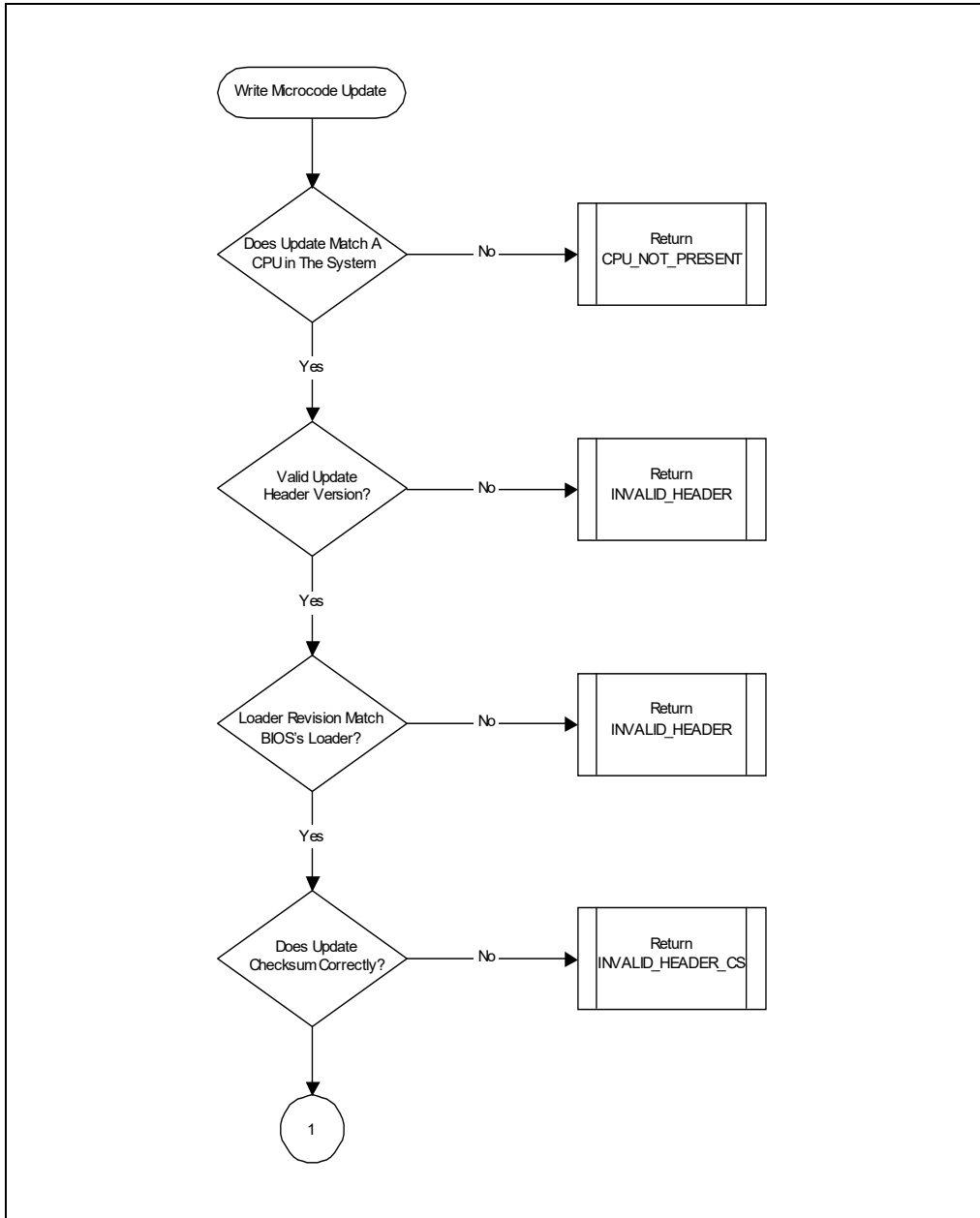


Figure 10-8. Microcode Update Write Operation Flow [1]

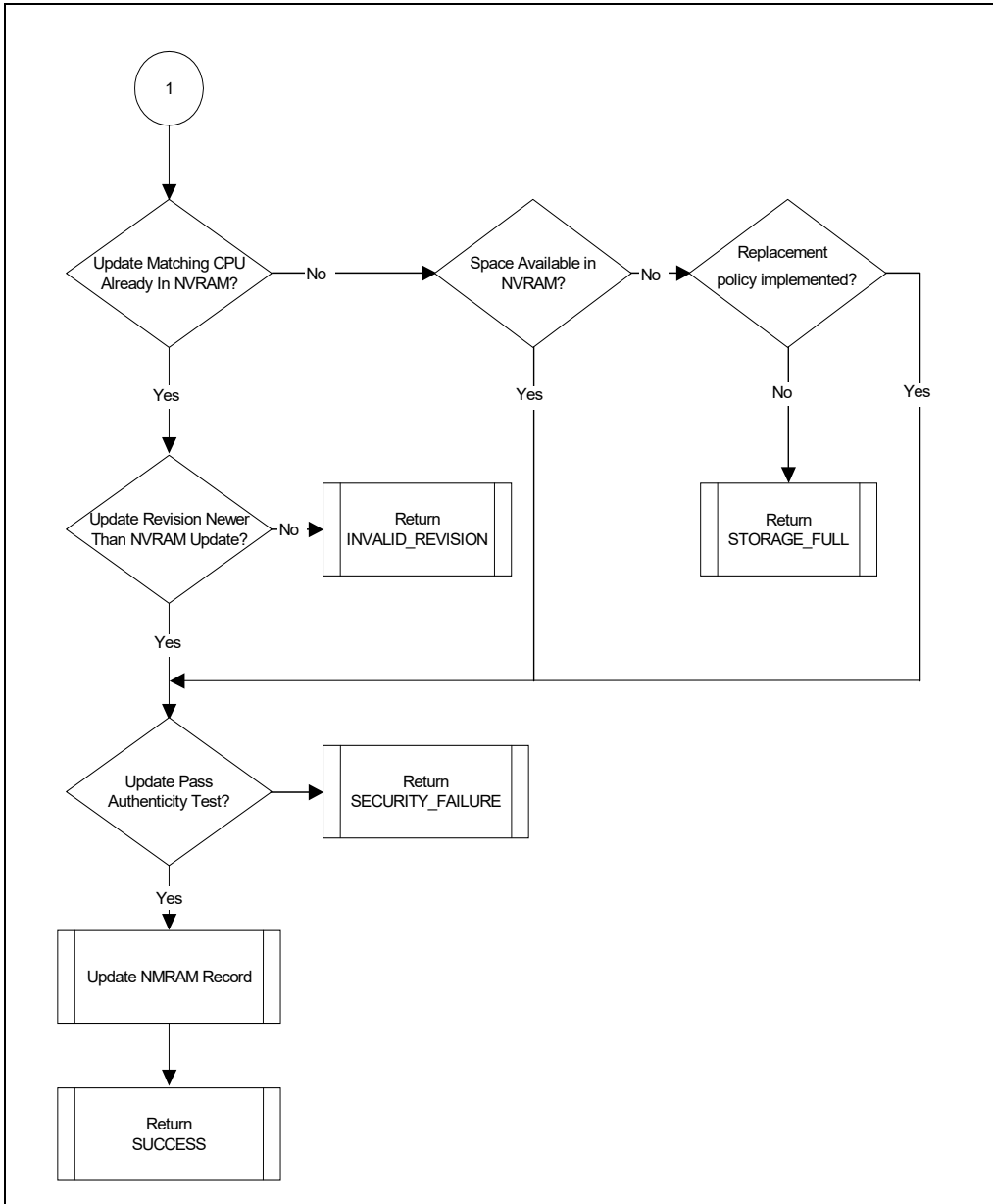


Figure 10-9. Microcode Update Write Operation Flow [2]

10.11.8.7 Function 02H—Microcode Update Control

This function enables loading of binary updates into the processor. Table 10-16 lists the parameters and return codes for the function.

Table 10-16. Parameters for the Control Update Sub-function

Input		
AX	Function Code	0D042H
BL	Sub-function	02H - Control update
BH	Task	See the description below.
CX	Scratch Pad1	Real mode segment of 64 KBytes of RAM block
DX	Scratch Pad2	Real mode segment of 64 KBytes of RAM block
SI	Scratch Pad3	Real mode segment of 64 KBytes of RAM block
SS:SP	Stack pointer	32 kilobytes of stack minimum
Output		
CF	Carry Flag	Carry Set - Failure - AH contains status Carry Clear - All return values valid.
AH	Return Code	Status of the call
AL	OEM Error	Additional OEM Information.
BL	Update Status	Either enable or disable indicator
Return Codes (see Table 10-19 for code definitions)		
SUCCESS		Function completed successfully.
READ_FAILURE		A failure occurred because of the inability to read the storage device.

This control is provided on a global basis for all updates and processors. The caller can determine the current status of update loading (enabled or disabled) without changing the state. The function does not allow the caller to disable loading of binary updates, as this poses a security risk.

The caller specifies the requested operation by placing one of the values from Table 10-17 in the BH register. After successfully completing this function, the BL register contains either the enable or the disable designator. Note that if the function fails, the update status return value is undefined.

Table 10-17. Mnemonic Values

Mnemonic	Value	Meaning
Enable	1	Enable the Update loading at initialization time.
Query	2	Determine the current state of the update control without changing its status.

The READ_FAILURE error code returned by this function has meaning only if the control function is implemented in the BIOS NVRAM. The state of this feature (enabled/disabled) can also be implemented using CMOS RAM bits where READ failure errors cannot occur.

10.11.8.8 Function 03H—Read Microcode Update Data

This function reads a currently installed microcode update from the BIOS storage into a caller-provided RAM buffer. Table 10-18 lists the parameters and return codes.

Table 10-18. Parameters for the Read Microcode Update Data Function

Input		
AX	Function Code	0D042H
BL	Sub-function	03H - Read Update
ES:DI	Buffer Address	Real Mode pointer to the Intel Update structure that will be written with the binary data

Table 10-18. Parameters for the Read Microcode Update Data Function (Contd.)

ECX	Scratch Pad1	Real Mode Segment address of 64 KBytes of RAM Block (lower 16 bits)
ECX	Scratch Pad2	Real Mode Segment address of 64 KBytes of RAM Block (upper 16 bits)
DX	Scratch Pad3	Real Mode Segment address of 64 KBytes of RAM Block
SS:SP	Stack pointer	32 KBytes of Stack Minimum
SI	Update Number	This is the index number of the update block to be read. This value is zero based and must be less than the update count returned from the presence test function.
Output		
CF	Carry Flag	Carry Set - Failure - AH contains Status
Carry Clear - All return values are valid.		
AH	Return Code	Status of the Call
AL	OEM Error	Additional OEM Information
Return Codes (see Table 10-19 for code definitions)		
SUCCESS		The function completed successfully.
READ_FAILURE		There was a failure because of the inability to read the storage device.
UPDATE_NUM_INVALID		Update number exceeds the maximum number of update blocks implemented by the BIOS.
NOT_EMPTY		The specified update block is a subsequent block in use to store a valid microcode update that spans multiple blocks. The specified block is not a header block and is not empty.

The read function enables the caller to read any microcode update data that already exists in a BIOS and make decisions about the addition of new updates. As a result of a successful call, the BIOS copies the microcode update into the location pointed to by ES:DI, with the contents of all Update block(s) that are used to store the specified microcode update.

If the specified block is not a header block, but does contain valid data from a microcode update that spans multiple update blocks, then the BIOS must return Failure with the NOT_EMPTY error code in AH.

An update block is considered unused and available for storing a new update if its Header Version contains the value 0FFFFFFFH after return from this function call. The actual implementation of NVRAM storage management is not specified here and is BIOS dependent. As an example, the actual data value used to represent an empty block by the BIOS may be zero, rather than 0FFFFFFFH. The BIOS is responsible for translating this information into the header provided by this function.

10.11.8.9 Return Codes

After the call has been made, the return codes listed in Table 10-19 are available in the AH register.

Table 10-19. Return Code Definitions

Return Code	Value	Description
SUCCESS	00H	The function completed successfully.
NOT_IMPLEMENTED	86H	The function is not implemented.
ERASE_FAILURE	90H	A failure because of the inability to erase the storage device.
WRITE_FAILURE	91H	A failure because of the inability to write the storage device.
READ_FAILURE	92H	A failure because of the inability to read the storage device.
STORAGE_FULL	93H	The BIOS non-volatile storage area is unable to accommodate the update because all available update blocks are filled with updates that are needed for processors in the system.
CPU_NOT_PRESENT	94H	The processor stepping does not currently exist in the system.
INVALID_HEADER	95H	The update header contains a header or loader version that is not recognized by the BIOS.
INVALID_HEADER_CS	96H	The update does not checksum correctly.
SECURITY_FAILURE	97H	The update was rejected by the processor.
INVALID_REVISION	98H	The same or more recent revision of the update exists in the storage device.
UPDATE_NUM_INVALID	99H	The update number exceeds the maximum number of update blocks implemented by the BIOS.
NOT_EMPTY	9AH	The specified update block is a subsequent block in use to store a valid microcode update that spans multiple blocks. The specified block is not a header block and is not empty.

8. Updates to Chapter 15, Volume 3B

Change bars and green text show changes to Chapter 15 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- Update to Figure 15-2, "IA32_PERF_CTL Register," to correct the name of bit 32.

This chapter describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

15.1 ENHANCED INTEL SPEEDSTEP® TECHNOLOGY

Enhanced Intel SpeedStep® Technology was introduced in the Pentium M processor. The technology enables the management of processor power consumption via performance state transitions. These states are defined as discrete operating points associated with different voltages and frequencies.

Enhanced Intel SpeedStep Technology differs from previous generations of Intel SpeedStep® Technology in two ways:

- Centralization of the control mechanism and software interface in the processor by using model-specific registers.
- Reduced hardware overhead; this permits more frequent performance state transitions.

Previous generations of the Intel SpeedStep Technology require processors to be a deep sleep state, holding off bus master transfers for the duration of a performance state transition. Performance state transitions under the Enhanced Intel SpeedStep Technology are discrete transitions to a new target frequency.

Support is indicated by CPUID, using ECX feature bit 07. Enhanced Intel SpeedStep Technology is enabled by setting IA32_MISC_ENABLE MSR, bit 16. On reset, bit 16 of IA32_MISC_ENABLE MSR is cleared.

15.1.1 Software Interface For Initiating Performance State Transitions

State transitions are initiated by writing a 16-bit value to the IA32_PERF_CTL register, see Figure 15-2. If a transition is already in progress, transition to a new value will subsequently take effect.

Reads of IA32_PERF_CTL determine the last targeted operating point. The current operating point can be read from IA32_PERF_STATUS. IA32_PERF_STATUS is updated dynamically.

The 16-bit encoding that defines valid operating points is model-specific. Applications and performance tools are not expected to use either IA32_PERF_CTL or IA32_PERF_STATUS and should treat both as reserved. Performance monitoring tools can access model-specific events and report the occurrences of state transitions.

15.2 P-STATE HARDWARE COORDINATION

The Advanced Configuration and Power Interface (ACPI) defines performance states (P-states) that are used to facilitate system software's ability to manage processor power consumption. Different P-states correspond to different performance levels that are applied while the processor is actively executing instructions. Enhanced Intel SpeedStep Technology supports P-states by providing software interfaces that control the operating frequency and voltage of a processor.

With multiple processor cores residing in the same physical package, hardware dependencies may exist for a subset of logical processors on a platform. These dependencies may impose requirements that impact the coordination of P-state transitions. As a result, multi-core processors may require an OS to provide additional software support for coordinating P-state transitions for those subsets of logical processors.

ACPI firmware can choose to expose P-states as dependent and hardware-coordinated to OS power management (OSPM) policy. To support OSPMs, multi-core processors must have additional built-in support for P-state hardware coordination and feedback.

Intel 64 and IA-32 processors with dependent P-states amongst a subset of logical processors permit hardware coordination of P-states and provide a hardware-coordination feedback mechanism using IA32_MPERF MSR and

IA32_APERF MSR. See Figure 15-1 for an overview of the two 64-bit MSRs and the bullets below for a detailed description.

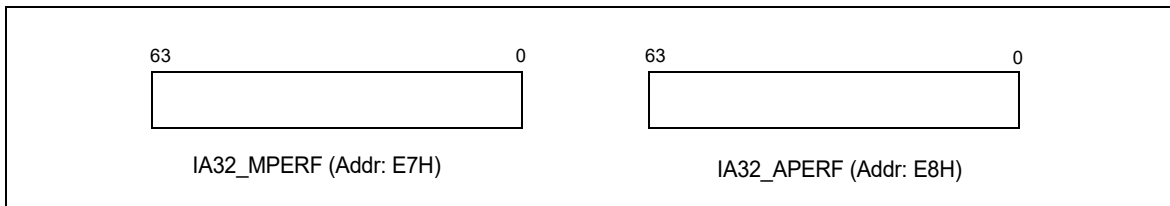


Figure 15-1. IA32_MPERF MSR and IA32_APERF MSR for P-state Coordination

- Use CPUID to check the P-State hardware coordination feedback capability bit. CPUID.06H.ECX[Bit 0] = 1 indicates IA32_MPERF MSR and IA32_APERF MSR are present.
- IA32_MPERF MSR (E7H) increments in proportion to a fixed frequency, which is configured when the processor is booted.
- IA32_APERF MSR (E8H) increments in proportion to actual performance, while accounting for hardware coordination of P-state and TM1/TM2; or software initiated throttling.
- The MSRs are per logical processor; they measure performance only when the targeted processor is in the C0 state.
- Only the IA32_APERF/IA32_MPERF ratio is architecturally defined; software should not attach meaning to the content of the individual of IA32_APERF or IA32_MPERF MSRs.
- When either MSR overflows, both MSRs are reset to zero and continue to increment.
- Both MSRs are full 64-bits counters. Each MSR can be written to independently. However, software should follow the guidelines illustrated in Example 15-1.

If P-states are exposed by the BIOS as hardware coordinated, software is expected to confirm processor support for P-state hardware coordination feedback and use the feedback mechanism to make P-state decisions. The OSPM is expected to either save away the current MSR values (for determination of the delta of the counter ratio at a later time) or reset both MSRs (execute WRMSR with 0 to these MSRs individually) at the start of the time window used for making the P-state decision. When not resetting the values, overflow of the MSRs can be detected by checking whether the new values read are less than the previously saved values.

Example 15-1 demonstrates steps for using the hardware feedback mechanism provided by IA32_APERF MSR and IA32_MPERF MSR to determine a target P-state.

Example 15-1. Determine Target P-state From Hardware Coordinated Feedback

```

DWORD PercentBusy; // Percentage of processor time not idle.
// Measure "PercentBusy" during previous sampling window.
// Typically, "PercentBusy" is measure over a time scale suitable for
// power management decisions
//
// RDMSR of MCNT and ACNT should be performed without delay.
// Software needs to exercise care to avoid delays between
// the two RDMSRs (for example, interrupts).
MCNT = RDMSR(IA32_MPERF);
ACNT = RDMSR(IA32_APERF);

// PercentPerformance indicates the percentage of the processor
// that is in use. The calculation is based on the PercentBusy,
// that is the percentage of processor time not idle and the P-state
// hardware coordinated feedback using the ACNT/MCNT ratio.
// Note that both values need to be calculated over the same
    
```

```

// time window.
    PercentPerformance = PercentBusy * (ACNT/MCNT);

// This example does not cover the additional logic or algorithms
// necessary to coordinate multiple logical processors to a target P-state.

TargetPstate = FindPstate(PercentPerformance);

if (TargetPstate ≠ currentPstate) {
    SetPState(TargetPstate);
}
// WRMSR of MCNT and ACNT should be performed without delay.
// Software needs to exercise care to avoid delays between
// the two WRMSRs (for example, interrupts).
WRMSR(IA32_MPERF, 0);
WRMSR(IA32_APERF, 0);

```

15.3 SYSTEM SOFTWARE CONSIDERATIONS AND OPPORTUNISTIC PROCESSOR PERFORMANCE OPERATION

An Intel 64 processor may support a form of processor operation that takes advantage of design headroom to opportunistically increase performance. The Intel® Turbo Boost Technology can convert thermal headroom into higher performance across multi-threaded and single-threaded workloads. The Intel® Dynamic Acceleration Technology feature can convert thermal headroom into higher performance if only one thread is active.

15.3.1 Intel® Dynamic Acceleration Technology

The Intel Core 2 Duo processor T 7700 introduces Intel Dynamic Acceleration Technology. Intel Dynamic Acceleration Technology takes advantage of thermal design headroom and opportunistically allows a single core to operate at a higher performance level when the operating system requests increased performance.

15.3.2 System Software Interfaces for Opportunistic Processor Performance Operation

Opportunistic processor performance operation, applicable to Intel Dynamic Acceleration Technology and Intel® Turbo Boost Technology, has the following characteristics:

- A transition from a normal state of operation (e.g., Intel Dynamic Acceleration Technology/Turbo mode disengaged) to a target state is not guaranteed, but may occur opportunistically after the corresponding enable mechanism is activated, the headroom is available and certain criteria are met.
- The opportunistic processor performance operation is generally transparent to most application software.
- System software (BIOS and Operating system) must be aware of hardware support for opportunistic processor performance operation and may need to temporarily disengage opportunistic processor performance operation when it requires more predictable processor operation.
- When opportunistic processor performance operation is engaged, the OS should use hardware coordination feedback mechanisms to prevent un-intended policy effects if it is activated during inappropriate situations.

15.3.2.1 Discover Hardware Support and Enabling of Opportunistic Processor Performance Operation

If an Intel 64 processor has hardware support for opportunistic processor performance operation, the power-on default state of IA32_MISC_ENABLE[38] indicates the presence of such hardware support. For Intel 64 processors that support opportunistic processor performance operation, the default value is 1, indicating its presence. For processors that do not support opportunistic processor performance operation, the default value is 0. The power-

on default value of IA32_MISC_ENABLE[38] allows BIOS to detect the presence of hardware support of opportunistic processor performance operation.

IA32_MISC_ENABLE[38] is shared across all logical processors in a physical package. It is written by BIOS during platform initiation to enable/disable opportunistic processor performance operation in conjunction of OS power management capabilities, see Section 15.3.2.2. BIOS can set IA32_MISC_ENABLE[38] with 1 to disable opportunistic processor performance operation; it must clear the default value of IA32_MISC_ENABLE[38] to 0 to enable opportunistic processor performance operation. OS and applications must use CPUID leaf 06H if it needs to detect processors that have opportunistic processor performance operation enabled.

When CPUID is executed with EAX = 06H on input, Bit 1 of EAX in Leaf 06H (i.e., CPUID.06H:EAX[1]) indicates opportunistic processor performance operation, such as Intel Dynamic Acceleration Technology, has been enabled by BIOS.

Opportunistic processor performance operation can be disabled by setting bit 38 of IA32_MISC_ENABLE. This mechanism is intended for BIOS only. If IA32_MISC_ENABLE[38] is set, CPUID.06H:EAX[1] will return 0.

15.3.2.2 OS Control of Opportunistic Processor Performance Operation

There may be phases of software execution in which system software cannot tolerate the non-deterministic aspects of opportunistic processor performance operation. For example, when calibrating a real-time workload to make a CPU reservation request to the OS, it may be undesirable to allow the possibility of the processor delivering increased performance that cannot be sustained after the calibration phase.

System software can temporarily disengage opportunistic processor performance operation by setting bit 32 of the IA32_PERF_CTL MSR (0199H), using a read-modify-write sequence on the MSR. The opportunistic processor performance operation can be re-engaged by clearing bit 32 in IA32_PERF_CTL MSR, using a read-modify-write sequence. The DISENAGE bit in IA32_PERF_CTL is not reflected in bit 32 of the IA32_PERF_STATUS MSR (0198H), and it is not shared between logical processors in a physical package. In order for OS to engage Intel Dynamic Acceleration Technology/Turbo mode, the BIOS must:

- Enable opportunistic processor performance operation, as described in Section 15.3.2.1.
- Expose the operating points associated with Intel Dynamic Acceleration Technology/Turbo mode to the OS.

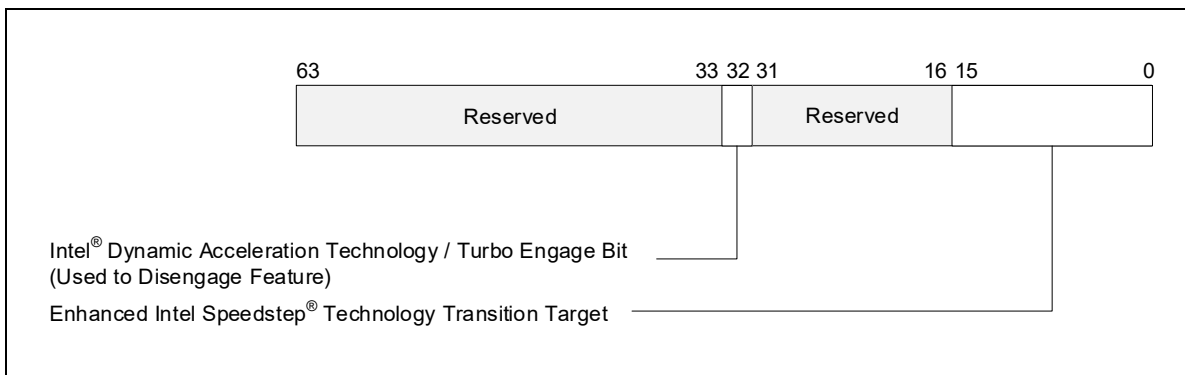


Figure 15-2. IA32_PERF_CTL Register

15.3.2.3 Required Changes to OS Power Management P-State Policy

Intel Dynamic Acceleration Technology and Intel Turbo Boost Technology can provide opportunistic performance greater than the performance level corresponding to the Processor Base frequency of the processor (see CPUID's processor frequency information). System software can use a pair of MSRs to observe performance feedback. Software must query for the presence of IA32_APERF and IA32_MPERF (see Section 15.2). The ratio between IA32_APERF and IA32_MPERF is architecturally defined and a value greater than unity indicates performance increase occurred during the observation period due to Intel Dynamic Acceleration Technology. Without incorporating such performance feedback, the target P-state evaluation algorithm can result in a non-optimal P-state target.

There are other scenarios under which OS power management may want to disable Intel Dynamic Acceleration Technology, some of these are listed below:

- When engaging ACPI defined passive thermal management, it may be more effective to disable Intel Dynamic Acceleration Technology for the duration of passive thermal management.
- When the user has indicated a policy preference of power savings over performance, OS power management may want to disable Intel Dynamic Acceleration Technology while that policy is in effect.

15.3.3 Intel® Turbo Boost Technology

Intel Turbo Boost Technology is supported in Intel Core i7 processors and Intel Xeon processors based on Nehalem microarchitecture. It uses the same principle of leveraging thermal headroom to dynamically increase processor performance for single-threaded and multi-threaded/multi-tasking environment. The programming interface described in Section 15.3.2 also applies to Intel Turbo Boost Technology.

15.3.4 Performance and Energy Bias Hint Support

Intel 64 processors may support additional software hint to guide the hardware heuristic of power management features to favor increasing dynamic performance or conserve energy consumption.

Software can detect the processor's capability to support the performance-energy bias preference hint by examining bit 3 of ECX in CPUID leaf 6. The processor supports this capability if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H).

Software can program the lowest four bits of IA32_ENERGY_PERF_BIAS MSR with a value from 0 - 15. The values represent a sliding scale, where a value of 0 (the default reset value) corresponds to a hint preference for highest performance and a value of 15 corresponds to the maximum energy savings. A value of 7 roughly translates into a hint to balance performance with energy consumption.

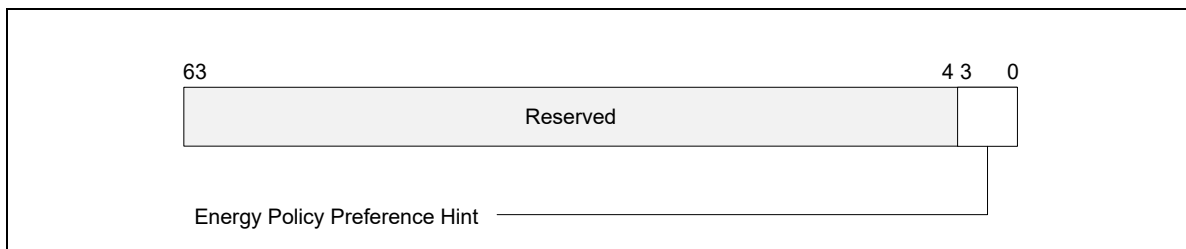


Figure 15-3. IA32_ENERGY_PERF_BIAS Register

The layout of IA32_ENERGY_PERF_BIAS is shown in Figure 15-3. The scope of IA32_ENERGY_PERF_BIAS is per logical processor, which means that each of the logical processors in the package can be programmed with a different value. This may be especially important in virtualization scenarios, where the performance / energy requirements of one logical processor may differ from the other. Conflicting "hints" from various logical processors at higher hierarchy level will be resolved in favor of performance over energy savings.

Software can use whatever criteria it sees fit to program the MSR with an appropriate value. However, the value only serves as a hint to the hardware and the actual impact on performance and energy savings is model specific.

15.4 HARDWARE-CONTROLLED PERFORMANCE STATES (HWP)

Intel processors may contain support for Hardware-Controlled Performance States (HWP), which autonomously selects performance states while utilizing OS supplied performance guidance hints. The Enhanced Intel Speed-Step® Technology provides a means for the OS to control and monitor discrete frequency-based operating points via the IA32_PERF_CTL and IA32_PERF_STATUS MSRs.

In contrast, HWP is an implementation of the ACPI-defined Collaborative Processor Performance Control (CPPC), which specifies that the platform enumerates a continuous, abstract unit-less, performance value scale that is not tied to a specific performance state / frequency by definition. While the enumerated scale is roughly linear in terms of a delivered integer workload performance result, the OS is required to characterize the performance value range to comprehend the delivered performance for an applied workload.

When HWP is enabled, the processor autonomously selects performance states as deemed appropriate for the applied workload and with consideration of constraining hints that are programmed by the OS. These OS-provided hints include minimum and maximum performance limits, preference towards energy efficiency or performance, and the specification of a relevant workload history observation time window. The means for the OS to override HWP's autonomous selection of performance state with a specific desired performance target is also provided, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations.

15.4.1 HWP Programming Interfaces

The programming interfaces provided by HWP include the following:

- The CPUID instruction allows software to discover the presence of HWP support in an Intel processor. Specifically, execute CPUID instruction with EAX=06H as input will return 5 bit flags covering the following aspects in bits 7 through 11 of CPUID.06H:EAX:
 - Availability of HWP baseline resource and capability, CPUID.06H:EAX[bit 7]: If this bit is set, HWP provides several new architectural MSRs: IA32_PM_ENABLE, IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS.
 - Availability of HWP Notification upon dynamic Guaranteed Performance change, CPUID.06H:EAX[bit 8]: If this bit is set, HWP provides IA32_HWP_INTERRUPT MSR to enable interrupt generation due to dynamic Performance changes and excursions.
 - Availability of HWP Activity window control, CPUID.06H:EAX[bit 9]: If this bit is set, HWP allows software to program activity window in the IA32_HWP_REQUEST MSR.
 - Availability of HWP energy/performance preference control, CPUID.06H:EAX[bit 10]: If this bit is set, HWP allows software to set an energy/performance preference hint in the IA32_HWP_REQUEST MSR.
 - Availability of HWP package level control, CPUID.06H:EAX[bit 11]: If this bit is set, HWP provides the IA32_HWP_REQUEST_PKG MSR to convey OS Power Management's control hints for all logical processors in the physical package.

Table 15-1. Architectural and Non-Architectural MSRs Related to HWP

Address	Architectural	Register Name	Description
770H	Y	IA32_PM_ENABLE	Enable/Disable HWP.
771H	Y	IA32_HWP_CAPABILITIES	Enumerates the HWP performance range (static and dynamic).
772H	Y	IA32_HWP_REQUEST_PKG	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for all logical processor in the physical package.
773H	Y	IA32_HWP_INTERRUPT	Controls HWP native interrupt generation (Guaranteed Performance changes, excursions).
774H	Y	IA32_HWP_REQUEST	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for a single logical processor.
775H	Y	IA32_HWP_PECI_REQUEST_INFO	Conveys embedded system controller requests to override some of the OS HWP Request settings via the PECI mechanism.
777H	Y	IA32_HWP_STATUS	Status bits indicating changes to Guaranteed Performance and excursions to Minimum Performance.
19CH	Y	IA32_THERM_STATUS[bits 15:12]	Conveys reasons for performance excursions.
64EH	N	MSR_PPERF	Productive Performance Count.

- Additionally, HWP may provide a non-architectural MSR, MSR_PPERF, which provides a quantitative metric to software of hardware's view of workload scalability. This hardware's view of workload scalability is implementation specific.

15.4.2 Enabling HWP

The layout of the IA32_PM_ENABLE MSR is shown in Figure 15-4. The bit fields are described below:

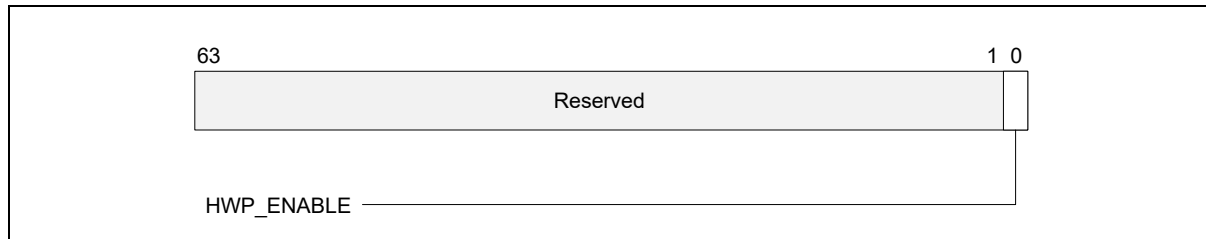


Figure 15-4. IA32_PM_ENABLE MSR

- **HWP_ENABLE (bit 0, R/W1Once)** — Software sets this bit to enable HWP with autonomous selection of processor P-States. When set, the processor will disregard input from the legacy performance control interface (IA32_PERF_CTL). Note this bit can only be enabled once from the default value. Once set, writes to the HWP_ENABLE bit are ignored. Only RESET will clear this bit. Default = zero (0).
- Bits 63:1 are reserved and must be zero.

After software queries CPUID and verifies the processor's support of HWP, system software can write 1 to IA32_PM_ENABLE.HWP_ENABLE (bit 0) to enable hardware controlled performance states. The default value of IA32_PM_ENABLE MSR at power-on is 0, i.e., HWP is disabled.

Additional MSRs associated with HWP may only be accessed after HWP is enabled, with the exception of IA32_HWP_INTERRUPT and MSR_PPERF. Accessing the IA32_HWP_INTERRUPT MSR requires only HWP is present as enumerated by CPUID but does not require enabling HWP.

IA32_PM_ENABLE is a package level MSR, i.e., writing to it from any logical processor within a package affects all logical processors within that package.

15.4.3 HWP Performance Range and Dynamic Capabilities

The OS reads the IA32_HWP_CAPABILITIES MSR to comprehend the limits of the HWP-managed performance range as well as the dynamic capability, which may change during processor operation. The enumerated performance range values reported by IA32_HWP_CAPABILITIES directly map to initial frequency targets (prior to workload-specific frequency optimizations of HWP). However the mapping is processor family specific.

The layout of the IA32_HWP_CAPABILITIES MSR is shown in Figure 15-5. The bit fields are described below:

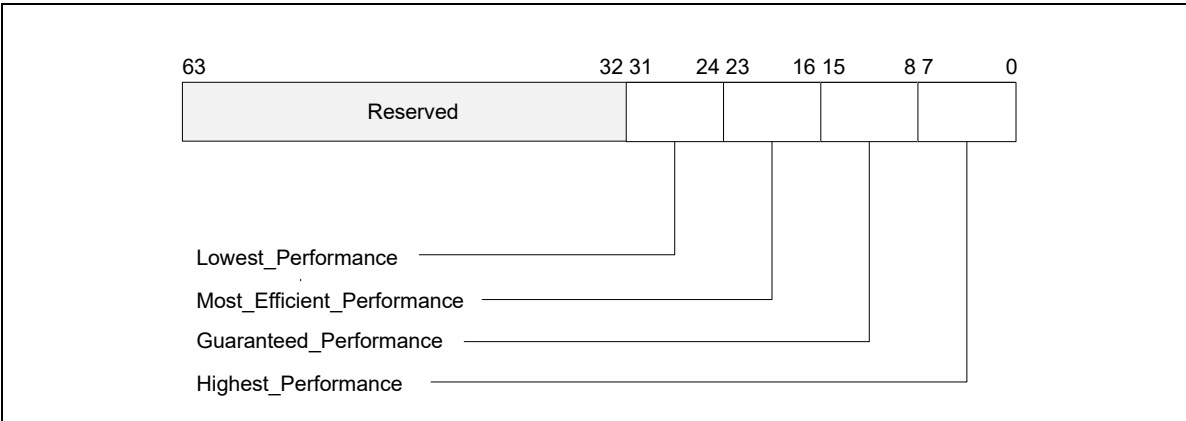


Figure 15-5. IA32_HWP_CAPABILITIES Register

- **Highest_Performance (bits 7:0, RO)** — Value for the maximum non-guaranteed performance level.
- **Guaranteed_Performance (bits 15:8, RO)** — Current value for the guaranteed performance level. This value can change dynamically as a result of internal or external constraints, e.g., thermal or power limits.
- **Most_Efficient_Performance (bits 23:16, RO)** — Current value of the most efficient performance level. This value can change dynamically as a result of workload characteristics.
- **Lowest_Performance (bits 31:24, RO)** — Value for the lowest performance level that software can program to IA32_HWP_REQUEST.
- Bits 63:32 are reserved and must be zero.

The value returned in the **Guaranteed_Performance** field is hardware's best-effort approximation of the available performance given current operating constraints. Changes to the **Guaranteed_Performance** value will primarily occur due to a shift in operational mode. This includes a power or other limit applied by an external agent, e.g., RAPL (see Figure 15.10.1), or the setting of a Configurable TDP level (see model-specific controls related to Programmable TDP Limit in Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.). Notification of a change to the **Guaranteed_Performance** occurs via interrupt (if configured) and the IA32_HWP_Status MSR. Changes to **Guaranteed_Performance** are indicated when a macroscopically meaningful change in performance occurs i.e., sustained for greater than one second. Consequently, notification of a change in **Guaranteed_Performance** will typically occur no more frequently than once per second. Rapid changes in platform configuration, e.g., docking/undocking, with corresponding changes to a Configurable TDP level could potentially cause more frequent notifications.

The value returned by the **Most_Efficient_Performance** field provides the OS with an indication of the practical lower limit for the IA32_HWP_REQUEST. The processor may not honor IA32_HWP_REQUEST.Maximum Performance settings below this value.

15.4.4 Managing HWP

15.4.4.1 IA32_HWP_REQUEST MSR (Address: 774H Logical Processor Scope)

Typically, the operating system controls HWP operation for each logical processor via the writing of control hints / constraints to the IA32_HWP_REQUEST MSR. The layout of the IA32_HWP_REQUEST MSR is shown in Figure 15-6. The bit fields are described below Figure 15-6.

Operating systems can control HWP by writing both IA32_HWP_REQUEST and IA32_HWP_REQUEST_PKG MSRs (see Section 15.4.4.2). Five valid bits within the IA32_HWP_REQUEST MSR let the operating system flexibly select which of its five hint / constraint fields should be derived by the processor from the IA32_HWP_REQUEST MSR and which should be derived from the IA32_HWP_REQUEST_PKG MSR. These five valid bits are supported if CPUID[6].EAX[17] is set.

When the IA32_HWP_REQUEST MSR Package Control bit is set, any valid bit that is NOT set indicates to the processor to use the respective field value from the IA32_HWP_REQUEST_PKG MSR. Otherwise, the values are derived from the IA32_HWP_REQUEST MSR. The valid bits are ignored when the IA32_HWP_REQUEST MSR Package Control bit is zero.

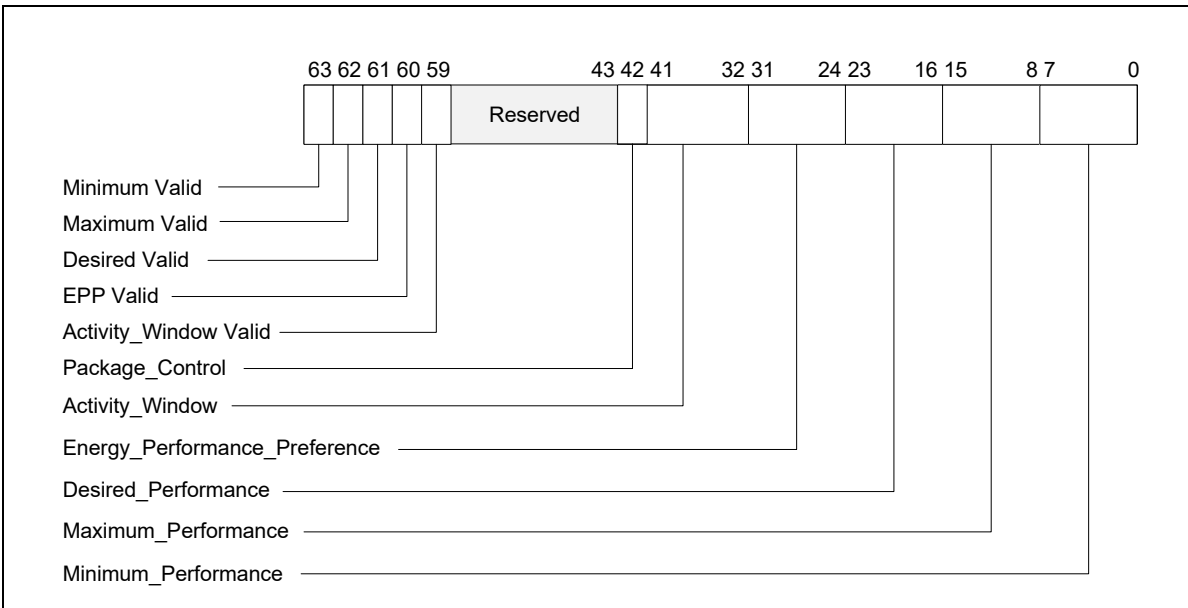


Figure 15-6. IA32_HWP_REQUEST Register

- **Minimum_Performance (bits 7:0, RW)** — Conveys a hint to the HWP hardware. The OS programs the minimum performance hint to achieve the required quality of service (QoS) or to meet a service level agreement (SLA) as needed. Note that an excursion below the level specified is possible due to hardware constraints. The default value of this field is IA32_HWP_CAPABILITIES.Lowest_Performance.
- **Maximum_Performance (bits 15:8, RW)** — Conveys a hint to the HWP hardware. The OS programs this field to limit the maximum performance that is expected to be supplied by the HWP hardware. Excursions above the limit requested by OS are possible due to hardware coordination between the processor cores and other components in the package. The default value of this field is IA32_HWP_CAPABILITIES.Highest_Performance.
- **Desired_Performance (bits 23:16, RW)** — Conveys a hint to the HWP hardware. When set to zero, hardware autonomous selection determines the performance target. When set to a non-zero value (between the range of Lowest_Performance and Highest_Performance of IA32_HWP_CAPABILITIES) conveys an explicit performance request hint to the hardware; effectively disabling HW Autonomous selection. The Desired_Performance input is non-constraining in terms of Performance and Energy Efficiency optimizations, which are independently controlled. The default value of this field is 0.
- **Energy_Performance_Preference (bits 31:24, RW)** — Conveys a hint to the HWP hardware. The OS may write a range of values from 0 (performance preference) to 0FFH (energy efficiency preference) to influence the rate of performance increase /decrease and the result of the hardware's energy efficiency and performance optimizations. The default value of this field is 80H. Note: If CPUID.06H:EAX[bit 10] indicates that this field is not supported, HWP uses the value of the IA32_ENERGY_PERF_BIAS MSR to determine the energy efficiency / performance preference.
- **Activity_Window (bits 41:32, RW)** — Conveys a hint to the HWP hardware specifying a moving workload history observation window for performance/frequency optimizations. If 0, the hardware will determine the appropriate window size. When writing a non-zero value to this field, this field is encoded in the format of bits 38:32 as a 7-bit mantissa and bits 41:39 as a 3-bit exponent value in powers of 10. The resultant value is in microseconds. Thus, the minimal/maximum activity window size is 1 microsecond/1270 seconds. Combined with the Energy_Performance_Preference input, Activity_Window influences the rate of performance increase

/ decrease. This non-zero hint only has meaning when Desired_Performance = 0. The default value of this field is 0.

- **Package_Control (bit 42, RW)** — When set, causes this logical processor's IA32_HWP_REQUEST control inputs to be derived from the IA32_HWP_REQUEST_PKG MSR.
- Bits 58:43 are reserved and must be zero.
- **Activity_Window_Valid (bit 59, RW)** — When set, indicates to the processor to derive the Activity Window field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **EPP_Valid (bit 60, RW)** — When set, indicates to the processor to derive the EPP field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Desired_Valid (bit 61, RW)** — When set, indicates to the processor to derive the Desired Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Maximum_Valid (bit 62, RW)** — When set, indicates to the processor to derive the Maximum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Minimum_Valid (bit 63, RW)** — When set, indicates to the processor to derive the Minimum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.

The HWP hardware clips and resolves the field values as necessary to the valid range. Reads return the last value written not the clipped values.

Processors may support a subset of IA32_HWP_REQUEST fields as indicated by CPUID. Reads of non-supported fields will return 0. Writes to non-supported fields are ignored.

The OS may override HWP's autonomous selection of performance state with a specific performance target by setting the Desired_Performance field to a non-zero value, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations, which are influenced by the Energy Performance Preference field.

Software may disable all hardware optimizations by setting Minimum_Performance = Maximum_Performance (subject to package coordination).

Note: The processor may run below the Minimum_Performance level due to hardware constraints including: power, thermal, and package coordination constraints. The processor may also run below the Minimum_Performance level for short durations (few milliseconds) following C-state exit, and when Hardware Duty Cycling (see Section 15.5) is enabled.

When the IA32_HWP_REQUEST MSR is set to fast access mode, writes of this MSR are posted, i.e., the WRMSR instruction retires before the data reaches its destination within the processor. It may retire even before all preceding IA stores are globally visible, i.e., it is not an architecturally serializing instruction anymore (no store fence). A new CPUID bit indicates this new characteristic of the IA32_HWP_REQUEST MSR (see Section 15.4.8 for additional details).

15.4.4.2 IA32_HWP_REQUEST_PKG MSR (Address: 772H Package Scope)

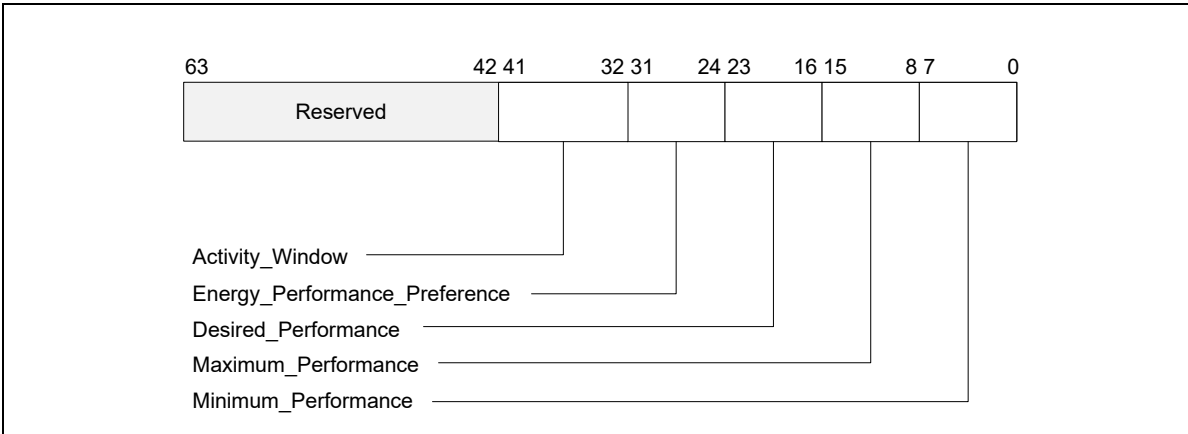


Figure 15-7. IA32_HWP_REQUEST_PKG Register

The structure of the IA32_HWP_REQUEST_PKG MSR (package-level) is identical to the IA32_HWP_REQUEST MSR with the exception of the the Package Control bit field and the five valid bit fields, which do not exist in the IA32_HWP_REQUEST_PKG MSR. Field values written to this MSR apply to all logical processors within the physical package with the exception of logical processors whose IA32_HWP_REQUEST.Package Control field is clear (zero). Single P-state Control mode is only supported when IA32_HWP_REQUEST_PKG is not supported.

15.4.4.3 IA32_HWP_PECI_REQUEST_INFO MSR (Address 775H Package Scope)

When an embedded system controller is integrated in the platform, it can override some of the OS HWP Request settings via the PECI mechanism. PECI initiated settings take precedence over the relevant fields in the IA32_HWP_REQUEST MSR and in the IA32_HWP_REQUEST_PKG MSR, irrespective of the Package Control bit or the Valid Bit values described above. PECI can independently control each of: Minimum Performance, Maximum Performance and EPP fields. This MSR contains both the PECI induced values and the control bits that indicate whether the embedded controller actually set the processor to use the respective value.

PECI override is supported if CPUID[6].EAX[16] is set.

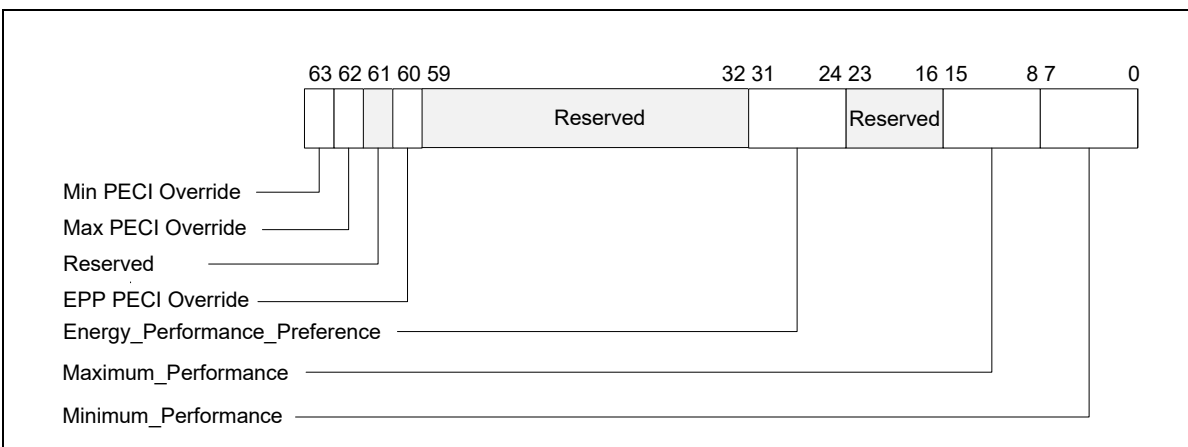


Figure 15-8. IA32_HWP_PECI_REQUEST_INFO MSR

The layout of the IA32_HWP_PECI_REQUEST_INFO MSR is shown in Figure 15-8. This MSR is writable by the embedded controller but is read-only by software executing on the CPU. This MSR has Package scope. The bit fields are described below:

- **Minimum_Performance (bits 7:0, RO)** — Used by the OS to read the latest value of Peci minimum performance input.
- **Maximum_Performance (bits 15:8, RO)** — Used by the OS to read the latest value of Peci maximum performance input.
- Bits 23:16 are reserved and must be zero.
- **Energy_Performance_Preference (bits 31:24, RO)** — Used by the OS to read the latest value of Peci energy performance preference input.
- Bits 59:32 are reserved and must be zero.
- **EPP_PECI_Override (bit 60, RO)** — Indicates whether Peci is currently overriding the Energy Performance Preference input. If set(1), Peci is overriding the Energy Performance Preference input. If clear(0), OS has control over Energy Performance Preference input.
- Bit 61 is reserved and must be zero.
- **Max_PECI_Override (bit 62, RO)** — Indicates whether Peci is currently overriding the Maximum Performance input. If set(1), Peci is overriding the Maximum Performance input. If clear(0), OS has control over Maximum Performance input.
- **Min_PECI_Override (bit 63, RO)** — Indicates whether Peci is currently overriding the Minimum Performance input. If set(1), Peci is overriding the Minimum Performance input. If clear(0), OS has control over Minimum Performance input.

HWP Request Field Hierarchical Resolution

HWP Request field resolution is fed by three MSRs: IA32_HWP_REQUEST, IA32_HWP_REQUEST_PKG, and IA32_HWP_PECI_REQUEST_INFO. The flow that the processor goes through to resolve which field value is chosen is shown below.

For each of the two HWP Request fields; Desired and Activity Window:

```
If IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
    Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
Else
    Resolved Field Value = IA32_HWP_REQUEST.<field>
```

For each of the three HWP Request fields; Min, Max, and EPP:

```
If IA32_HWP_PECI_REQUEST_INFO.<field> Peci Override bit = 1
    Resolved Field Value = IA32_HWP_PECI_REQUEST_INFO.<field>
Else if IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
    Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
Else
    Resolved Field Value = IA32_HWP_REQUEST.<field>
```

15.4.4.4 IA32_HWP_CTL MSR (Address: 776H Logical Processor Scope)

IA32_HWP_CTL[0] controls the behavior of IA32_HWP_REQUEST Package Control [bit 42]. This control bit allows the IA32_HWP_REQUEST MSR to stay in INIT mode most of the time (Control Bit is equal to its RESET value of 0) thus avoiding actual saving/restoring of the MSR contents when the OS adds it to the register set saved and restored by XSAVES/XRSTORS.

- When IA32_HWP_CTL[0] = 0:
 - If IA32_HWP_REQUEST[42] = 0, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
 - If IA32_HWP_REQUEST[42] = 1, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according

to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 15.4.4.1 for additional details.

- When IA32_HWP_CTL[0] = 1, the behavior is reversed:
 - If IA32_HWP_REQUEST[42] = 1, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
 - If IA32_HWP_REQUEST[42] = 0, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 15.4.4.1 for additional details.

Section 15-2 summarizes the IA32_HWP_CTL MSR bit 0 control behavior.

Table 15-2. IA32_HWP_CTL MSR Bit 0 Behavior

Field	Description		
Thread request PKG CTL meaning	Defines which HWP Request MSR is used, whether thread level or package level. When the package MSR is used, the thread MSR valid bits define which thread MSR fields override the package (default 0).		
	IA32_HWP_CTL[PKG_CTL_PLR]	IA32_HWP_REQUEST[PKG_CTL]	HWP Request MSR Used
	0	0	IA32_HWP_REQUEST MSR
	0	1	IA32_HWP_REQUEST_PKG MSR
	1	0	IA32_HWP_REQUEST_PKG MSR
1	1	IA32_HWP_REQUEST MSR	

This MSR is supported if CPUID[6].EAX[22] is set.

If the IA32_PM_ENABLE[HWP_ENABLE] (bit 0) is not set, access to this MSR will generate a #GP fault.

15.4.5 HWP Feedback

The processor provides several types of feedback to the OS during HWP operation.

The IA32_MPERF MSR and IA32_APERF MSR mechanism (see Section 15.2) allows the OS to calculate the resultant effective frequency delivered over a time period. Energy efficiency and performance optimizations directly impact the resultant effective frequency delivered.

The layout of the IA32_HWP_STATUS MSR is shown in Figure 15-9. It provides feedback regarding changes to IA32_HWP_CAPABILITIES.Guaranteed_Performance, IA32_HWP_CAPABILITIES.Highest_Performance, excursions to IA32_HWP_CAPABILITIES.Minimum_Performance, and PECI_Override entry/exit events. The bit fields are described below:

- **Guaranteed_Performance_Change (bit 0, RWC0)** — If set (1), a change to Guaranteed_Performance has occurred. Software should query IA32_HWP_CAPABILITIES.Guaranteed_Performance value to ascertain the new Guaranteed Performance value and to assess whether to re-adjust HWP hints via IA32_HWP_REQUEST. Software must clear this bit by writing a zero (0).
- Bit 1 is reserved and must be zero.
- **Excursion_To_Minimum (bit 2, RWC0)** — If set (1), an excursion to Minimum_Performance of IA32_HWP_REQUEST has occurred. Software must clear this bit by writing a zero (0).
- **Highest_Change (bit 3, RWC0)** — If set (1), a change to Highest Performance has occurred. Software should query IA32_HWP_CAPABILITIES to ascertain the new Highest Performance value. Software must clear this bit by writing a zero (0). Interrupts upon Highest Performance change are supported if CPUID[6].EAX[15] is set.
- **PECI_Override_Entry (bit 4, RWC0)** — If set (1), an embedded/management controller has started a PECI override of one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are now overridden via the PECI mechanism and what their values are (see Section 15.4.4.3 for additional details).

Software must clear this bit by writing a zero (0). Interrupts upon PECI override entry are supported if CPUID[6].EAX[16] is set.

- **PECI_Override_Exit (bit 5, RWC0)** — If set (1), an embedded/management controller has stopped overriding one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are still overridden via the PECI mechanism and which fields are now back under software control (see Section 15.4.4.3 for additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override exit are supported if CPUID[6].EAX[16] is set.
- Bits 63:6 are reserved and must be zero.

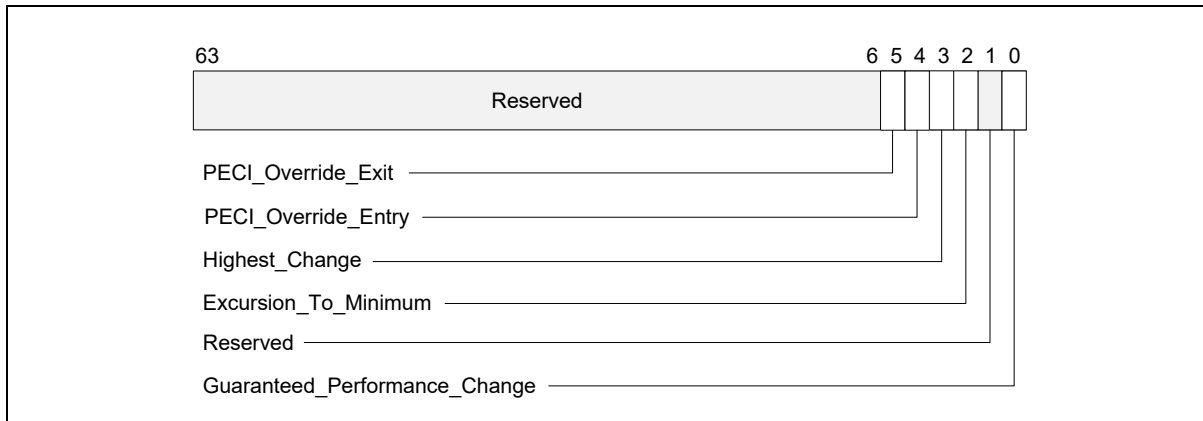


Figure 15-9. IA32_HWP_STATUS MSR

The status bits of IA32_HWP_STATUS must be cleared (0) by software so that a new status condition change will cause the hardware to set the bit again and issue the notification. Status bits are not set for “normal” excursions, e.g., running below Minimum Performance for short durations during C-state exit. Changes to Guaranteed_Performance, Highest_Performance, excursions to Minimum_Performance, or PECI_Override entry/exit will occur no more than once per second.

The OS can determine the specific reasons for a Guaranteed_Performance change or an excursion to Minimum_Performance in IA32_HWP_REQUEST by examining the associated status and log bits reported in the IA32_THERM_STATUS MSR. The layout of the IA32_HWP_STATUS MSR that HWP uses to support software query of HWP feedback is shown in Figure 15-10. The bit fields of IA32_THERM_STATUS associated with HWP feedback are described below (Bit fields of IA32_THERM_STATUS unrelated to HWP can be found in Section 15.8.5.2).

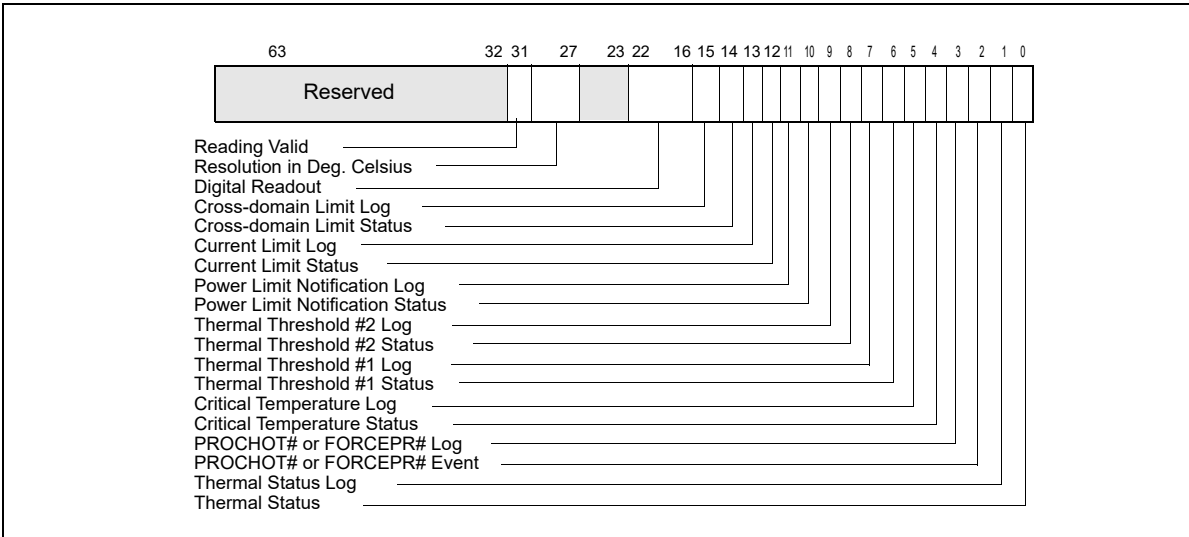


Figure 15-10. IA32_THERM_STATUS Register With HWP Feedback

- Bits 11:0, See Section 15.8.5.2.
- **Current Limit Status (bit 12, RO)** — If set (1), indicates an electrical current limit (e.g., Electrical Design Point/IccMax) is being exceeded and is adversely impacting energy efficiency optimizations.
- **Current Limit Log (bit 13, RWC0)** — If set (1), an electrical current limit has been exceeded that has adversely impacted energy efficiency optimizations since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- **Cross-domain Limit Status (bit 14, RO)** — If set (1), indicates another hardware domain (e.g., processor graphics) is currently limiting energy efficiency optimizations in the processor core domain.
- **Cross-domain Limit Log (bit 15, RWC0)** — If set (1), indicates another hardware domain (e.g., processor graphics) has limited energy efficiency optimizations in the processor core domain since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- Bits 63:16, See Section 15.8.5.2.

15.4.5.1 Non-Architectural HWP Feedback

The Productive Performance (MSR_PPERF) MSR (non-architectural) provides hardware's view of workload scalability, which is a rough assessment of the relationship between frequency and workload performance, to software. The layout of the MSR_PPERF is shown in Figure 15-11.

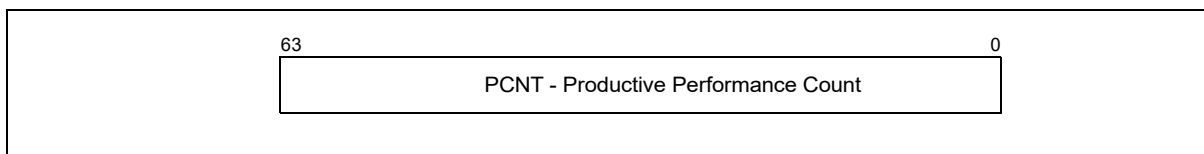


Figure 15-11. MSR_PPERF MSR

- **PCNT (bits 63:0, RO)** — Similar to IA32_APERF but only counts cycles perceived by hardware as contributing to instruction execution (e.g., unhalted and unstalled cycles). This counter increments at the same rate as IA32_APERF, where the ratio of (Δ PCNT/ Δ ACNT) is an indicator of workload scalability (0% to 100%). Note that values in this register are valid even when HWP is not enabled.

15.4.6 HWP Notifications

Processors may support interrupt-based notification of changes to HWP status as indicated by CPUID. If supported, the IA32_HWP_INTERRUPT MSR is used to enable interrupt-based notifications. Notification events, when enabled, are delivered using the existing thermal LVT entry. The layout of the IA32_HWP_INTERRUPT is shown in Figure 15-12. The bit fields are described below:

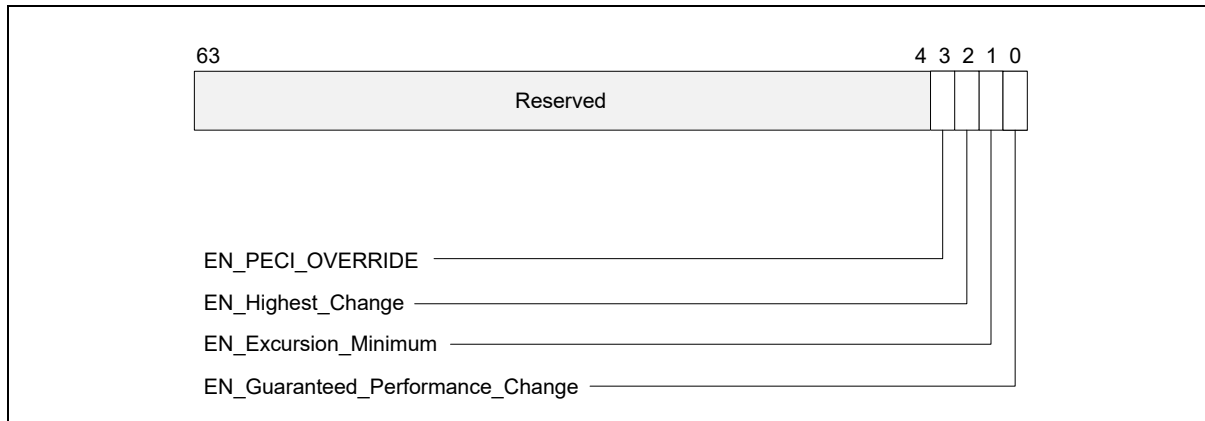


Figure 15-12. IA32_HWP_INTERRUPT MSR

- **EN_Guaranteed_Performance_Change (bit 0, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Guaranteed_Performance occurs. The default value is 0 (Interrupt generation is disabled).
- **EN_Excursion_Minimum (bit 1, RW)** — When set (1), an HWP Interrupt will be generated whenever the HWP hardware is unable to meet the IA32_HWP_REQUEST.Minimum_Performance setting. The default value is 0 (Interrupt generation is disabled).
- **EN_Highest_Change (bit 2, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Highest_Performance occurs. The default value is 0 (interrupt generation is disabled). Interrupts upon Highest Performance change are supported if CPUID[6].EAX[15] is set.
- **EN_PECI_OVERRIDE (bit 3, RW)** — When set (1), an HWP Interrupt will be generated whenever PECI starts or stops overriding any of the three HWP fields described in Section 15.4.4.3. The default value is 0 (interrupt generation is disabled). See Section 15.4.5 and Section 15.4.4.3 for details on how the OS learns what is the current set of HWP fields that are overridden by PECI. Interrupts upon PECI override change are supported if CPUID[6].EAX[16] is set.
- Bits 63:4 are reserved and must be zero.

15.4.7 Idle Logical Processor Impact on Core Frequency

Intel processors use one of two schemes for setting core frequency:

1. All cores share same frequency.
2. Each physical core is set to a frequency of its own.

In both cases the two logical processors that share a single physical core are set to the same frequency, so the processor accounts for the IA32_HWP_REQUEST MSR fields of both logical processors when defining the core frequency or the whole package frequency.

When **CPUID[6].EAX[20]** is set and only one logical processor of the two is active, while the other is idle (in any **C1 sub-state** or in a deeper sleep state), only the **active logical processor's** IA32_HWP_REQUEST MSR fields are considered, i.e., the HWP Request fields of a logical processor in the C1E sub-state or in a deeper sleep state are ignored.

Note: when a logical processor is in **C1 state** its HWP Request fields are accounted for.

15.4.8 Fast Write of Uncore MSR (Model Specific Feature)

There are a few logical processor scope MSRs whose values need to be observed outside the logical processor. The WRMSR instruction takes over 1000 cycles to complete (retire) for those MSRs. This overhead forces operating systems to avoid writing them too often whereas in many cases it is preferable that the OS writes them quite frequently for optimal power/performance operation of the processor.

The model specific “Fast Write MSR” feature reduces this overhead by an order of magnitude to a level of 100 cycles for a selected subset of MSRs.

Note: Writes to Fast Write MSRs are posted, i.e., when the WRMSR instruction completes, the data may still be “in transit” within the processor. Software can check the status by querying the processor to ensure data is already visible outside the logical processor (see Section 15.4.8.3 for additional details). Once the data is visible outside the logical processor, software is ensured that later writes by the same logical processor to the same MSR will be visible later (will not bypass the earlier writes).

MSRs that are selected for Fast Write are specified in a special capability MSR (see Section 15.4.8.1). Architectural MSRs that existed prior to the introduction of this feature and are selected for Fast Write, thus turning from slow to fast write MSRs, will be noted as such via a new CPUID bit. New MSRs that are fast upon introduction will be documented as such without an additional CPUID bit.

Three model specific MSRs are associated with the feature itself. They enable enumerating, controlling, and monitoring it. All three are logical processor scope.

15.4.8.1 FAST_UNCORE_MSRS_CAPABILITY (Address: 0x65F, Logical Processor Scope)

Operating systems or BIOS can read the FAST_UNCORE_MSRS_CAPABILITY MSR to enumerate those MSRs that are Fast Write MSRs.

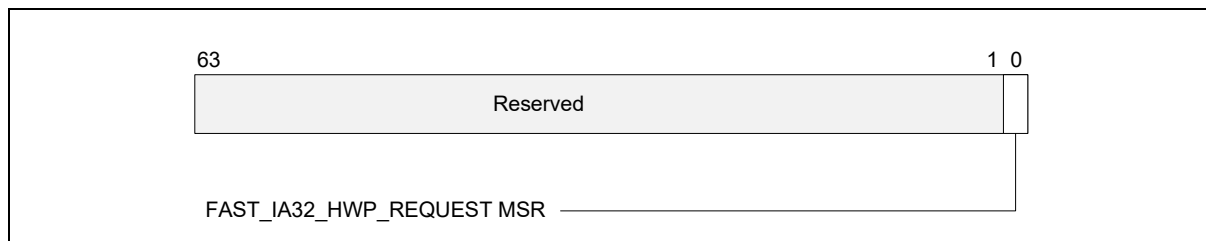


Figure 15-13. FAST_UNCORE_MSRS_CAPABILITY MSR

- **FAST_IA32_HWP_REQUEST MSR (bit 0, RO)** — When set (1), indicates that the IA32_HWP_REQUEST MSR is supported as a Fast Write MSR. A value of 0 indicates the IA32_HWP_REQUEST MSR is not supported as a Fast Write MSR.
- Bits 63:1 are reserved and must be zero.

15.4.8.2 FAST_UNCORE_MSRS_CTL (Address: 0x657, Logical Processor Scope)

Operating Systems or BIOS can use the FAST_UNCORE_MSRS_CTL MSR to opt-in or opt-out for fast write of specific MSRs that are enabled for Fast Write by the processor.

Note: Not all MSRs that are selected for this feature will necessarily have this opt-in/opt-out option. They may be supported in fast write mode only.

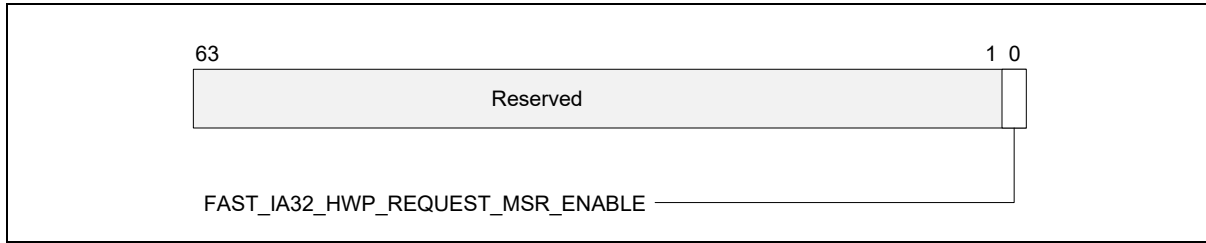


Figure 15-14. FAST_UNCORE_MSRS_CTL MSR

- **FAST_IA32_HWP_REQUEST_MSR_ENABLE (bit 0, RW)** — When set (1), enables fast access mode for the IA32_HWP_REQUEST MSR and sets the low latency, posted IA32_HWP_REQUEST MSR' CPUID[6].EAX[18]. The default value is 0. Note that this bit can only be enabled once from the default value. Once set, writes to this bit are ignored. Only RESET will clear this bit.
- Bits 63:1 are reserved and must be zero.

15.4.8.3 FAST_UNCORE_MSRS_STATUS (Address: 0x65E, Logical Processor Scope)

Software that executes the WRMSR instruction of a Fast Write MSR can check whether the data is already visible outside the logical processor by reading the FAST_UNCORE_MSRS_STATUS MSR. For each Fast Write MSR there is a status bit that indicates whether the data is already visible outside the logical processor or is still in “transit”.

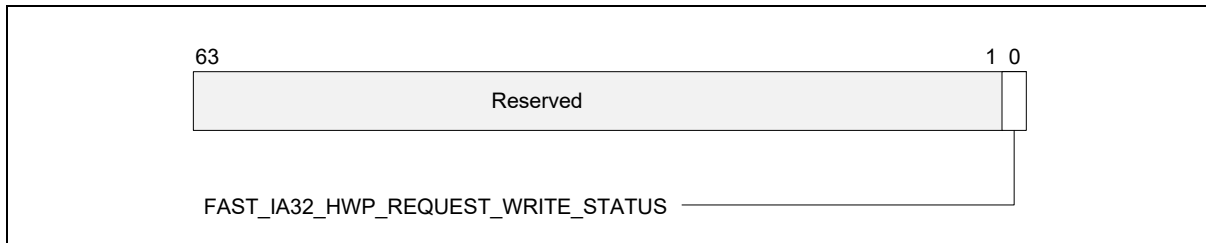


Figure 15-15. FAST_UNCORE_MSRS_STATUS MSR

- **FAST_IA32_HWP_REQUEST_WRITE_STATUS (bit 0, RO)** — Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor.
- Bits 63:1 are reserved and must be zero.

15.4.9 Fast_IA32_HWP_REQUEST CPUID

IA32_HWP_REQUEST is an architectural MSR that exists in processors whose CPUID[6].EAX[7] is set (HWP BASE is enabled). This MSR has logical processor scope, but after its contents are written the contents become visible outside the logical processor. When the FAST_IA32_HWP_REQUEST CPUID[6].EAX[18] bit is set, writes to the IA32_HWP_REQUEST MSR are visible outside the logical processor via the “Fast Write” feature described in Section 15.4.8.

15.4.10 Recommendations for OS use of HWP Controls

Common Cases of Using HWP

The default HWP control field values are expected to be suitable for many applications. The OS can enable autonomous HWP for these common cases by

- Setting IA32_HWP_REQUEST.Desired_Performance = 0 (hardware autonomous selection determines the performance target). Set IA32_HWP_REQUEST.Activity_Window = 0 (enable HW dynamic selection of window size).

To maximize HWP benefit for the common cases, the OS should set

- IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance and
- IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance.

Setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance is functionally equivalent to using of the IA32_PERF_CTL interface and is therefore not recommended (bypassing HWP).

Calibrating HWP for Application-Specific HWP Optimization

In some applications, the OS may have Quality of Service requirements that may not be met by the default values. The OS can characterize HWP by:

- keeping IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance to prevent non-linearity in the characterization process,
- utilizing the range values enumerated from the IA32_HWP_CAPABILITIES MSR to program IA32_HWP_REQUEST while executing workloads of interest and observing the power and performance result.

The power and performance result of characterization is also influenced by the IA32_HWP_REQUEST.Energy_Performance_Preference field, which must also be characterized.

Characterization can be used to set IA32_HWP_REQUEST.Minimum_Performance to achieve the required QOS in terms of performance. If IA32_HWP_REQUEST.Minimum_Performance is set higher than IA32_HWP_CAPABILITIES.Guaranteed_Performance then notification of excursions to Minimum Performance may be continuous.

If autonomous selection does not deliver the required workload performance, the OS should assess the current delivered effective frequency and for the duration of the specific performance requirement set IA32_HWP_REQUEST.Desired_Performance \neq 0 and adjust IA32_HWP_REQUEST.Energy_Performance_Preference as necessary to achieve the required workload performance. The MSR_PPERR.PCNT value can be used to better comprehend the potential performance result from adjustments to IA32_HWP_REQUEST.Desired_Performance. The OS should set IA32_HWP_REQUEST.Desired_Performance = 0 to re-enable autonomous selection.

Tuning for Maximum Performance or Lowest Power Consumption

Maximum performance will be delivered by setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance and setting IA32_HWP_REQUEST.Energy_Performance_Preference = 0 (performance preference).

Lowest power will be achieved by setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance and setting IA32_HWP_REQUEST.Energy_Performance_Preference = 0FFH (energy efficiency preference).

Mixing Logical Processor and Package Level HWP Field Settings

Using the IA32_HWP_REQUEST.Package_Control bit and the five valid bits in that MSR, the OS can mix and match between selecting the Logical Processor scope fields and the Package level fields. For example, the OS can set all logical cores' IA32_HWP_REQUEST.Package_Control bit to '1', and for those logical processors if it prefers a different EPP value than the one set in the IA32_HWP_REQUEST_PKG MSR, the OS can set the desired EPP value and the EPP valid bit. This overrides the package EPP value for only a subset of the logical processors in the package.

Additional Guidelines

Set IA32_HWP_REQUEST.Energy_Performance_Preference as appropriate for the platform's current mode of operation. For example, a mobile platforms' setting may be towards performance preference when on AC power and more towards energy efficiency when on DC power.

The use of the Running Average Power Limit (RAPL) processor capability (see section 14.7.1) is highly recommended when HWP is enabled. Use of IA32_HWP_Request.Maximum_Performance for thermal control is subject to limitations and can adversely impact the performance of other processor components, e.g., graphics

If default values deliver undesirable performance latency in response to events, the OS should set IA32_HWP_REQUEST.Activity_Window to a low (non-zero) value and IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) for the event duration.

Similarly, for “real-time” threads, set IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) and IA32_HWP_REQUEST.Activity_Window to a low value, e.g., 01H, for the duration of their execution.

When executing low priority work that may otherwise cause the hardware to deliver high performance, set IA32_HWP_REQUEST.Activity_Window to a longer value and reduce the IA32_HWP_Request.Maximum_Performance value as appropriate to control energy efficiency. Adjustments to IA32_HWP_REQUEST.Energy_Performance_Preference may also be necessary.

15.5 HARDWARE DUTY CYCLING (HDC)

Intel processors may contain support for Hardware Duty Cycling (HDC), which enables the processor to autonomously force its components inside the physical package into idle state. For example, the processor may selectively force only the processor cores into an idle state.

HDC is disabled by default on processors that support it. System software can dynamically enable or disable HDC to force one or more components into an idle state or wake up those components previously forced into an idle state. Forced Idling (and waking up) of multiple components in a physical package can be done with one WRMSR to a packaged-scope MSR from any logical processor within the same package.

HDC does not delay events such as timer expiration, but it may affect the latency of short (less than 1 msec) software threads, e.g., if a thread is forced to idle state just before completion and entering a “natural idle”.

HDC forced idle operation can be thought of as operating at a lower effective frequency. The effective average frequency computed by software will include the impact of HDC forced idle.

The primary use of HDC is enable system software to manage low active workloads to increase the package level C6 residency. Additionally, HDC can lower the effective average frequency in case of power or thermal limitation.

When HDC forces a logical processor, a processor core or a physical package to enter an idle state, its C-State is set to C3 or deeper. The deep “C-states” referred to in this section are processor-specific C-states.

15.5.1 Hardware Duty Cycling Programming Interfaces

The programming interfaces provided by HDC include the following:

- The CPUID instruction allows software to discover the presence of HDC support in an Intel processor. Specifically, execute CPUID instruction with EAX=06H as input, bit 13 of EAX indicates the processor’s support of the following aspects of HDC.
 - Availability of HDC baseline resource, CPUID.06H:EAX[bit 13]: If this bit is set, HDC provides the following architectural MSRs: IA32_PKG_HDC_CTL, IA32_PM_CTL1, and the IA32_THREAD_STALL MSRs.
- Additionally, HDC may provide several non-architectural MSR.

Table 15-3. Architectural and non-Architecture MSRs Related to HDC

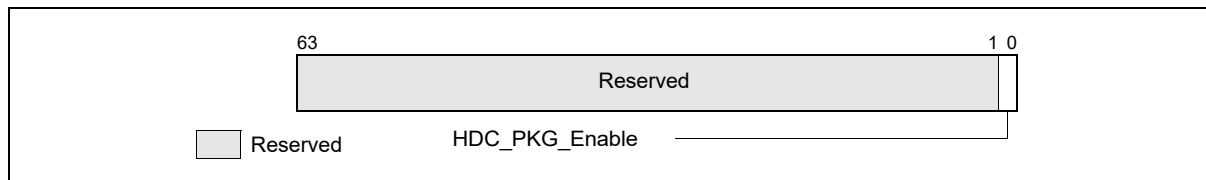
Address	Architectural	Register Name	Description
DB0H	Y	IA32_PKG_HDC_CTL	Package Enable/Disable HDC.
DB1H	Y	IA32_PM_CTL1	Per-logical-processor select control to allow/block HDC forced idling.
DB2H	Y	IA32_THREAD_STALL	Accumulate stalled cycles on this logical processor due to HDC forced idling.
653H	N	MSR_CORE_HDC_RESIDENCY	Core level stalled cycle counter due to HDC forced idling on one or more logical processor.
655H	N	MSR_PKG_HDC_SHALLOW_RESIDENCY	Accumulate the cycles the package was in C2 ¹ state and at least one logical processor was in forced idle
656H	N	MSR_PKG_HDC_DEEP_RESIDENCY	Accumulate the cycles the package was in the software specified Cx ¹ state and at least one logical processor was in forced idle. Cx is specified in MSR_PKG_HDC_CONFIG_CTL.
652H	N	MSR_PKG_HDC_CONFIG_CTL	HDC configuration controls

NOTES:

1. The package “C-states” referred to in this section are processor-specific C-states.

15.5.2 Package level Enabling HDC

The layout of the IA32_PKG_HDC_CTL MSR is shown in Figure 15-16. IA32_PKG_HDC_CTL is a writable MSR from any logical processor in a package. The bit fields are described below:

**Figure 15-16. IA32_PKG_HDC_CTL MSR**

- **HDC_PKG_Enable (bit 0, R/W)** — Software sets this bit to enable HDC operation by allowing the processor to force to idle all “HDC-allowed” (see Figure 15.5.3) logical processors in the package. Clearing this bit disables HDC operation in the package by waking up all the processor cores that were forced into idle by a previous ‘0’-to-‘1’ transition in IA32_PKG_HDC_CTL.HDC_PKG_Enable. This bit is writable only if CPUID.06H:EAX[bit 13] = 1. Default = zero (0).
- Bits 63:1 are reserved and must be zero.

After processor support is determined via CPUID, system software can enable HDC operation by setting IA32_PKG_HDC_CTL.HDC_PKG_Enable to 1. At reset, IA32_PKG_HDC_CTL.HDC_PKG_Enable is cleared to 0. A ‘0’-to-‘1’ transition in HDC_PKG_Enable allows the processor to force to idle all HDC-allowed (indicated by the non-zero state of IA32_PM_CTL1[bit 0]) logical processors in the package. A ‘1’-to-‘0’ transition wakes up those HDC force-idled logical processors.

Software can enable or disable HDC using this package level control multiple times from any logical processor in the package. Note the latency of writing a value to the package-visible IA32_PKG_HDC_CTL.HDC_PKG_Enable is longer than the latency of a WRMSR operation to a Logical Processor MSR (as opposed to package level MSR) such as: IA32_PM_CTL1 (described in Section 15.5.3). Propagation of the change in IA32_PKG_HDC_CTL.HDC_PKG_Enable and reaching all HDC idled logical processor to be woken up may take on the order of core C6 exit latency.

15.5.3 Logical-Processor Level HDC Control

The layout of the IA32_PM_CTL1 MSR is shown in Figure 15-17. Each logical processor in a package has its own IA32_PM_CTL1 MSR. The bit fields are described below:

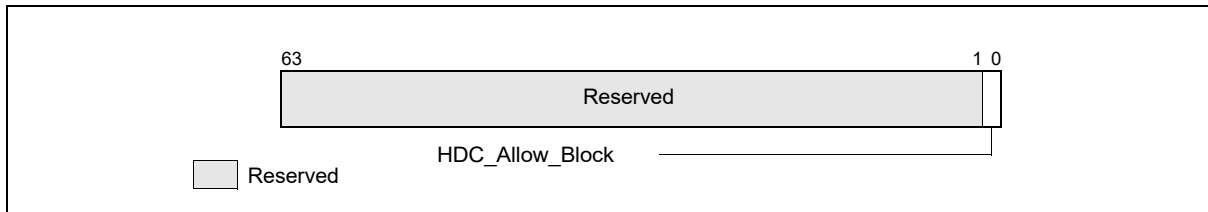


Figure 15-17. IA32_PM_CTL1 MSR

- **HDC_Allow_Block (bit 0, R/W)** — Software sets this bit to allow this logical processors to honor the package-level IA32_PKG_HDC_CTL.HDC_PKG_Enable control. Clearing this bit prevents this logical processor from using the HDC. This bit is writable only if CPUID.06H:EAX[bit 13] = 1. Default = one (1).
- Bits 63:1 are reserved and must be zero.

Fine-grain OS control of HDC operation at the granularity of per-logical-processor is provided by IA32_PM_CTL1. At RESET, all logical processors are allowed to participate in HDC operation such that OS can manage HDC using the package-level IA32_PKG_HDC_CTL.

Writes to IA32_PM_CTL1 complete with the latency that is typical to WRMSR to a Logical Processor level MSR. When the OS chooses to manage HDC operation at per-logical-processor granularity, it can write to IA32_PM_CTL1 on one or more logical processors as desired. Each write to IA32_PM_CTL1 must be done by code that executes on the logical processor targeted to be allowed into or blocked from HDC operation.

Blocking one logical processor for HDC operation may have package level impact. For example, the processor may decide to stop duty cycling of all other Logical Processors as well.

The propagation of IA32_PKG_HDC_CTL.HDC_PKG_Enable in a package takes longer than a WRMSR to IA32_PM_CTL1. The last completed write to IA32_PM_CTL1 on a logical processor will be honored when a '0'-to-'1' transition of IA32_PKG_HDC_CTL.HDC_PKG_Enable arrives to a logical processor.

15.5.4 HDC Residency Counters

There is a collection of counters available for software to track various residency metrics related to HDC operation. In general, HDC residency time is defined as the time in HDC forced idle state at the granularity of per-logical-processor, per-core, or package. At the granularity of per-core/package-level HDC residency, at least one of the logical processor in a core/package must be in the HDC forced idle state.

15.5.4.1 IA32_THREAD_STALL

Software can track per-logical-processor HDC residency using the architectural MSR IA32_THREAD_STALL. The layout of the IA32_THREAD_STALL MSR is shown in Figure 15-18. Each logical processor in a package has its own IA32_THREAD_STALL MSR. The bit fields are described below:

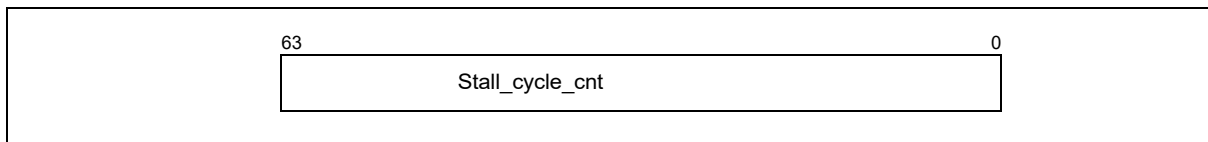


Figure 15-18. IA32_THREAD_STALL MSR

- **Stall_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after the logical processor exits from the forced idled C-state. At each update, the number of cycles that the logical processor was stalled due to forced-idle will be added to the counter. This counter is available only if CPUID.06H:EAX[bit 13] = 1. Default = zero (0).

A value of zero in IA32_THREAD_STALL indicates either HDC is not supported or the logical processor never serviced any forced HDC idle. A non-zero value in IA32_THREAD_STALL indicates the HDC forced-idle residency times of the logical processor. It also indicates the forced-idle cycles due to HDC that could appear as C0 time to traditional OS accounting mechanisms (e.g., time-stamping OS idle/exit events).

Software can read IA32_THREAD_STALL irrespective of the state of IA32_PKG_HDC_CTL and IA32_PM_CTL1, as long as CPUID.06H:EAX[bit 13] = 1.

15.5.4.2 Non-Architectural HDC Residency Counters

Processors that support HDC operation may provide the following model-specific HDC residency counters.

MSR_CORE_HDC_RESIDENCY

Software can track per-core HDC residency using the counter MSR_CORE_HDC_RESIDENCY. This counter increments when the core is in C3 state or deeper (all logical processors in this core are idle due to either HDC or other mechanisms) and at least one of the logical processors is in HDC forced idle state. The layout of the MSR_CORE_HDC_RESIDENCY is shown in Figure 15-19. Each processor core in a package has its own MSR_CORE_HDC_RESIDENCY MSR. The bit fields are described below:

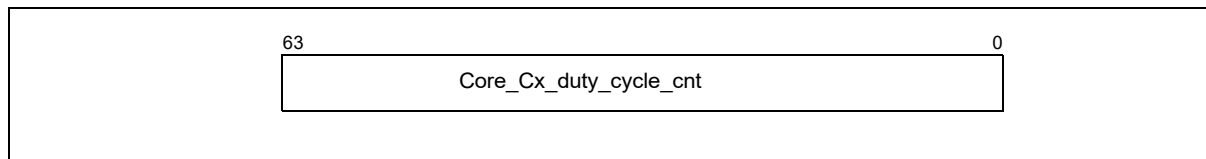


Figure 15-19. MSR_CORE_HDC_RESIDENCY MSR

- **Core_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after core C-state exit from a forced idled C-state. At each update, the increment counts cycles when the core is in a Cx state (all its logical processor are idle) and at least one logical processor in this core was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR will cause a #GP fault. Default = zero (0).

A value of zero in MSR_CORE_HDC_RESIDENCY indicates either HDC is not supported or this processor core never serviced any forced HDC idle.

MSR_PKG_HDC_SHALLOW_RESIDENCY

The counter MSR_PKG_HDC_SHALLOW_RESIDENCY allows software to track HDC residency time when the package is in C2 state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. The layout of the MSR_PKG_HDC_SHALLOW_RESIDENCY is shown in Figure 15-20. There is one MSR_PKG_HDC_SHALLOW_RESIDENCY per package. The bit fields are described below:

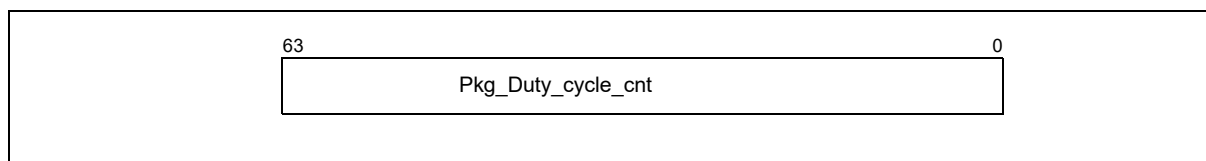


Figure 15-20. MSR_PKG_HDC_SHALLOW_RESIDENCY MSR

- Pkg_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. Package shallow residency may be implementation specific. In the initial implementation, the threshold is package C2-state. The count is updated only after package C2-state exit from a forced idled C-state. At each update, the increment counts cycles when the package is in C2 state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_DEEP_RESIDENCY

The counter MSR_PKG_HDC_DEEP_RESIDENCY allows software to track HDC residency time when the package is in a software-specified package Cx state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. Selection of a specific package Cx state can be configured using MSR_PKG_HDC_CONFIG. The layout of the MSR_PKG_HDC_DEEP_RESIDENCY is shown in Figure 15-21. There is one MSR_PKG_HDC_DEEP_RESIDENCY per package. The bit fields are described below:

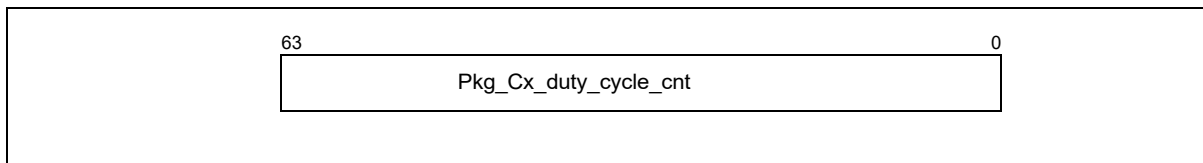


Figure 15-21. MSR_PKG_HDC_DEEP_RESIDENCY MSR

- Pkg_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after package C-state exit from a forced idle state. At each update, the increment counts cycles when the package is in the software-configured Cx state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_CONFIG

MSR_PKG_HDC_CONFIG allows software to configure the package Cx state that the counter MSR_PKG_HDC_DEEP_RESIDENCY monitors. The layout of the MSR_PKG_HDC_CONFIG is shown in Figure 15-22. There is one MSR_PKG_HDC_CONFIG per package. The bit fields are described below:

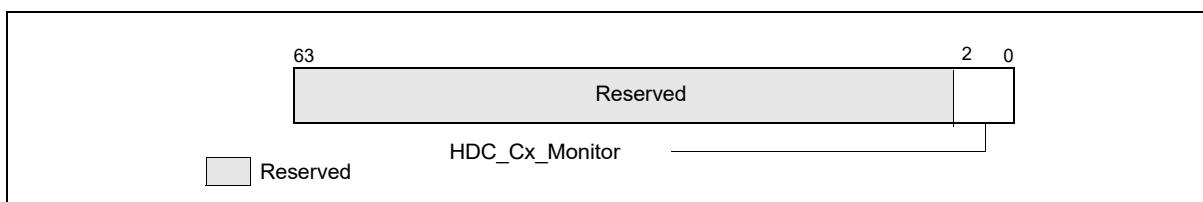


Figure 15-22. MSR_PKG_HDC_CONFIG MSR

- Pkg_Cx_Monitor (bits 2:0, R/W)** — Selects which package C-state the MSR_HDC_DEEP_RESIDENCY counter will monitor. The encoding of the HDC_Cx_Monitor field are: **0**: no-counting; **1**: count package C2 only; **2**: count package C3 and deeper; **3**: count package C6 and deeper; **4**: count package C7 and deeper; other encodings are reserved. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).
- Bits 63:3 are reserved and must be zero.

15.5.5 MPERF and APERF Counters Under HDC

HDC operation can be thought of as an average effective frequency drop due to all or some of the Logical Processors enter an idle state period.

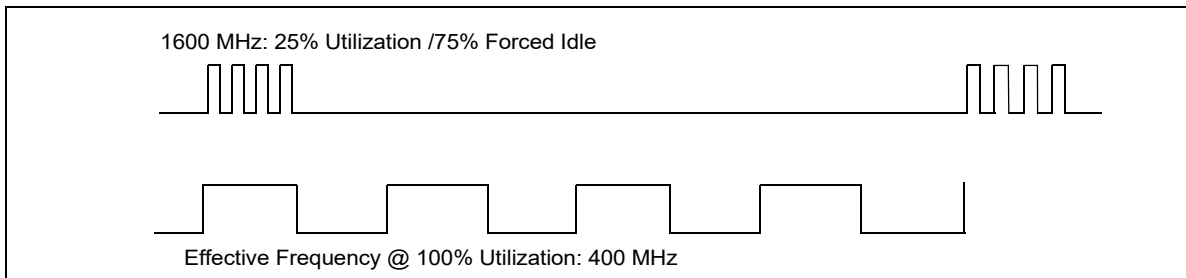


Figure 15-23. Example of Effective Frequency Reduction and Forced Idle Period of HDC

By default, the IA32_MPERF counter counts during forced idle periods as if the logical processor was active. The IA32_APERF counter does not count during forced idle state. This counting convention allows the OS to compute the average effective frequency of the Logical Processor between the last MWAIT exit and the next MWAIT entry (OS visible C0) by $\Delta\text{ACNT}/\Delta\text{MCNT} * \text{TSC Frequency}$.

15.6 HARDWARE FEEDBACK INTERFACE AND INTEL® THREAD DIRECTOR

Intel processors that enumerate CPUID.06H.0H:EAX.HW_FEEDBACK[bit 19] as 1 support Hardware Feedback Interface (HFI). Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a hardware feedback interface structure in memory. Details on this table structure are described in Section 15.6.1.

Intel processors that enumerate CPUID.06H.0H:EAX[bit 23] as 1 support Intel® Thread Director. Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a memory resident table and software thread specific index (Class ID) that points into that table and selects which data to use for that software thread. Details on this table structure are described in Section 15.6.2.

15.6.1 Hardware Feedback Interface Table Structure

This structure has a global header that is 16 bytes in size. Following this global header, there is one 8 byte entry per logical processor in the socket. The structure is designed as follows.

Table 15-4. Hardware Feedback Interface Structure

Byte Offset	Size (Bytes)	Description
0	16	Global Header
16	8	Per Logical Processor Entry
24	8	Per Logical Processor Entry
...
16 + n*8	8	Per Logical Processor Entry

The global header is structured as shown in Table 15-5.

Table 15-5. Hardware Feedback Interface Global Header Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	8	Timestamp	Timestamp of when the table was last updated by hardware. This is a timestamp in crystal clock units. Initialized by the OS to 0.
8	1	Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
9	1	Energy Efficiency Capability Changed	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
10	6	Reserved	Initialized by the OS to 0.

The per logical processor scheduler feedback entry is structured as follows. The operating system can determine the index of the logical processor feedback entry for a logical processor using CPUID.06H.0H:EDX[31:16] by executing CPUID on that logical processor.

Table 15-6. Hardware Feedback Interface Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.0H:EDX[0] enumerates support for Performance capability reporting.
1	1	Energy Efficiency Capability	Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.0H:EDX[1] enumerates support for Energy Efficiency capability reporting.
2	6	Reserved	The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback.

15.6.2 Intel® Thread Director Table Structure

This structure has a global header that is at least 16 bytes in size. Its size depends on the number of classes and capabilities enumerated by the CPUID instruction (see notes below Table 15-7). Following this global header there are multiple Logical Processor related entries. The structure is designed as follows.

Table 15-7. Intel® Thread Director Table Structure

Byte Offset ^{1,2,3}	Size (Bytes)	Description
0	$8 + CP^4 * CL^4 + R8^5$	Global Header
$8 + CP * CL + R8$	$CL * CP + R8$	Per Logical Processor Entry ₀ ⁶
$8 + 2 * (CP * CL + R8)$	$CL * CP + R8$	Per Logical Processor Entry ₁
...
$8 + (N^7 - 1) * (CP * CL + R8)$	$CL * CP + R8$	Logical Processor Entry _{N-1}

NOTES:

1. Byte offset of Capability_{cp} of Class_{cl} change indication: $8 + CP * cl + cp$.
2. Byte offset of LP Entry_i: $8 + (i+1) * (CP * CL + R8)$.
3. Byte offset of capability_{cp} of class_{cl} of LP Entry_i: $8 + (i+1) * (CP * CL + R8) + CP * cl + cp$.
4. Both upper case CL and CP denote total number of classes and capabilities defined for the processor. Lower case cl and cp denote one instance of a class or capability. cl and cp are counted starting at zero. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for the number of classes (CL) and the number of supported capabilities (CP). CP (# of capabilities): number of enumerated bits in CPUID.06H.0H.EDX[7:0] and CL (# of classes): CPUID.06H.0H.ECX[15:8].
5. R8 is the number of bytes necessary to round up the Capability Change Indication array and the Logical Processor Entry to whole multiple of 8 bytes.
6. Table size: $8 + (N+1) * (CP * CL + R8)$.
7. N is the number of Logical Processor Entries in the table. It is not greater than the number of Logical Processors on the socket, but may be lower.
8. The Operating System can determine the index for the Logical Processor Entry within the Intel Thread Director table using CPUID.06H.0H:EDX[31:16] by executing the CPUID instruction on that Logical Processor.
9. The Operating System should allocate space to accommodate for one such structure per socket in the system.
10. The Intel Thread Director table structure extends the Hardware Feedback Interface table structure without breaking backward compatibility. The Hardware Feedback Interface can be viewed as having two capabilities and a single class.

The global header is structured as shown in Table 15-8.

Table 15-8. Intel® Thread Director Global Header Structure

Byte Offset	Size (Bytes)	Description	
0	8	Time-stamp of when the table was last updated by hardware. This is a time-stamp in crystal clock units. Initialized by the OS to 0.	
8	1	Class 0 Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + 1	1	Class 0 Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
...			
8 + CP - 1	1	Class 0 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
8 + CP	1	Class 1 Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + CP + 1	1	Class 1 Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
...			
8 + 2*CP - 1	1	Class 1 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
...			Change indication for Capabilities of additional Classes if exist.
8 + (CL-1)*CP	1	Class #(CL-1) Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + (CL-1)*CP + 1	1	Class #(CL-1) Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.

Table 15-8. Intel® Thread Director Global Header Structure (Contd.)

Byte Offset	Size (Bytes)	Description	
...			
8 + CL*CP - 1	1	Class #(CL-1) change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
8 + CL*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.

The logical processor capability entry in the Intel Thread Director table is structured as follows.

Table 15-9. Intel® Thread Director Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Class 0 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
1	1	Energy Efficiency Capability	Class 0 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
CP - 1	1	Capability #(CP-1)	Class 0 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.
CP + R8	1	Performance Capability	Class 1 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
CP + 1	1	Energy Efficiency Capability	Class 1 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
2*CP - 1	1	Capability #(CP-1)	Class 1 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).

Table 15-9. Intel® Thread Director Logical Processor Entry Structure (Contd.)

Byte Offset	Size (Bytes)	Field Name	Description
2*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.
...			
(CL-1)*CP	1	Performance Capability	Class #(CL-1) Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
(CL-1)*CP + 1	1	Energy Efficiency Capability	Class #(CL-1) Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
CL*CP - 1	1	Capability #(CP-1)	Class #(CL-1) Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CL*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.

15.6.3 Intel® Thread Director Usage Model

When the OS Scheduler needs to decide which one of multiple free logical processors to assign to a software thread that is ready to execute, it can choose one of the following options:

1. The free logical processor with the highest performance value of that software thread class, if the system is scheduling for performance.
2. The free logical processor with the highest energy efficiency value of that software thread class, if the system is scheduling for energy efficiency.

When the OS Scheduler needs to decide which of two logical processors (i,j) to assign to which of two software threads whose Class IDs are k1 and k2, it can compute the two performance ratios: $\text{Perf Ratio}_1 = \text{Perf}_{ik1} / \text{Perf}_{jk1}$ and $\text{Perf Ratio}_2 = \text{Perf}_{ik2} / \text{Perf}_{jk2}$, or two energy efficiency ratios: $\text{Energy Eff. Ratio}_1 = \text{Energy Eff}_{ik1} / \text{Energy Eff}_{jk1}$ and $\text{Energy Eff. Ratio}_2 = \text{Energy Eff}_{ik2} / \text{Energy Eff}_{jk2}$ between the two logical processors for each of the two classes, depending on whether the OS is scheduling for performance or for energy efficiency.

For example, assume that the system is scheduling for performance and that $\text{Perf Ratio}_1 > \text{Perf Ratio}_2$. The OS Scheduler will assign the software thread whose Class ID is k1 to logical processor i, and the one whose Class ID is k2 to logical processor j.

When the two software threads in question belong to the same Class ID, the OS Scheduler can schedule to higher performance logical processors within that class when scheduling for performance and to higher energy efficiency logical processors within that class when scheduling for energy efficiency.

The highest to lowest ordering may be different between classes across cores and between the performance column and the energy efficiency column of the same class across cores.

15.6.4 Hardware Feedback Interface Pointer

The physical address of the HFI/Intel Thread Director structure is programmed by the OS into a package scoped MSR named IA32_HW_FEEDBACK_PTR. The MSR is structured as follows:

- Bits 63:MAXPHYADDR¹ – Reserved.
- Bits MAXPHYADDR-1:12 – ADDR. This is the physical address of the page frame of the first page of this structure.
- Bits 11:1 – Reserved.
- Bit 0 – Valid. When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.

The address of this MSR is 17D0H. This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

CPUID.06H.0H:EDX[11:8] enumerates the size of memory that must be allocated by the OS for this structure.

15.6.5 Hardware Feedback Interface Configuration

The operating system enables HFI/Intel Thread Director using a package scoped MSR named IA32_HW_FEEDBACK_CONFIG (address 17D1H). This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

The MSR is structured as follows:

- Bits 63:2 – Reserved.
- Bit 1 – Enable Intel Thread Director (or multi-class support). Both bits 0 and 1 must be set for Intel Thread Director to be enabled. The extra class columns in the Intel Thread Director table are updated by hardware immediately following setting those two bits, as well as during run time as necessary.
- Bit 0 – Enable. When set to 1, enables HFI.

Before enabling HFI, the OS must set a valid hardware feedback interface structure using IA32_HW_FEEDBACK_PTR.

When the OS sets bit 0 only, the hardware populates class 0 capabilities only in the HFI structure. When bit 1 is set after or together with bit 0, the Intel Thread Director multi-class structure is populated.

When either the HFI structure or the Intel Thread Director structure are ready to use by the OS, the hardware sets IA32_PACKAGE_THERM_STATUS[bit 26]. An interrupt is generated by the hardware if IA32_PACKAGE_THERM_INTERRUPT[bit 25] is set.

When the OS clears bit 1 but leaves bit 0 set, Intel Thread Director is disabled, but HFI is kept operational. IA32_PACKAGE_THERM_STATUS[bit 26] is NOT set in this case.

Clearing bit 0 disables both HFI and Intel Thread Director, independent of the bit 1 state. Setting bit 1 to '1' while keeping bit 0 at '0' is an invalid combination which is quietly ignored.

When the OS clears bit 0, hardware sets the IA32_PACKAGE_THERM_STATUS[bit 26] to 1 to acknowledge disabling of the interface. The OS should wait for this bit to be set to 1 to reclaim the memory of the Intel Thread Director structure, as by setting IA32_PACKAGE_THERM_STATUS[bit 26] hardware guarantees not to write into the Intel Thread Director structure anymore.

The OS may clear bit 0 only after receiving an indication from the hardware that the structure initialization is complete via the same IA32_PACKAGE_THERM_STATUS[bit 26], following enabling of HFI/Intel Thread Director, thus avoiding a race condition between OS and hardware.

Bit 1 is valid only if CPUID[6].EAX[bit 23] is set. When setting this bit while support is not enumerated, the hardware generates #GP.

Table 15-10 summarizes the control options described above.

See Section 15.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

1. MAXPHYADDR is reported in CPUID.80000008H:EAX[7:0].

Table 15-10. IA32_HW_FEEDBACK_CONFIG Control Options

Pre-Bit 1	Pre-Bit 0	Post-Bit 1	Post-Bit 0	Action	IA32_PACKAGE_THERM_STATUS [bit 26] and Interrupt
0	0	0	0	Reset value.	Both Hardware Feedback Interface and Intel Thread Director are disabled, no status bit set, no interrupt is generated.
0	0	0	1	Enable HFI structure.	Set the status bit and generate interrupt if enabled.
0	0	1	0	Invalid option; quietly ignored by the hardware.	No action (no update in the table).
0	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	0	0	Disable HFI support.	Set the status bit and generate interrupt if enabled.
0	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	1	1	Enable Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	0	0	0	No action; keeps HFI and Intel Thread Director disabled.	No action (no update in the table).
1	0	0	1	Enable HFI.	Set the status bit and generate interrupt if enabled.
1	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	1	Disable Intel Thread Director; keep HFI enabled.	No action (no update in the table).
1	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.

15.6.6 Hardware Feedback Interface Notifications

The IA32_PACKAGE_THERM_STATUS MSR is extended with a new bit, hardware feedback interface structure change status (bit 26, R/WC0), to indicate that the hardware has updated the HFI/Intel Thread Director structure. This is a sticky bit and once set, indicates that the OS should read the structure to determine the change and adjust its scheduling decisions. Once set, the hardware will not generate any further updates to this structure until the OS clears this bit by writing 0.

The OS can enable interrupt-based notifications when the structure is updated by hardware through a new enable bit, hardware feedback interrupt enable (bit 25, R/W), in the IA32_PACKAGE_THERM_INTERRUPT MSR. When this bit is set to 1, it enables the generation of an interrupt when the HFI/Intel Thread Director structure is updated by hardware. When the enable bit transitions from 0 to 1, hardware will generate an initial notify, with the IA32_PACKAGE_THERM_STATUS bit 26 set to 1, to indicate that the OS should read the current HFI/Intel Thread Director structure.

15.6.7 Hardware Feedback Interface and Intel® Thread Director Structure Dynamic Update

The HFI/Intel Thread Director structure can be updated dynamically during run time. Changes to the structure may occur to one or more of its cells. Such changes may occur for one or more logical processors. The hardware sets a non-zero value in the “capability change” field of the HFI/Intel Thread Director structure as an indication for the OS to read that capability for all logical processors. A thermal interrupt is delivered to indicate to the OS that the structure has just changed. Section 15.6.6 contains more details on this notification mechanism. The hardware clears all “capability change” fields after the OS resets IA32_PACKAGE_THERM_STATUS[bit 26].

Zeroing a performance or energy efficiency cell hints to the OS that it is beneficial not to schedule software threads of that class on the associated logical processor for performance or energy efficiency reasons, respectively. If SMT is supported, it may be the case that the hardware zeroes one of the core's logical processors only. Zeroing the performance and energy efficiency cells of all classes for a logical processor implies that the hardware provides a hint to the OS to completely avoid scheduling work on that logical processor.

Zeroing a performance and energy efficiency cell hint of a logical processor across all classes along with Capability Flag bit 1 set to 1 across all capabilities and classes, indicates to the OS to force idle logical processor(s), and if affinity activity occurs on those logical processor(s), the OS should inject idle periods such that overall utilization of those idled cores has a minimal-to-no impact to power. Capability Flag bit 1 will be set to 1 while this hint persists.

When EE=255 is set on one or more logical processors, it represents a request that the OS attempt to consolidate work to those logical processors with EE=255. These requests are made when the SOC has knowledge that consolidating the work to a subset of cores will result in significantly better platform energy efficiency. Examples of consolidating work would include, but not limited to, delaying less important work as needed to provide compute bandwidth for more important work, and routing interrupts to the logical processors with EE=255. When the cumulative workload requires performance greater than that which is available on the subset of cores with EE=255, it is expected that the OS will scale the work out to additional logical processors.

A few example reasons for runtime changes in the HGS/Intel Thread Director Table:

- Over clocking run time update that changes the capability values.
- Change in run time physical constraints.
- Run time performance or energy efficiency optimization.
- Change in core frequency, voltage, or power budget.

15.6.8 Logical Processor Scope Intel® Thread Director Configuration

The operating system enables Intel Thread Director at the logical processor scope using a logical processor scope MSR named IA32_HW_FEEDBACK_THREAD_CONFIG (address 17D4H).

The MSR is read/write and is structured as follows:

- Bits 63:1 – Reserved.
- Bit 0 – Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).

Bit 0 of the logical processor scope configuration MSR can be cleared or set regardless of the state of the HFI/Intel Thread Director package configuration MSR state. Even when bit 0 of all logical processor configuration MSRs is clear, the processor can still update the Intel Thread Director structure if it is still enabled in the IA32_HW_FEEDBACK_CONFIG package scope MSR. When the operating system clears IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0], hardware clears the history accumulated on that logical processor which otherwise drives assigning the Class ID to the software thread that executes on that logical processor. As long as IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set, the Class ID is available for the operating system to read, independent of the state of the package scope IA32_HW_FEEDBACK_CONFIG[1:0] bits.

See Section 15.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

15.6.9 Implicit Reset of Package and Logical Processor Scope Configuration MSRs

HFI/Intel Thread Director enable bits are reset by hardware in the following scenarios:

1. When GETSEC[SENDER] is executed:
 - a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on all sockets (packages) in the system.
 - b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors in the system across all sockets.
 - c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR across all sockets.
2. When GETSEC[ENTERACCS] is executed:
 - a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.
 - b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors on the socket where the GETSEC[ENTERACCS] instruction was executed.
 - c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.
3. When an INIT or a wait-for-SIPI state are processed by a logical processor:
 - a. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on that logical processor, whether the signal was in the context of GETSEC[ENTERACCS] or not.

If the OS requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

15.6.10 Logical Processor Scope Intel® Thread Director Run Time Characteristics

The processor provides the operating system with run time feedback about the execution characteristics of the software thread executing on logical processors whose IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set.

The run time feedback is communicated via a read-only MSR named IA32_THREAD_FEEDBACK_CHAR. This is a logical processor scope MSR whose address is 17D2H. This MSR is structured as follows:

- Bit 63 – Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.
- Bits 62:8 – Reserved.
- Bits 7:0 – Application Class ID, pointing into the Intel Thread Director structure described in Table 15-8.

This MSR is valid only if CPUID.06H:EAX[bit 23] is set.

The valid bit is cleared by the hardware in the following cases:

- The hardware does not have enough information to provide the operating system with a reliable Class ID.
- The operating system cleared the logical processor's IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] bit.

The HRESET instruction is executed while configured to reset the Intel Thread Director history.

15.6.11 Logical Processor Scope History

The operating system can reset the Intel Thread Director related history accumulated on the current logical processor it is executing on by issuing the HRESET instruction. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for enumeration of the HRESET

instruction. See also the “HRESET—History Reset” instruction description in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

15.6.11.1 Enabling Intel® Thread Director History Reset

The IA32_HRESET_ENABLE MSR is a read/write MSR and is structured as follows:

- Bits 63:32 – Reserved.
- Bits 31:1 – Reserved for other capabilities that can be reset by the HRESET instruction.
- Bit 0 – Enable reset of the Intel Thread Director history.

The operating system should set IA32_HRESET_ENABLE[bit 0] to enable Intel Thread Director history reset via the HRESET instruction.

15.6.11.2 Implicit Intel® Thread Director History Reset

The Intel Thread Director history is implicitly reset in the following scenarios:

1. When the processor enters or exits SMM mode and IA32_DEBUGCTL.MSR.FREEZE_WHILE_SMM (bit 14) is set, the Intel Thread Director history is implicitly reset by the processor.
2. When GETSEC[SENDER] is executed, the processor resets the Intel Thread Director history on all logical processors in the system, including logical processors on other sockets (other than the one GETSEC[SENDER] is executed).
3. When GETSEC[ENTERACCS] is executed, the processor resets the Intel Thread Director history on the logical processor it is executed on.
4. When an INIT or a wait-for-SIPI state are processed by a logical processor, the Intel Thread Director history is reset whether the signal was a result of GETSEC[ENTERACCS] or not.

If the operating system requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

15.7 MWAIT EXTENSIONS FOR ADVANCED POWER MANAGEMENT

IA-32 processors may support a number of C-states¹ that reduce power consumption for inactive states. Intel Core Solo and Intel Core Duo processors support both deeper C-state and MWAIT extensions that can be used by OS to implement power management policy.

Software should use CPUID to discover if a target processor supports the enumeration of MWAIT extensions. If CPUID.05H.ECX[Bit 0] = 1, the target processor supports MWAIT extensions and their enumeration (see Chapter 4, “Instruction Set Reference, M-U,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B).

If CPUID.05H.ECX[Bit 1] = 1, the target processor supports using interrupts as break-events for MWAIT, even when interrupts are disabled. Use this feature to measure C-state residency as follows:

- Software can write to bit 0 in the MWAIT Extensions register (ECX) when issuing an MWAIT to enter into a processor-specific C-state or sub C-state.
- When a processor comes out of an inactive C-state or sub C-state, software can read a timestamp before an interrupt service routine (ISR) is potentially executed.

CPUID.05H.EDX allows software to enumerate processor-specific C-states and sub C-states available for use with MWAIT extensions. IA-32 processors may support more than one C-state of a given C-state type. These are called sub C-states. Numerically higher C-state have higher power savings and latency (upon entering and exiting) than lower-numbered C-state.

1. The processor-specific C-states defined in MWAIT extensions can map to ACPI defined C-state types (C0, C1, C2, C3). The mapping relationship depends on the definition of a C-state by processor implementation and is exposed to OSPM by the BIOS using the ACPI defined _CST table.

At CPL = 0, system software can specify desired C-state and sub C-state by using the MWAIT hints register (EAX). Processors will not go to C-state and sub C-state deeper than what is specified by the hint register. If CPL > 0 and if MONITOR/MWAIT is supported at CPL > 0, the processor will only enter C1-state (regardless of the C-state request in the hints register).

Executing MWAIT generates an exception on processors operating at a privilege level where MONITOR/MWAIT are not supported.

NOTE

If MWAIT is used to enter a C-state (including sub C-state) that is numerically higher than C1, a store to the address range armed by MONITOR instruction will cause the processor to exit MWAIT if the store was originated by other processor agents. A store from non-processor agent may not cause the processor to exit MWAIT.

15.8 THERMAL MONITORING AND PROTECTION

The IA-32 architecture provides the following mechanisms for monitoring temperature and controlling thermal power:

1. The **catastrophic shutdown detector** forces processor execution to stop if the processor's core temperature rises above a preset limit.
2. **Automatic and adaptive thermal monitoring mechanisms** force the processor to reduce its power consumption in order to operate within predetermined temperature limits.
3. The **software controlled clock modulation mechanism** permits operating systems to implement power management policies that reduce power consumption; this is in addition to the reduction offered by automatic thermal monitoring mechanisms.
4. **On-die digital thermal sensor and interrupt mechanisms** permit the OS to manage thermal conditions natively without relying on BIOS or other system board components.

The first mechanism is not visible to software. The other three mechanisms are visible to software using processor feature information returned by executing CPUID with EAX = 1.

The second mechanism includes:

- **Automatic thermal monitoring** provides two modes of operation. One mode modulates the clock duty cycle; the second mode changes the processor's frequency. Both modes are used to control the core temperature of the processor.
- **Adaptive thermal monitoring** can provide flexible thermal management on processors made of multiple cores.

The third mechanism modulates the clock duty cycle of the processor. As shown in Figure 15-24, the phrase 'duty cycle' does not refer to the actual duty cycle of the clock signal. Instead it refers to the time period during which the clock signal is allowed to drive the processor chip. By using the stop clock mechanism to control how often the processor is clocked, processor power consumption can be modulated.

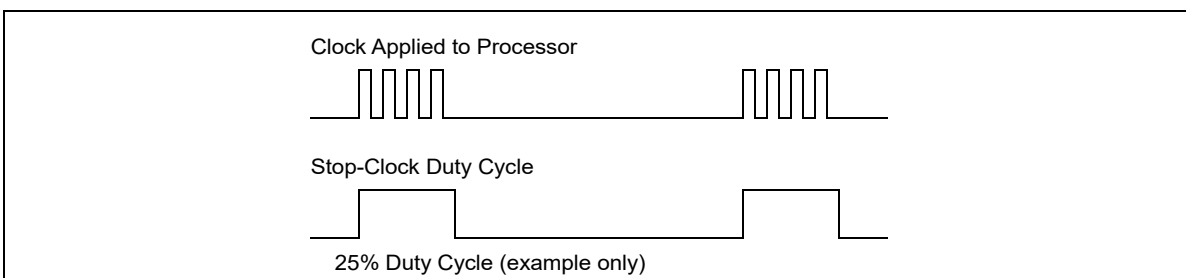


Figure 15-24. Processor Modulation Through Stop-Clock Mechanism

For previous automatic thermal monitoring mechanisms, software controlled mechanisms that changed processor operating parameters to impact changes in thermal conditions. Software did not have native access to the native thermal condition of the processor; nor could software alter the trigger condition that initiated software program control.

The fourth mechanism (listed above) provides access to an on-die digital thermal sensor using a model-specific register and uses an interrupt mechanism to alert software to initiate digital thermal monitoring.

15.8.1 Catastrophic Shutdown Detector

P6 family processors introduced a thermal sensor that acts as a catastrophic shutdown detector. This catastrophic shutdown detector was also implemented in Pentium 4, Intel Xeon and Pentium M processors. It is always enabled. When processor core temperature reaches a factory preset level, the sensor trips and processor execution is halted until after the next reset cycle.

15.8.2 Thermal Monitor

Pentium 4, Intel Xeon and Pentium M processors introduced a second temperature sensor that is factory-calibrated to trip when the processor's core temperature crosses a level corresponding to the recommended thermal design envelop. The trip-temperature of the second sensor is calibrated below the temperature assigned to the catastrophic shutdown detector.

15.8.2.1 Thermal Monitor 1

The Pentium 4 processor uses the second temperature sensor in conjunction with a mechanism called Thermal Monitor 1 (TM1) to control the core temperature of the processor. TM1 controls the processor's temperature by modulating the duty cycle of the processor clock. Modulation of duty cycles is processor model specific. Note that the processors STPCLK# pin is not used here; the stop-clock circuitry is controlled internally.

Support for TM1 is indicated by `CPUID.1:EDX.TM[bit 29] = 1`.

TM1 is enabled by setting the thermal-monitor enable flag (bit 3) in `IA32_MISC_ENABLE` [see Chapter 2, "Model-Specific Registers (MSRs)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4]. Following a power-up or reset, the flag is cleared, disabling TM1. BIOS is required to enable only one automatic thermal monitoring modes. Operating systems and applications must not disable the operation of these mechanisms.

15.8.2.2 Thermal Monitor 2

An additional automatic thermal protection mechanism, called Thermal Monitor 2 (TM2), was introduced in the Intel Pentium M processor and also incorporated in newer models of the Pentium 4 processor family. Intel Core Duo and Solo processors, and Intel Core 2 Duo processor family all support TM1 and TM2. TM2 controls the core temperature of the processor by reducing the operating frequency and voltage of the processor and offers a higher performance level for a given level of power reduction than TM1.

TM2 is triggered by the same temperature sensor as TM1. The mechanism to enable TM2 may be implemented differently across various IA-32 processor families with different CPUID signatures in the family encoding value, but will be uniform within an IA-32 processor family.

Support for TM2 is indicated by `CPUID.1:ECX.TM2[bit 8] = 1`.

15.8.2.3 Two Methods for Enabling TM2

On processors with CPUID family/model/stepping signature encoded as 0x69n or 0x6Dn (early Pentium M processors), TM2 is enabled if the `TM_SELECT` flag (bit 16) of the `MSR_THERM2_CTL` register is set to 1 (Figure 15-25) and bit 3 of the `IA32_MISC_ENABLE` register is set to 1.

Following a power-up or reset, the TM_SELECT flag may be cleared. BIOS is required to enable either TM1 or TM2. Operating systems and applications must not disable mechanisms that enable TM1 or TM2. If bit 3 of the IA32_MISC_ENABLE register is set and TM_SELECT flag of the MSR_THERM2_CTL register is cleared, TM1 is enabled.

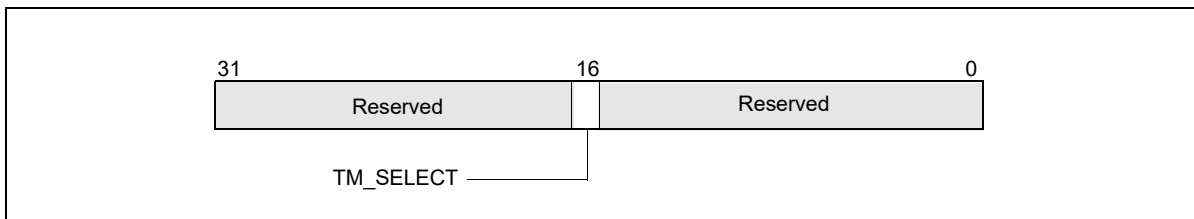


Figure 15-25. MSR_THERM2_CTL Register On Processors with CPUID Family/Model/Stepping Signature Encoded as 0x69n or 0x6Dn

On processors introduced after the Pentium 4 processor (this includes most Pentium M processors), the method used to enable TM2 is different. TM2 is enable by setting bit 13 of IA32_MISC_ENABLE register to 1. This applies to Intel Core Duo, Core Solo, and Intel Core 2 processor family.

The target operating frequency and voltage for the TM2 transition after TM2 is triggered is specified by the value written to MSR_THERM2_CTL, bits 15:0 (Figure 15-26). Following a power-up or reset, BIOS is required to enable at least one of these two thermal monitoring mechanisms. If both TM1 and TM2 are supported, BIOS may choose to enable TM2 instead of TM1. Operating systems and applications must not disable the mechanisms that enable TM1 or TM2; and they must not alter the value in bits 15:0 of the MSR_THERM2_CTL register.

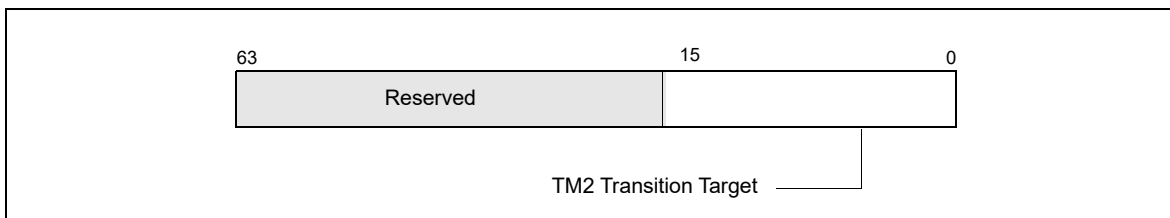


Figure 15-26. MSR_THERM2_CTL Register for Supporting TM2

15.8.2.4 Performance State Transitions and Thermal Monitoring

If the thermal control circuitry (TCC) for thermal monitor (TM1/TM2) is active, writes to the IA32_PERF_CTL will effect a new target operating point as follows:

- If TM1 is enabled and the TCC is engaged, the performance state transition can commence before the TCC is disengaged.
- If TM2 is enabled and the TCC is engaged, the performance state transition specified by a write to the IA32_PERF_CTL will commence after the TCC has disengaged.

15.8.2.5 Thermal Status Information

The status of the temperature sensor that triggers the thermal monitor (TM1/TM2) is indicated through the thermal status flag and thermal status log flag in the IA32_THERM_STATUS MSR (see Figure 15-27).

The functions of these flags are:

- **Thermal Status flag, bit 0** — When set, indicates that the processor core temperature is currently at the trip temperature of the thermal monitor and that the processor power consumption is being reduced via either TM1 or TM2, depending on which is enabled. When clear, the flag indicates that the core temperature is below the thermal monitor trip temperature. This flag is read only.

- **Thermal Status Log flag, bit 1** — When set, indicates that the thermal sensor has tripped since the last power-up or reset or since the last time that software cleared this flag. This flag is a sticky bit; once set it remains set until cleared by software or until a power-up or reset of the processor. The default state is clear.

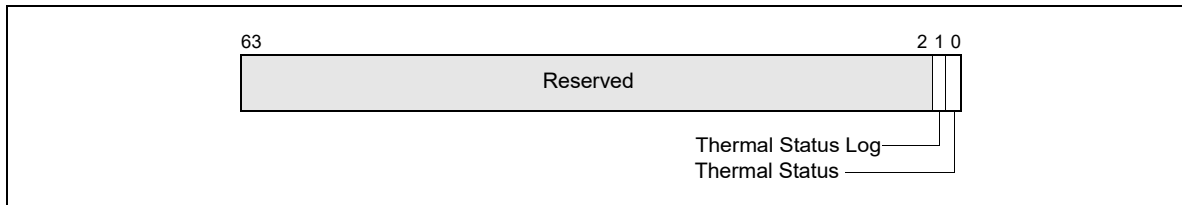


Figure 15-27. IA32_THERM_STATUS MSR

After the second temperature sensor has been tripped, the thermal monitor (TM1/TM2) will remain engaged for a minimum time period (on the order of 1 ms). The thermal monitor will remain engaged until the processor core temperature drops below the preset trip temperature of the temperature sensor, taking hysteresis into account.

While the processor is in a stop-clock state, interrupts will be blocked from interrupting the processor. This holding off of interrupts increases the interrupt latency, but does not cause interrupts to be lost. Outstanding interrupts remain pending until clock modulation is complete.

The thermal monitor can be programmed to generate an interrupt to the processor when the thermal sensor is tripped; this is called a thermal interrupt. The delivery mode, mask, and vector for this interrupt can be programmed through the thermal entry in the local APIC's LVT (see Section 11.5.1, "Local Vector Table"). The low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR (see Figure 15-28) control when the interrupt is generated; that is, on a transition from a temperature below the trip point to above and/or vice-versa.

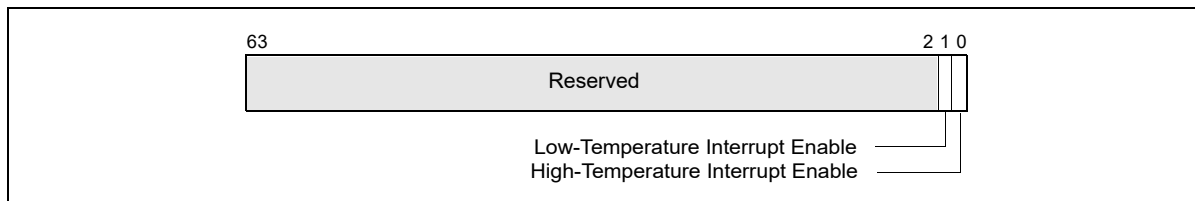


Figure 15-28. IA32_THERM_INTERRUPT MSR

- **High-Temperature Interrupt Enable flag, bit 0** — Enables an interrupt to be generated on the transition from a low-temperature to a high-temperature when set; disables the interrupt when clear.(R/W).
- **Low-Temperature Interrupt Enable flag, bit 1** — Enables an interrupt to be generated on the transition from a high-temperature to a low-temperature when set; disables the interrupt when clear.

The thermal interrupt can be masked by the thermal LVT entry. After a power-up or reset, the low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR are cleared (interrupts are disabled) and the thermal LVT entry is set to mask interrupts. This interrupt should be handled either by the operating system or system management mode (SMM) code.

Note that the operation of the thermal monitoring mechanism has no effect upon the clock rate of the processor's internal high-resolution timer (time stamp counter).

15.8.2.6 Adaptive Thermal Monitor

The Intel Core 2 Duo processor family supports enhanced thermal management mechanism, referred to as Adaptive Thermal Monitor (Adaptive TM).

Unlike TM2, Adaptive TM is not limited to one TM2 transition target. During a thermal trip event, Adaptive TM (if enabled) selects an optimal target operating point based on whether or not the current operating point has effectively cooled the processor.

Similar to TM2, Adaptive TM is enable by BIOS. The BIOS is required to test the TM1 and TM2 feature flags and enable all available thermal control mechanisms (including Adaptive TM) at platform initiation.

Adaptive TM is available only to a subset of processors that support TM2.

In each chip-multiprocessing (CMP) silicon die, each core has a unique thermal sensor that triggers independently. These thermal sensor can trigger TM1 or TM2 transitions in the same manner as described in Section 15.8.2.1 and Section 15.8.2.2. The trip point of the thermal sensor is not programmable by software since it is set during the fabrication of the processor.

Each thermal sensor in a processor core may be triggered independently to engage thermal management features. In Adaptive TM, both cores will transition to a lower frequency and/or lower voltage level if one sensor is triggered. Triggering of this sensor is visible to software via the thermal interrupt LVT entry in the local APIC of a given core.

15.8.3 Software Controlled Clock Modulation

Pentium 4, Intel Xeon and Pentium M processors also support software-controlled clock modulation. This provides a means for operating systems to implement a power management policy to reduce the power consumption of the processor. Here, the stop-clock duty cycle is controlled by software through the IA32_CLOCK_MODULATION MSR (see Figure 15-29).

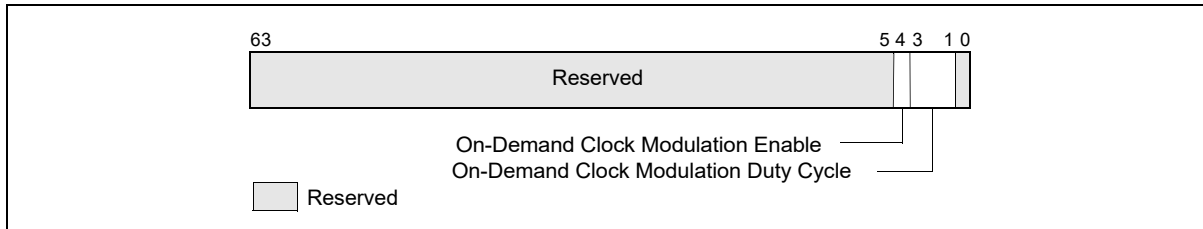


Figure 15-29. IA32_CLOCK_MODULATION MSR

The IA32_CLOCK_MODULATION MSR contains the following flag and field used to enable software-controlled clock modulation and to select the clock modulation duty cycle:

- **On-Demand Clock Modulation Enable, bit 4** — Enables on-demand software controlled clock modulation when set; disables software-controlled clock modulation when clear.
- **On-Demand Clock Modulation Duty Cycle, bits 1 through 3** — Selects the on-demand clock modulation duty cycle (see Table 15-11). This field is only active when the on-demand clock modulation enable flag is set.

Note that the on-demand clock modulation mechanism (like the thermal monitor) controls the processor’s stop-clock circuitry internally to modulate the clock signal. The STPCLK# pin is not used in this mechanism.

Table 15-11. On-Demand Clock Modulation Duty Cycle Field Encoding

Duty Cycle Field Encoding	Duty Cycle
000B	Reserved
001B	12.5% (Default)
010B	25.0%
011B	37.5%
100B	50.0%
101B	63.5%
110B	75%
111B	87.5%

The on-demand clock modulation mechanism can be used to control processor power consumption. Power management software can write to the IA32_CLOCK_MODULATION MSR to enable clock modulation and to select a modulation duty cycle. If on-demand clock modulation and TM1 are both enabled and the thermal status of the processor is hot (bit 0 of the IA32_THERM_STATUS MSR is set), clock modulation at the duty cycle specified by TM1 takes precedence, regardless of the setting of the on-demand clock modulation duty cycle.

For Hyper-Threading Technology enabled processors, the IA32_CLOCK_MODULATION register is duplicated for each logical processor. In order for the On-demand clock modulation feature to work properly, the feature must be enabled on all the logical processors within a physical processor. If the programmed duty cycle is not identical for all the logical processors, the processor core clock will modulate to the highest duty cycle programmed for processors with any of the following CPUID DisplayFamily_DisplayModel signatures (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*): 06_1A, 06_1C, 06_1E, 06_1F, 06_25, 06_26, 06_27, 06_2C, 06_2E, 06_2F, 06_35, 06_36, and 0F_xx. For all other processors, if the programmed duty cycle is not identical for all logical processors in the same core, the processor core will modulate at the lowest programmed duty cycle.

For multiple processor cores in a physical package, each processor core can modulate to a programmed duty cycle independently.

For the P6 family processors, on-demand clock modulation was implemented through the chipset, which controlled clock modulation through the processor's STPCLK# pin.

15.8.3.1 Extension of Software Controlled Clock Modulation

Extension of the software controlled clock modulation facility supports on-demand clock modulation duty cycle with 4-bit dynamic range (increased from 3-bit range). Granularity of clock modulation duty cycle is increased to 6.25% (compared to 12.5%).

Four bit dynamic range control is provided by using bit 0 in conjunction with bits 3:1 of the IA32_CLOCK_MODULATION MSR (see Figure 15-30).

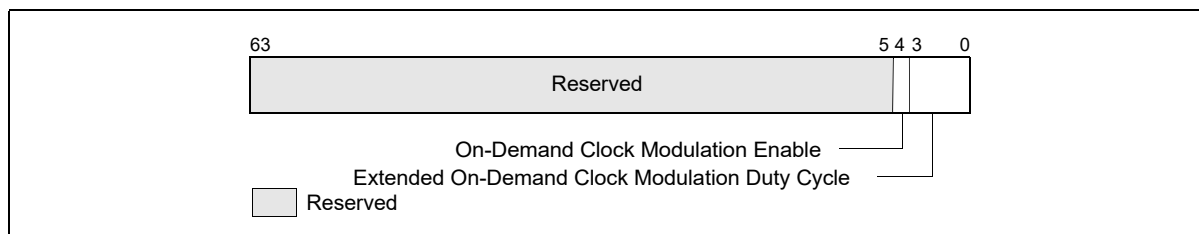


Figure 15-30. IA32_CLOCK_MODULATION MSR with Clock Modulation Extension

Extension to software controlled clock modulation is supported only if CPUID.06H:EAX[Bit 5] = 1. If CPUID.06H:EAX[Bit 5] = 0, then bit 0 of IA32_CLOCK_MODULATION is reserved.

15.8.4 Detection of Thermal Monitor and Software Controlled Clock Modulation Facilities

The ACPI flag (bit 22) of the CPUID feature flags indicates the presence of the IA32_THERM_STATUS, IA32_THERM_INTERRUPT, IA32_CLOCK_MODULATION MSRs, and the xAPIC thermal LVT entry.

The TM1 flag (bit 29) of the CPUID feature flags indicates the presence of the automatic thermal monitoring facilities that modulate clock duty cycles.

15.8.4.1 Detection of Software Controlled Clock Modulation Extension

Processor's support of software controlled clock modulation extension is indicated by CPUID.06H:EAX[Bit 5] = 1.

15.8.5 On Die Digital Thermal Sensors

On die digital thermal sensor can be read using an MSR (no I/O interface). In Intel Core Duo processors, each core has a unique digital sensor whose temperature is accessible using an MSR. The digital thermal sensor is the preferred method for reading the die temperature because (a) it is located closer to the hottest portions of the die, (b) it enables software to accurately track the die temperature and the potential activation of thermal throttling.

15.8.5.1 Digital Thermal Sensor Enumeration

The processor supports a digital thermal sensor if CPUID.06H.EAX[0] = 1. If the processor supports digital thermal sensor, EBX[bits 3:0] determine the number of thermal thresholds that are available for use.

Software sets thermal thresholds by using the IA32_THERM_INTERRUPT MSR. Software reads output of the digital thermal sensor using the IA32_THERM_STATUS MSR.

15.8.5.2 Reading the Digital Sensor

Unlike traditional analog thermal devices, the output of the digital thermal sensor is a temperature relative to the maximum supported operating temperature of the processor.

Temperature measurements returned by digital thermal sensors are always at or below TCC activation temperature. Critical temperature conditions are detected using the “Critical Temperature Status” bit. When this bit is set, the processor is operating at a critical temperature and immediate shutdown of the system should occur. Once the “Critical Temperature Status” bit is set, reliable operation is not guaranteed.

See Figure 15-31 for the layout of IA32_THERM_STATUS MSR. Bit fields include:

- **Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Thermal Status Log (bit 1, R/WCO)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#). Bit 1 = 1 if PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **PROCHOT# or FORCEPR# Event (bit 2, RO)** — Indicates whether PROCHOT# or FORCEPR# is being asserted by another agent on the platform.

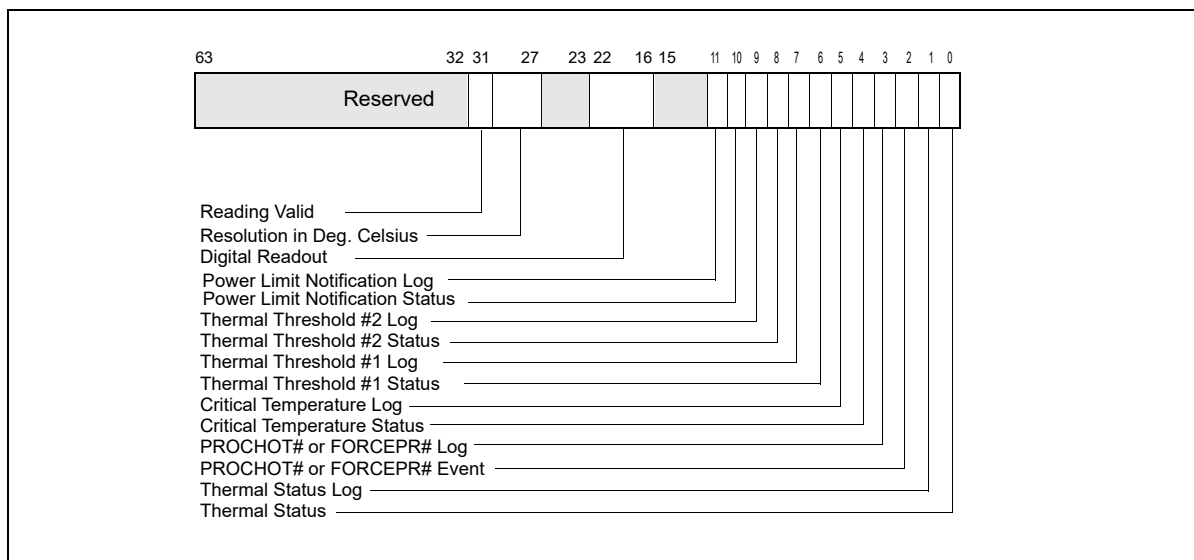


Figure 15-31. IA32_THERM_STATUS Register

- **PROCHOT# or FORCEPR# Log (bit 3, R/WC0)** — Sticky bit that indicates whether PROCHOT# or FORCEPR# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, PROCHOT# or FORCEPR# has been externally asserted. Software may clear this bit by writing a zero. External PROCHOT# assertions are only acknowledged if the Bidirectional Prochot feature is enabled.
- **Critical Temperature Status (bit 4, RO)** — Indicates whether the critical temperature detector output signal is currently active. If bit 4 = 1, the critical temperature detector output signal is currently active.
- **Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual temperature is currently higher than or equal to the value set in Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to TT#1. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.
- **Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual temperature is currently higher than or equal to the value set in Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to TT#2. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.
- **Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- **Power Limitation Status (bit 10, RO)** — Indicates whether the processor is currently operating below OS-requested P-state (specified in IA32_PERF_CTL) or OS-requested clock modulation duty cycle (specified in IA32_CLOCK_MODULATION). This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification can be delivered independently to IA32_PACKAGE_THERM_STATUS MSR.
- **Power Notification Log (bit 11, R/WC0)** — Sticky bit that indicates the processor went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET. This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification is indicated independently in IA32_PACKAGE_THERM_STATUS MSR.
- **Digital Readout (bits 22:16, RO)** — Digital temperature reading in 1 degree Celsius relative to the TCC activation temperature.
 - 0: TCC Activation temperature,
 - 1: (TCC Activation - 1) , etc. See the processor's data sheet for details regarding TCC activation.
 A lower reading in the Digital Readout field (bits 22:16) indicates a higher actual temperature.
- **Resolution in Degrees Celsius (bits 30:27, RO)** — Specifies the resolution (or tolerance) of the digital thermal sensor. The value is in degrees Celsius. It is recommended that new threshold values be offset from the current temperature by at least the resolution + 1 in order to avoid hysteresis of interrupt generation.
- **Reading Valid (bit 31, RO)** — Indicates if the digital readout in bits 22:16 is valid. The readout is valid if bit 31 = 1.

Changes to temperature can be detected using two thresholds (see Figure 15-32); one is set above and the other below the current temperature. These thresholds have the capability of generating interrupts using the core's local APIC which software must then service. Note that the local APIC entries used by these thresholds are also used by the Intel® Thermal Monitor; it is up to software to determine the source of a specific interrupt.

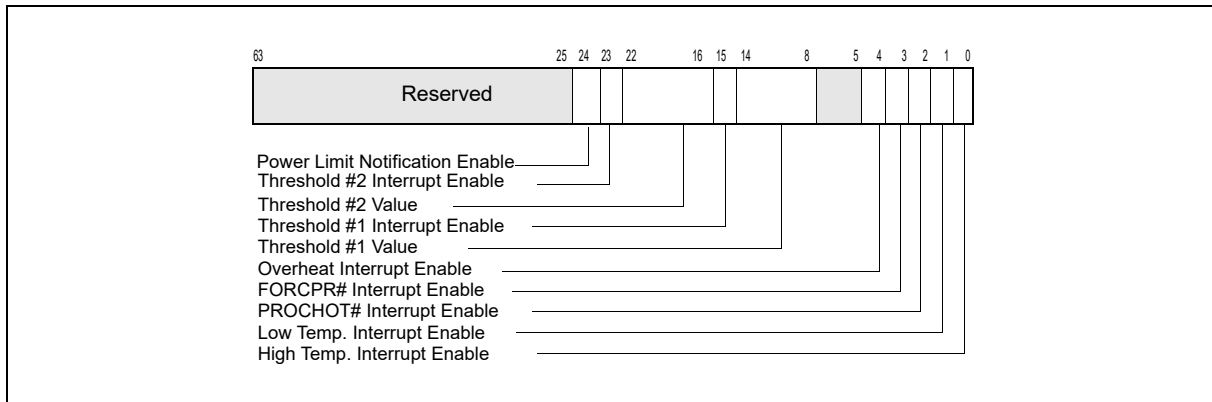


Figure 15-32. IA32_THERM_INTERRUPT Register

See Figure 15-32 for the layout of IA32_THERM_INTERRUPT MSR. Bit fields include:

- **High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.
- **PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **FORCEPR# Interrupt Enable (bit 3, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when FORCEPR# has been asserted by another agent on the platform. Bit 3 = 0 disables the interrupt; bit 3 = 1 enables the interrupt.
- **Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #1 Status and Log bits as well as the Threshold #1 thermal interrupt delivery.
- **Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Threshold #2 Value (bits 22:16, R/W)** — A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #2 Status and Log bits as well as the Threshold #2 thermal interrupt delivery.
- **Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #2 setting in any direction. Bit 23 = 1 enables the interrupt; bit 23 = 0 disables the interrupt.
- **Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of power notification events when the processor went below OS-requested P-state or OS-requested clock modulation duty cycle. This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification can be enabled independently by IA32_PACKAGE_THERM_INTERRUPT MSR.

15.8.6 Power Limit Notification

Platform firmware may be capable of specifying a power limit to restrict power delivered to a platform component, such as a physical processor package. This constraint imposed by platform firmware may occasionally cause the processor to operate below OS-requested P or T-state. A power limit notification event can be delivered using the existing thermal LVT entry in the local APIC.

Software can enumerate the presence of the processor's support for power limit notification by verifying `CPUID.06H:EAX[bit 4] = 1`.

If `CPUID.06H:EAX[bit 4] = 1`, then `IA32_THERM_INTERRUPT` and `IA32_THERM_STATUS` provides the following facility to manage power limit notification:

- Bits 10 and 11 in `IA32_THERM_STATUS` informs software of the occurrence of processor operating below OS-requested P-state or clock modulation duty cycle setting (see Figure 15-31).
- Bit 24 in `IA32_THERM_INTERRUPT` enables the local APIC to deliver a thermal event when the processor went below OS-requested P-state or clock modulation duty cycle setting (see Figure 15-32).

15.9 PACKAGE LEVEL THERMAL MANAGEMENT

The thermal management facilities like `IA32_THERM_INTERRUPT` and `IA32_THERM_STATUS` are often implemented with a processor core granularity. To facilitate software manage thermal events from a package level granularity, two architectural MSR is provided for package level thermal management. The `IA32_PACKAGE_THERM_STATUS` and `IA32_PACKAGE_THERM_INTERRUPT` MSRs use similar interfaces as `IA32_THERM_STATUS` and `IA32_THERM_INTERRUPT`, but are shared in each physical processor package.

Software can enumerate the presence of the processor's support for package level thermal management facility (`IA32_PACKAGE_THERM_STATUS` and `IA32_PACKAGE_THERM_INTERRUPT`) by verifying `CPUID.06H:EAX[bit 6] = 1`.

The layout of `IA32_PACKAGE_THERM_STATUS` MSR is shown in Figure 15-33.

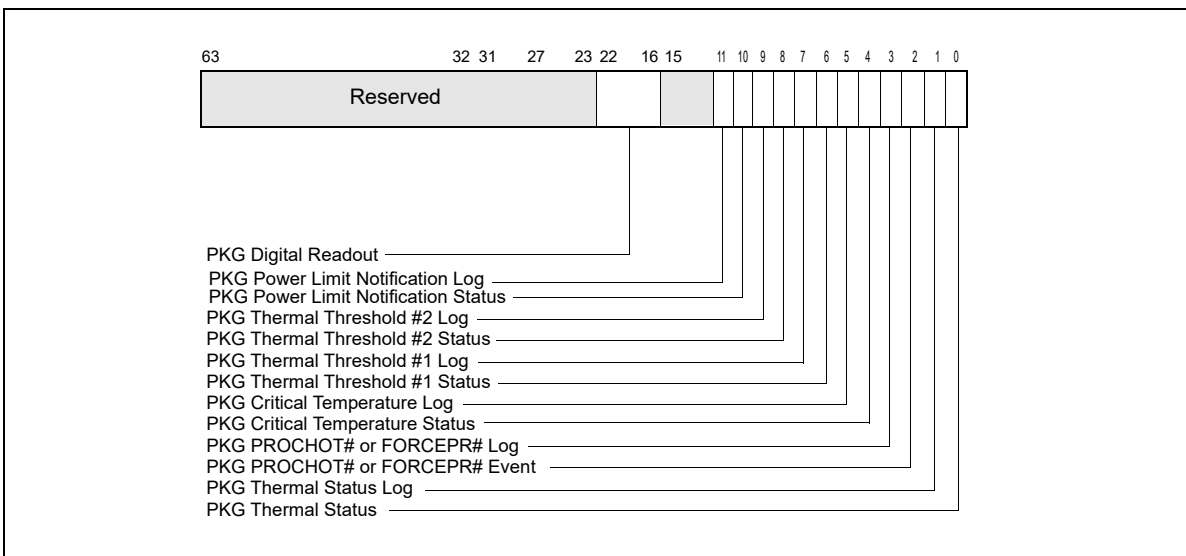


Figure 15-33. `IA32_PACKAGE_THERM_STATUS` Register

- **Package Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (`PROCHOT#`) for the package is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.

- **Package Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#) of the package. Bit 1 = 1 if package PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **Package PROCHOT# Event (bit 2, RO)** — Indicates whether package PROCHOT# is being asserted by another agent on the platform.
- **Package PROCHOT# Log (bit 3, R/WC0)** — Sticky bit that indicates whether package PROCHOT# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, package PROCHOT# has been externally asserted. Software may clear this bit by writing a zero.
- **Package Critical Temperature Status (bit 4, RO)** — Indicates whether the package critical temperature detector output signal is currently active. If bit 4 = 1, the package critical temperature detector output signal is currently active.
- **Package Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the package critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to PTT#1. Quantitative information of actual package temperature can be inferred from Package Digital Readout, bits 22:16.
- **Package Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Package Thermal Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to PTT#2. Quantitative information of actual temperature can be inferred from Package Digital Readout, bits 22:16.
- **Package Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Package Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- **Package Power Limitation Status (bit 10, RO)** — Indicates package power limit is forcing one or more processors to operate below OS-requested P-state. Note that package power limit violation may be caused by processor cores or by devices residing in the uncore. Software can examine IA32_THERM_STATUS to determine if the cause originates from a processor core (see Figure 15-31).
- **Package Power Notification Log (bit 11, R/WC0)** — Sticky bit that indicates any processor in the package went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET.
- **Package Digital Readout (bits 22:16, RO)** — Package digital temperature reading in 1 degree Celsius relative to the package TCC activation temperature.
 - 0: Package TCC Activation temperature,
 - 1: (PTCC Activation - 1) , etc. See the processor's data sheet for details regarding PTCC activation.
 A lower reading in the Package Digital Readout field (bits 22:16) indicates a higher actual temperature.

The layout of IA32_PACKAGE_THERM_INTERRUPT MSR is shown in Figure 15-34.

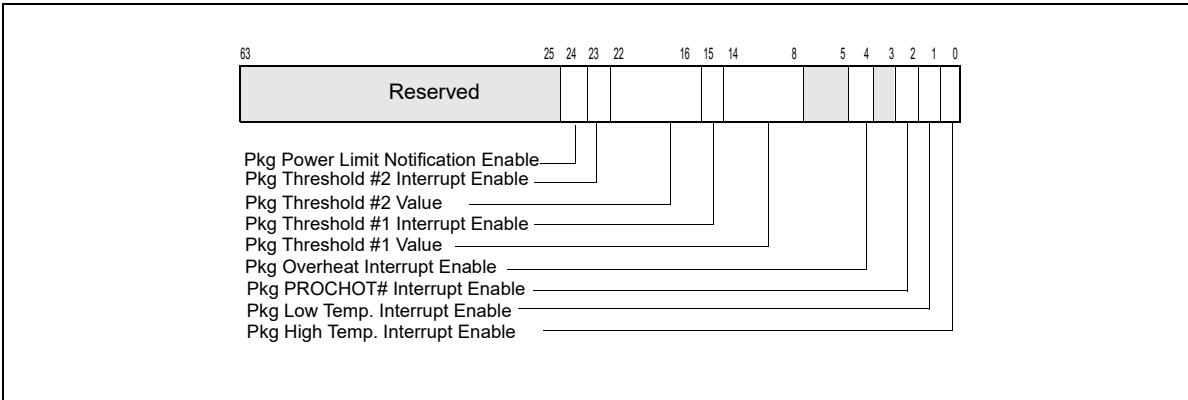


Figure 15-34. IA32_PACKAGE_THERM_INTERRUPT Register

- **Package High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a package high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Package Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.
- **Package PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when Package PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **Package Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Package Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Package Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the Package TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Package Digital Readout and is used to generate the Package Thermal Threshold #1 Status and Log bits as well as the Package Threshold #1 thermal interrupt delivery.
- **Package Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Package Threshold #2 Value (bits 22:16, R/W)** — A temperature threshold, encoded relative to the PTCC Activation temperature (using the same format as the Package Digital Readout). This threshold is compared against the Package Digital Readout and is used to generate the Package Thermal Threshold #2 Status and Log bits as well as the Package Threshold #2 thermal interrupt delivery.
- **Package Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #2 setting in any direction. Bit 23 = 1 enables the interrupt; bit 23 = 0 disables the interrupt.
- **Package Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of package power notification events.

15.9.1 Support for Passive and Active cooling

Passive and active cooling may be controlled by the OS power management agent through ACPI control methods. On platforms providing package level thermal management facility described in the previous section, it is recommended that active cooling (FAN control) should be driven by measuring the package temperature using the IA32_PACKAGE_THERM_INTERRUPT MSR.

Passive cooling (frequency throttling) should be driven by measuring (a) the core and package temperatures, or (b) only the package temperature. If measured package temperature led the power management agent to choose which core to execute passive cooling, then all cores need to execute passive cooling. Core temperature is measured using the IA32_THERMAL_STATUS and IA32_THERMAL_INTERRUPT MSRs. The exact implementation details depend on the platform firmware and possible solutions include defining two different thermal zones (one for core temperature and passive cooling and the other for package temperature and active cooling).

15.10 PLATFORM SPECIFIC POWER MANAGEMENT SUPPORT

This section covers power management interfaces that are not architectural but addresses the power management needs of several platform specific components. Specifically, RAPL (Running Average Power Limit) interfaces provide mechanisms to enforce power consumption limit. Power limiting usages have specific usages in client and server platforms.

For client platform power limit control and for server platforms used in a data center, the following power and thermal related usages are desirable:

- Platform Thermal Management: Robust mechanisms to manage component, platform, and group-level thermals, either proactively or reactively (e.g., in response to a platform-level thermal trip point).
- Platform Power Limiting: More deterministic control over the system's power consumption, for example to meet battery life targets on rack-level or container-level power consumption goals within a datacenter.
- Power/Performance Budgeting: Efficient means to control the power consumed (and therefore the sustained performance delivered) within and across platforms.

The server and client usage models are addressed by RAPL interfaces, which expose multiple domains of power rationing within each processor socket. Generally, these RAPL domains may be viewed to include hierarchically:

- Package domain is the processor die.
- Memory domain includes the directly-attached DRAM; an additional power plane may constitute a separate domain.

In order to manage the power consumed across multiple sockets via RAPL, individual limits must be programmed for each processor complex. Programming specific RAPL domain across multiple sockets is not supported.

15.10.1 RAPL Interfaces

RAPL interfaces consist of non-architectural MSRs. Each RAPL domain supports the following set of capabilities, some of which are optional as stated below.

- Power limit - MSR interfaces to specify power limit, time window; lock bit, clamp bit etc.
- Energy Status - Power metering interface providing energy consumption information.
- Perf Status (Optional) - Interface providing information on the performance effects (regression) due to power limits. It is defined as a duration metric that measures the power limit effect in the respective domain. The meaning of duration is domain specific.
- Power Info (Optional) - Interface providing information on the range of parameters for a given domain, minimum power, maximum power etc.
- Policy (Optional) - 4-bit priority information that is a hint to hardware for dividing budget between sub-domains in a parent domain.

Each of the above capabilities requires specific units in order to describe them. Power is expressed in Watts, Time is expressed in Seconds, and Energy is expressed in Joules. Scaling factors are supplied to each unit to make the information presented meaningful in a finite number of bits. Units for power, energy, and time are exposed in the read-only MSR_RAPL_POWER_UNIT MSR.

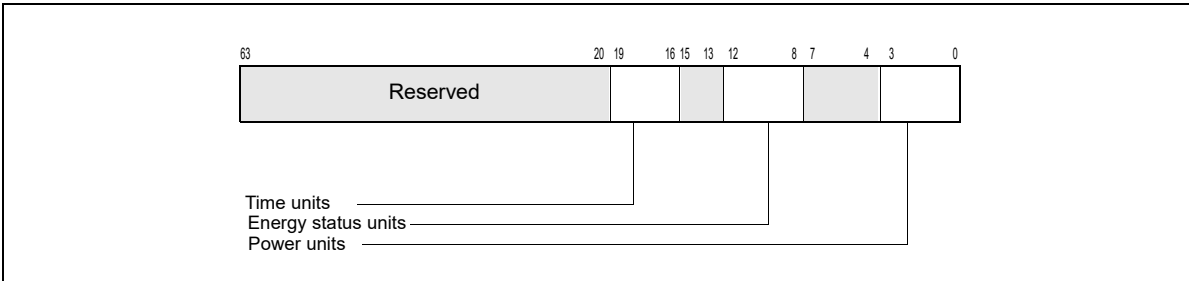


Figure 15-35. MSR_RAPL_POWER_UNIT Register

MSR_RAPL_POWER_UNIT (Figure 15-35) provides the following information across all RAPL domains:

- **Power Units** (bits 3:0): Power related information (in Watts) is based on the multiplier, $1/2^{\text{PU}}$; where PU is an unsigned integer represented by bits 3:0. Default value is 0011b, indicating power unit is in 1/8 Watts increment.
- **Energy Status Units** (bits 12:8): Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 10000b, indicating energy status unit is in 15.3 micro-Joules increment.
- **Time Units** (bits 19:16): Time related information (in Seconds) is based on the multiplier, $1/2^{\text{TU}}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating time unit is in 976 micro-seconds increment.

15.10.2 RAPL Domains and Platform Specificity

The specific RAPL domains available in a platform vary across product segments. Platforms targeting the client segment support the following RAPL domain hierarchy:

- Package
- Two power planes: PP0 and PP1 (PP1 may reflect to uncore devices)

Platforms targeting the server segment support the following RAPL domain hierarchy:

- Package
- Power plane: PP0
- DRAM

Each level of the RAPL hierarchy provides a respective set of RAPL interface MSRs. Table 15-12 lists the RAPL MSR interfaces available for each RAPL domain. The power limit MSR of each RAPL domain is located at offset 0 relative to an MSR base address which is non-architectural (see Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4). The energy status MSR of each domain is located at offset 1 relative to the MSR base address of respective domain.

Table 15-12. RAPL MSR Interfaces and RAPL Domains

Domain	Power Limit (Offset 0)	Energy Status (Offset 1)	Policy (Offset 2)	Perf Status (Offset 3)	Power Info (Offset 4)
PKG	MSR_PKG_POWER_LIMIT	MSR_PKG_ENERGY_STATUS	RESERVED	MSR_PKG_PERF_STATUS	MSR_PKG_POWER_INFO
DRAM	MSR_DRAM_POWER_LIMIT	MSR_DRAM_ENERGY_STATUS	RESERVED	MSR_DRAM_PERF_STATUS	MSR_DRAM_POWER_INFO
PP0	MSR_PP0_POWER_LIMIT	MSR_PP0_ENERGY_STATUS	MSR_PP0_POLICY	MSR_PP0_PERF_STATUS	RESERVED
PP1	MSR_PP1_POWER_LIMIT	MSR_PP1_ENERGY_STATUS	MSR_PP1_POLICY	RESERVED	RESERVED

The presence of the optional MSR interfaces (the three right-most columns of Table 15-12) may be model-specific. See Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4 for details.

15.10.3 Package RAPL Domain

The MSR interfaces defined for the package RAPL domain are:

- MSR_PKG_POWER_LIMIT allows software to set power limits for the package and measurement attributes associated with each limit,
- MSR_PKG_ENERGY_STATUS reports measured actual energy usage,
- MSR_PKG_POWER_INFO reports the package power range information for RAPL usage.

MSR_PKG_PERF_STATUS can report the performance impact of power limiting, but its availability may be model-specific.

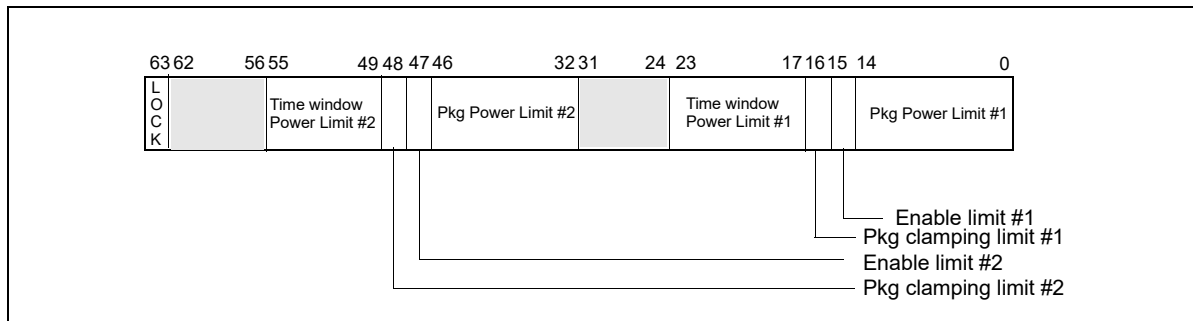


Figure 15-36. MSR_PKG_POWER_LIMIT Register

MSR_PKG_POWER_LIMIT allows a software agent to define power limitation for the package domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_PKG_POWER_LIMIT. Two power limits can be specified, corresponding to time windows of different sizes. Each power limit provides independent clamping control that would permit the processor cores to go below OS-requested state to meet the power limits. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_PKG_POWER_LIMIT (Figure 15-36) are:

- **Package Power Limit #1** (bits 14:0): Sets the average power usage limit of the package domain corresponding to time window # 1. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.

- **Enable Power Limit #1**(bit 15): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #1** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #1** (bits 23:17): Indicates the time window for power limit #1

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time_Unit}$$
 Here "Y" is the unsigned integer value represented. by bits 21:17, "Z" is an unsigned integer represented by bits 23:22. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Package Power Limit #2**(bits 46:32): Sets the average power usage limit of the package domain corresponding to time window # 2. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #2**(bit 47): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #2** (bit 48): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #2** (bits 55:49): Indicates the time window for power limit #2

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time_Unit}$$
 Here "Y" is the unsigned integer value represented. by bits 53:49, "Z" is an unsigned integer represented by bits 55:54. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT. This field may have a hard-coded value in hardware and ignores values written by software.
- **Lock** (bit 63): If set, all write attempts to this MSR are ignored until next RESET.

MSR_PKG_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the package domain. This MSR is updated every ~1msec. It has a wraparound time of around 60 secs when power consumption is high, and may be longer otherwise.

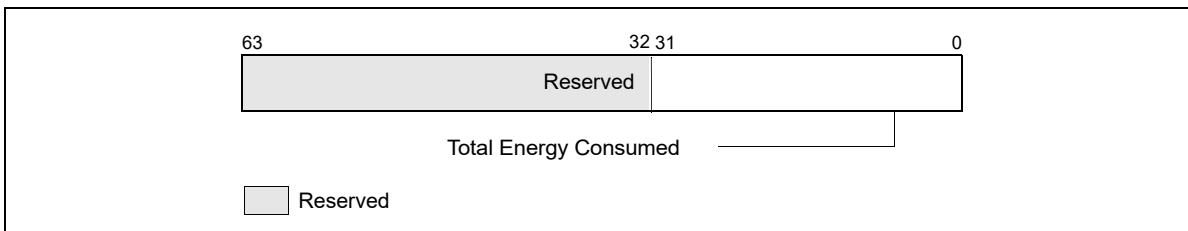


Figure 15-37. MSR_PKG_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PKG_POWER_INFO is a read-only MSR. It reports the package power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the package domain. It also provides the largest possible time window for software to program the RAPL interface.

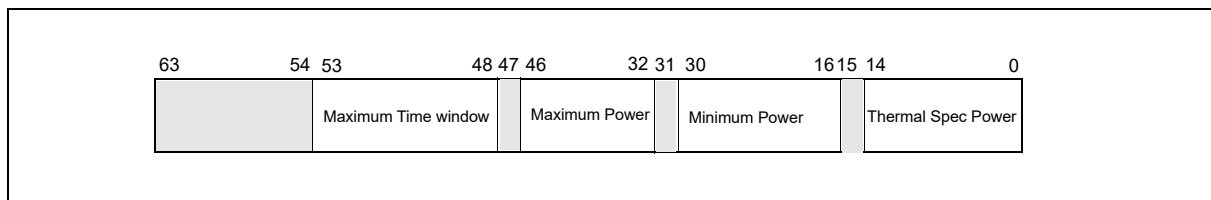


Figure 15-38. MSR_PKG_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.

- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_PKG_POWER_LIMIT. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

MSR_PKG_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific (see Chapter 2, "Model-Specific Registers (MSRs)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4).

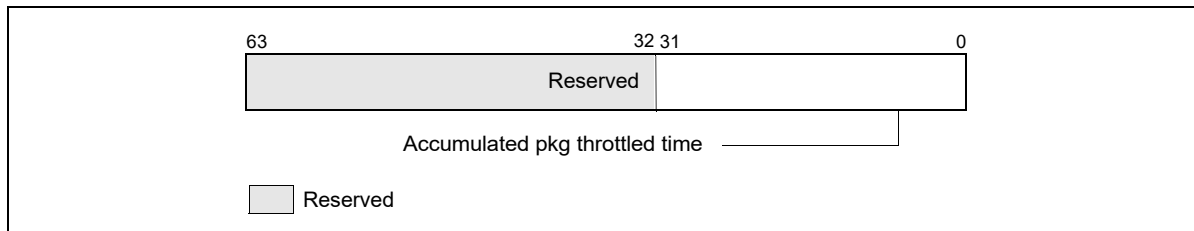


Figure 15-39. MSR_PKG_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the package has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

15.10.4 PP0/PP1 RAPL Domains

The MSR interfaces defined for the PP0 and PP1 domains are identical in layout. Generally, PP0 refers to the processor cores. The availability of PP1 RAPL domain interface is platform-specific. For a client platform, the PP1 domain refers to the power plane of a specific device in the uncore. For server platforms, the PP1 domain is not supported, but its PP0 domain supports the MSR_PP0_PERF_STATUS interface.

- MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow software to set power limits for the respective power plane domain.
- MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS report actual energy usage on a power plane.
- MSR_PP0_POLICY/MSR_PP1_POLICY allow software to adjust balance for respective power plane.

MSR_PP0_PERF_STATUS can report the performance impact of power limiting, but it is not available in client platforms.

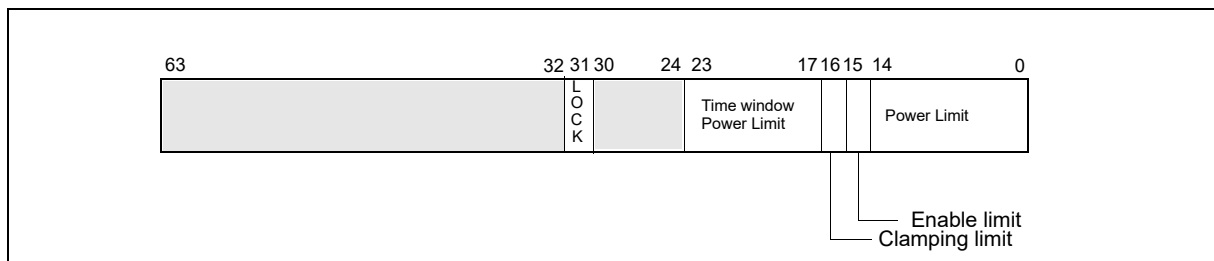


Figure 15-40. MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT Register

MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow a software agent to define power limitation for the respective power plane domain. A lock mechanism in each power plane domain allows the software agent to enforce power limit settings independently. Once a lock bit is set, the power limit settings in that power plane are static and un-modifiable until next RESET.

The bit fields of MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT (Figure 15-40) are:

- **Power Limit** (bits 14:0): Sets the average power usage limit of the respective power plane domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit** (bit 15): 0 = disabled; 1 = enabled.
- **Clamping Limitation** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit #1 will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to the MSR and corresponding policy MSR_PP0_POLICY/MSR_PP1_POLICY are ignored until next RESET.

MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS are read-only MSRs. They report the actual energy use for the respective power plane domains. These MSRs are updated every ~ 1 msec.

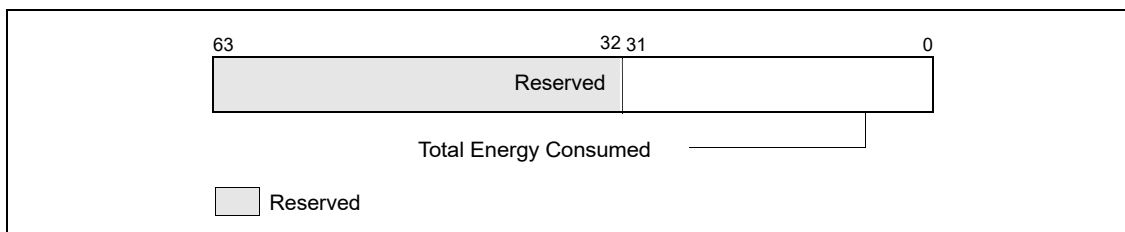


Figure 15-41. MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since the last time this register was cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PP0_POLICY/MSR_PP1_POLICY provide balance power policy control for each power plane by providing inputs to the power budgeting management algorithm. On platforms that support PP0 (IA cores) and PP1 (uncore graphic device), the default values give priority to the non-IA power plane. These MSRs enable the PCU to balance power consumption between the IA cores and uncore graphic device.

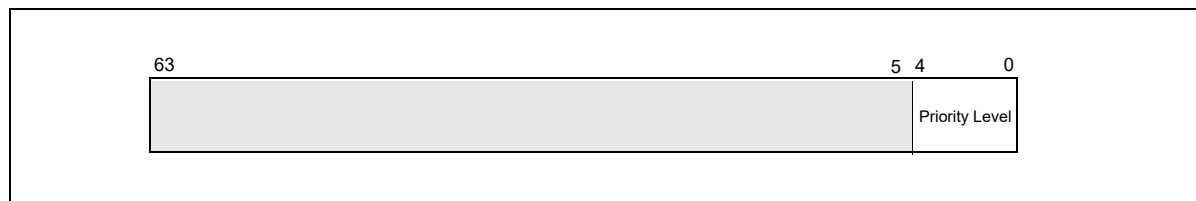


Figure 15-42. MSR_PP0_POLICY/MSR_PP1_POLICY Register

- **Priority Level** (bits 4:0): Priority level input to the PCU for respective power plane. PP0 covers the IA processor cores, PP1 covers the uncore graphic device. The value 31 is considered highest priority.

MSR_PP0_PERF_STATUS is a read-only MSR. It reports the total time for which the PP0 domain was throttled due to the power limits. This MSR is supported only in server platform. Throttling in this context is defined as going below the OS-requested P-state or T-state.

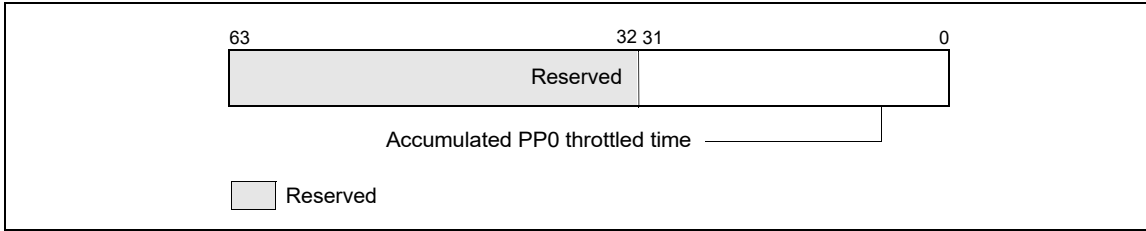


Figure 15-43. MSR_PPO_PERF_STATUS MSR

- **Accumulated PPO Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the PPO domain has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

15.10.5 DRAM RAPL Domain

The MSR interfaces defined for the DRAM domains are supported only in the server platform. The MSR interfaces are:

- MSR_DRAM_POWER_LIMIT allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.
- MSR_DRAM_ENERGY_STATUS reports measured actual energy usage.
- MSR_DRAM_POWER_INFO reports the DRAM domain power range information for RAPL usage.
- MSR_DRAM_PERF_STATUS can report the performance impact of power limiting.

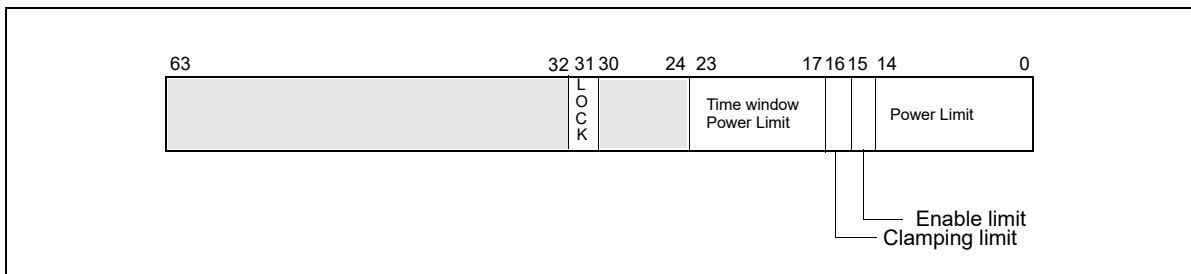


Figure 15-44. MSR_DRAM_POWER_LIMIT Register

MSR_DRAM_POWER_LIMIT allows a software agent to define power limitation for the DRAM domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_DRAM_POWER_LIMIT. A power limit can be specified along with a time window. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_DRAM_POWER_LIMIT (Figure 15-44) are:

- **DRAM Power Limit #1**(bits 14:0): Sets the average power usage limit of the DRAM domain corresponding to time window # 1. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #1**(bit 15): 0 = disabled; 1 = enabled.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to this MSR are ignored until next RESET.

MSR_DRAM_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the DRAM domain. This MSR is updated every ~1msec.

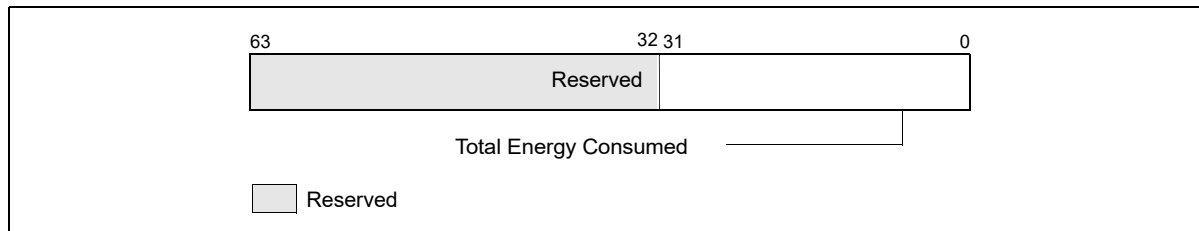


Figure 15-45. MSR_DRAM_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_POWER_INFO is a read-only MSR. It reports the DRAM power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the DRAM domain. It also provides the largest possible time window for software to program the RAPL interface.

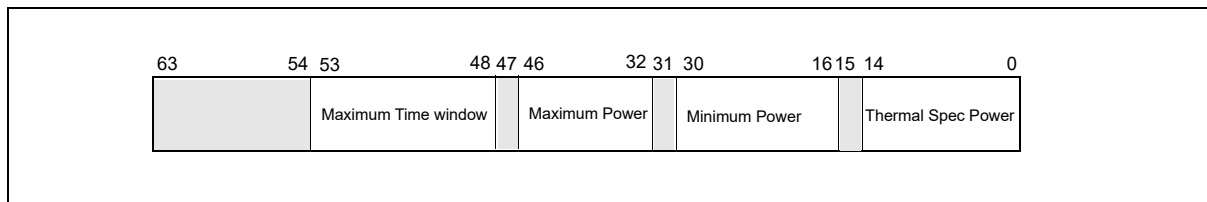


Figure 15-46. MSR_DRAM_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_DRAM_POWER_LIMIT. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific (see Chapter 2, "Model-Specific Registers (MSRs)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4).

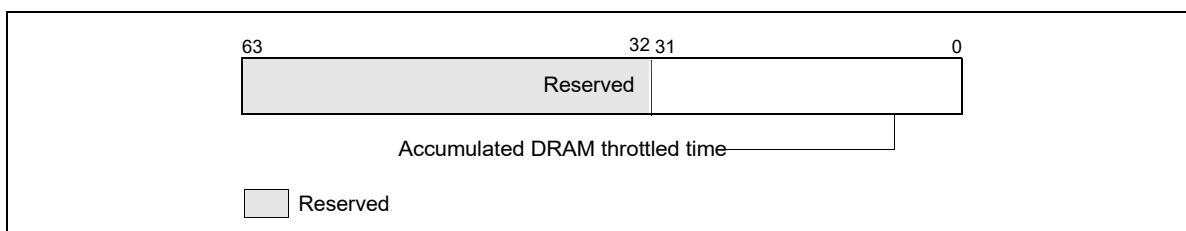


Figure 15-47. MSR_DRAM_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the DRAM domain has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

9. Updates to Chapter 17, Volume 3B

Change bars and green text show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- Updated Section 17.13, "Incremental Decoding Information: 4th Generation Intel® Xeon® Scalable Processor Family, Machine Error Codes For Machine Check," with machine check corrections as necessary.

CHAPTER 17

INTERPRETING MACHINE CHECK ERROR CODES

Encoding of the model-specific and other information fields is different across processor families. The differences are documented in the following sections.

17.1 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY 06H, MACHINE ERROR CODES FOR MACHINE CHECK

This section provides information for interpreting additional model-specific fields for external bus errors relating to processor family 06H. The references to processor family 06H refers to only IA-32 processors with CPUID signatures listed in Table 17-1.

Table 17-1. CPUID DisplayFamily_DisplayModel Signatures for Processor Family 06H

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_0EH	Intel® Core™ Duo processor, Intel® Core™ Solo processor
06_0DH	Intel Pentium M processor
06_09H	Intel Pentium M processor
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon Processor, Intel Pentium III Processor
06_03H, 06_05H	Intel Pentium II Xeon Processor, Intel Pentium II Processor
06_01H	Intel Pentium Pro Processor

These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally as compound errors with a general form of **0000 1PPT RRRR IILL** in the MCA error code field. See Chapter 16 for information on the interpretation of compound error codes. Incremental decoding information is listed in Table 17-2.

Table 17-2. Incremental Decoding Information: Processor Family 06H Machine Error Codes for Machine Check

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0		
Model Specific Errors	18:16	Reserved	Reserved
	24:19	Bus Queue Request Type	000000: BQ_DCU_READ_TYPE error. 000010: BQ_IFU_DEMAND_TYPE error. 000011: BQ_IFU_DEMAND_NC_TYPE error. 000100: BQ_DCU_RFO_TYPE error. 000101: BQ_DCU_RFO_LOCK_TYPE error. 000110: BQ_DCU_ITOM_TYPE error. 001000: BQ_DCU_WB_TYPE error. 001010: BQ_DCU_WCEVICT_TYPE error. 001011: BQ_DCU_WCLINE_TYPE error. 001100: BQ_DCU_BTM_TYPE error.

Table 17-2. Incremental Decoding Information: Processor Family 06H Machine Error Codes for Machine Check

Type	Bit No.	Bit Function	Bit Description
			001101: BQ_DCU_INTACK_TYPE error. 001110: BQ_DCU_INVALL2_TYPE error. 001111: BQ_DCU_FLUSH2_TYPE error. 010000: BQ_DCU_PART_RD_TYPE error. 010010: BQ_DCU_PART_WR_TYPE error. 010100: BQ_DCU_SPEC_CYC_TYPE error. 011000: BQ_DCU_IO_RD_TYPE error. 011001: BQ_DCU_IO_WR_TYPE error. 011100: BQ_DCU_LOCK_RD_TYPE error. 011110: BQ_DCU_SPLock_RD_TYPE error. 011101: BQ_DCU_LOCK_WR_TYPE error.
	27:25	Bus Queue Error Type	000: BQ_ERR_HARD_TYPE error. 001: BQ_ERR_DOUBLE_TYPE error. 010: BQ_ERR_AERR2_TYPE error. 100: BQ_ERR_SINGLE_TYPE error. 101: BQ_ERR_AERR1_TYPE error.
	28	FRC Error	1 if FRC error active.
	29	BERR	1 if BERR is driven.
	30	Internal BINIT	1 if BINIT driven for this processor.
	31	Reserved	Reserved
Other Information	34:32	Reserved	Reserved
	35	External BINIT	1 if BINIT is received from external bus.
	36	Response Parity Error	This bit is asserted in IA32_MCI_STATUS if this component has received a parity error on the RS[2:0]# pins for a response transaction. The RS signals are checked by the RSP# external pin.
	37	Bus BINIT	This bit is asserted in IA32_MCI_STATUS if this component has received a hard error response on a split transaction one access that has needed to be split across the 64-bit external bus interface into two accesses).
	38	Timeout BINIT	This bit is asserted in IA32_MCI_STATUS if this component has experienced a ROB time-out, which indicates that no micro-instruction has been retired for a predetermined period of time. A ROB time-out occurs when the 15-bit ROB time-out counter carries a 1 out of its high order bit. ² The timer is cleared when a micro-instruction retires, an exception is detected by the core processor, RESET is asserted, or when a ROB BINIT occurs. The ROB time-out counter is prescaled by the 8-bit PIC timer which is a divide by 128 of the bus clock (the bus clock is 1:2, 1:3, 1:4 of the core clock ³). When a carry out of the 8-bit PIC timer occurs, the ROB counter counts up by one. While this bit is asserted, it cannot be overwritten by another error.
	41:39	Reserved	Reserved
	42	Hard Error	This bit is asserted in IA32_MCI_STATUS if this component has initiated a bus transactions which has received a hard error response. While this bit is asserted, it cannot be overwritten.
	43	IERR	This bit is asserted in IA32_MCI_STATUS if this component has experienced a failure that causes the IERR pin to be asserted. While this bit is asserted, it cannot be overwritten.

Table 17-2. Incremental Decoding Information: Processor Family 06H Machine Error Codes for Machine Check

Type	Bit No.	Bit Function	Bit Description
	44	AERR	This bit is asserted in IA32_MCi_STATUS if this component has initiated 2 failing bus transactions which have failed due to Address Parity Errors (AERR asserted). While this bit is asserted, it cannot be overwritten.
	45	UECC	The Uncorrectable ECC error bit is asserted in IA32_MCi_STATUS for uncorrected ECC errors. While this bit is asserted, the ECC syndrome field will not be overwritten.
	46	CECC	The correctable ECC error bit is asserted in IA32_MCi_STATUS for corrected ECC errors.
	54:47	ECC Syndrome	The ECC syndrome field in IA32_MCi_STATUS contains the 8-bit ECC syndrome only if the error was a correctable/uncorrectable ECC error and there wasn't a previous valid ECC error syndrome logged in IA32_MCi_STATUS. A previous valid ECC error in IA32_MCi_STATUS is indicated by IA32_MCi_STATUS.bit45 (uncorrectable error occurred) being asserted. After processing an ECC error, machine check handling software should clear IA32_MCi_STATUS.bit45 so that future ECC error syndromes can be logged.
	56:55	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.
2. For processors with a CPUID signature of 06_0EH, a ROB time-out occurs when the 23-bit ROB time-out counter carries a 1 out of its high order bit.
3. For processors with a CPUID signature of 6_06_60H and later, the PIC timer will count crystal clock cycles.

17.2 INCREMENTAL DECODING INFORMATION: INTEL® CORE™ 2 PROCESSOR FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

Table 17-4 provides information for interpreting additional model-specific fields for external bus errors relating to processors based on Intel® Core™ microarchitecture, which implements the P4 bus specification. Table 17-3 lists the CPUID signatures for Intel 64 processors that are covered by Table 17-4. These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally as compound errors with a general form of **0000 1PPT RRRR IILL** in the MCA error code field. See Chapter 16 for information on the interpretation of compound error codes.

Table 17-3. CPUID DisplayFamily_DisplayModel Signatures for Processors Based on Intel® Core™ Microarchitecture

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_1DH	Intel® Xeon® Processor 7400 series
06_17H	Intel® Xeon® Processor 5200, 5400 series, Intel® Core™ 2 Quad processor Q9650
06_0FH	Intel® Xeon® Processor 3000, 3200, 5100, 5300, 7300 series, Intel® Core™ 2 Quad, Intel® Core™ 2 Extreme, Intel® Core™ 2 Duo processors, Intel Pentium dual-core processors

**Table 17-4. Incremental Bus Error Codes of Machine Check for Processors
Based on Intel® Core™ Microarchitecture**

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0		
Model Specific Errors	18:16	Reserved	Reserved
	24:19	Bus Queue Request Type	'000001: BQ_PREF_READ_TYPE error. '000000: BQ_DCU_READ_TYPE error. '000010: BQ_IFU_DEMAND_TYPE error '000011: BQ_IFU_DEMAND_NC_TYPE error. '000100: BQ_DCU_RFO_TYPE error. '000101: BQ_DCU_RFO_LOCK_TYPE error. '000110: BQ_DCU_ITOM_TYPE error. '001000: BQ_DCU_WB_TYPE error. '001010: BQ_DCU_WCEVICT_TYPE error. '001011: BQ_DCU_WCLINE_TYPE error. '001100: BQ_DCU_BTM_TYPE error. '001101: BQ_DCU_INTACK_TYPE error. '001110: BQ_DCU_INVALL2_TYPE error. '001111: BQ_DCU_FLUSHL2_TYPE error. '010000: BQ_DCU_PART_RD_TYPE error. '010010: BQ_DCU_PART_WR_TYPE error. '010100: BQ_DCU_SPEC_CYC_TYPE error. '011000: BQ_DCU_IO_RD_TYPE error. '011001: BQ_DCU_IO_WR_TYPE error. '011100: BQ_DCU_LOCK_RD_TYPE error. '011110: BQ_DCU_SPLOCK_RD_TYPE error. '011101: BQ_DCU_LOCK_WR_TYPE error. '100100: BQ_L2_WI_RFO_TYPE error. '100110: BQ_L2_WI_ITOM_TYPE error.
	27:25	Bus Queue Error Type	'001: Address Parity Error. '010: Response Hard Error. '011: Response Parity Error.
	28	MCE Driven	1 if MCE is driven.
	29	MCE Observed	1 if MCE is observed.
	30	Internal BINIT	1 if BINIT driven for this processor.
	31	BINIT Observed	1 if BINIT is observed for this processor.
	Other Information	33:32	Reserved
34		PIC and FSB Data Parity	Data Parity detected on either PIC or FSB access.
35		Reserved	Reserved
36		Response Parity Error	This bit is asserted in IA32_MCI_STATUS if this component has received a parity error on the RS[2:0]# pins for a response transaction. The RS signals are checked by the RSP# external pin.

**Table 17-4. Incremental Bus Error Codes of Machine Check for Processors
Based on Intel® Core™ Microarchitecture (Contd.)**

Type	Bit No.	Bit Function	Bit Description
	37	FSB Address Parity	Address parity error detected: 1: Address parity error detected. 0: No address parity error.
	38	Timeout BINIT	This bit is asserted in IA32_MCI_STATUS if this component has experienced a ROB time-out, which indicates that no micro-instruction has been retired for a predetermined period of time. A ROB time-out occurs when the 23-bit ROB time-out counter carries a 1 out of its high order bit. The timer is cleared when a micro-instruction retires, an exception is detected by the core processor, RESET is asserted, or when a ROB BINIT occurs. The ROB time-out counter is prescaled by the 8-bit PIC timer which is a divide by 128 of the bus clock the bus clock is 1:2, 1:3, 1:4 of the core clock). When a carry out of the 8-bit PIC timer occurs, the ROB counter counts up by one. While this bit is asserted, it cannot be overwritten by another error.
	41:39	Reserved	Reserved
	42	Hard Error	This bit is asserted in IA32_MCI_STATUS if this component has initiated a bus transactions which has received a hard error response. While this bit is asserted, it cannot be overwritten.
	43	IERR	This bit is asserted in IA32_MCI_STATUS if this component has experienced a failure that causes the IERR pin to be asserted. While this bit is asserted, it cannot be overwritten.
	44	Reserved	Reserved
	45	Reserved	Reserved
	46	Reserved	Reserved
	54:47	Reserved	Reserved
	56:55	Reserved	Reserved.
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.2.1 Model-Specific Machine Check Error Codes for Intel® Xeon® Processor 7400 Series

The Intel® Xeon® processor 7400 series has machine check register banks that generally follow the description of Chapter 16 and Section 17.2. Additional error codes specific to the Intel Xeon processor 7400 series are described in this section.

MC4_STATUS[63:0] is the main error logging for the processor's L3 and front side bus errors for the Intel Xeon processor 7400 series. It supports the L3 Errors, Bus and Interconnect Errors Compound Error Codes in the MCA Error Code Field.

17.2.1.1 Processor Machine Check Status Register, Incremental MCA Error Code Definition

The Intel Xeon processor 7400 series uses compound MCA Error Codes for logging its Bus internal machine check errors, L3 Errors, and Bus/Interconnect Errors. It defines incremental Machine Check error types (IA32_MC6_STATUS[15:0]) beyond those defined in Chapter 16. Table 17-5 lists these incremental MCA error code types that apply to IA32_MC6_STATUS. Error code details are specified in MC6_STATUS [31:16] (see Section 17.2.2), the “Model Specific Error Code” field. The information in the “Other_Info” field (MC4_STATUS[56:32]) is common to the three processor error types. It contains a correctable event count and specifies the MC6_MISC register format.

Table 17-5. Incremental MCA Error Code Types for Intel® Xeon® Processor 7400

Processor MCA_Error_Code (MC6_STATUS[15:0])			
Type	Error Code	Binary Encoding	Meaning
C	Internal Error	0000 0100 0000 0000	Internal Error Type Code.
B	Bus and Interconnect Error	0000 100x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations.
		0000 101x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations.
		0000 110x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations.
		0000 1110 0000 1111	Bus and Interconnection Error Type Code.
		0000 1111 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations.

The **Bold faced** binary encodings are the only encodings used by the processor for MC4_STATUS[15:0].

17.2.2 Intel® Xeon® Processor 7400 Model Specific Error Code Field

17.2.2.1 Processor Model Specific Error Code Field, Type B: Bus and Interconnect Error Codes

The Model Specific Error Code field in MC6_STATUS (bits 31:16) is defined in Table 17-6.

Table 17-6. Type B: Bus and Interconnect Error Codes

Bit Number	Sub-Field Name	Description
16	FSB Request Parity	Parity error detected during FSB request phase.
19:17	Reserved	Reserved
20	FSB Hard Fail Response	“Hard Failure” response received for a local transaction.
21	FSB Response Parity	Parity error on FSB response field detected.
22	FSB Data Parity	FSB data parity error on inbound data detected.
31:23	Reserved	Reserved

17.2.2.2 Processor Model Specific Error Code Field, Type C: Cache Bus Controller Error Codes

Table 17-7. Type C: Cache Bus Controller Error Codes

MC4_STATUS[31:16] (MSCE) Value	Error Description
0000_0000_0000_0001 0001H	Inclusion Error from Core 0.
0000_0000_0000_0010 0002H	Inclusion Error from Core 1.
0000_0000_0000_0011 0003H	Write Exclusive Error from Core 0.
0000_0000_0000_0100 0004H	Write Exclusive Error from Core 1.
0000_0000_0000_0101 0005H	Inclusion Error from FSB.
0000_0000_0000_0110 0006H	SNP Stall Error from FSB.
0000_0000_0000_0111 0007H	Write Stall Error from FSB.
0000_0000_0000_1000 0008H	FSB Arb Timeout Error.
0000_0000_0000_1010 000AH	Inclusion Error from Core 2.
0000_0000_0000_1011 000BH	Write Exclusive Error from Core 2.
0000_0010_0000_0000 0200H	Internal Timeout Error.
0000_0011_0000_0000 0300H	Internal Timeout Error.
0000_0100_0000_0000 0400H	Intel® Cache Safe Technology Queue Full Error or Disabled-ways-in-a-set overflow.
0000_0101_0000_0000 0500H	Quiet cycle Timeout Error (correctable).
1100_0000_0000_0010 C002H	Correctable ECC event on outgoing Core 0 data.
1100_0000_0000_0100 C004H	Correctable ECC event on outgoing Core 1 data.
1100_0000_0000_1000 C008H	Correctable ECC event on outgoing Core 2 data.
1110_0000_0000_0010 E002H	Uncorrectable ECC error on outgoing Core 0 data.
1110_0000_0000_0100 E004H	Uncorrectable ECC error on outgoing Core 1 data.
1110_0000_0000_1000 E008H	Uncorrectable ECC error on outgoing Core 2 data.
— All other encodings —	Reserved

17.3 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR 3400, 3500, 5500 SERIES, MACHINE ERROR CODES FOR MACHINE CHECK

Table 17-8 through Table 17-12 provide information for interpreting additional model-specific fields for memory controller errors relating to the Intel® Xeon® processor 3400, 3500, 5500 series with CPUID DisplayFamily_DisplaySignature 06_1AH, which supports Intel® QuickPath Interconnect links. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC0 and IA32_MC1, incremental error codes for internal machine check are reported in the register bank IA32_MC7, and incremental error codes for the memory controller unit are reported in the register bank IA32_MC8.

17.3.1 Intel® QPI Machine Check Errors

Table 17-8. Intel® QPI Machine Check Error Codes for IA32_MCO_STATUS and IA32_MC1_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model Specific Errors	16	Header Parity	If 1, QPI Header had bad parity.
	17	Data Parity	If 1, QPI Data packet had bad parity.
	18	Retries Exceeded	If 1, the number of QPI retries was exceeded.
	19	Received Poison	If 1, received a data packet that was marked as poisoned by the sender.
	21:20	Reserved	Reserved
	22	Unsupported Message	If 1, QPI received a message encoding it does not support.
	23	Unsupported Credit	If 1, QPI credit type is not supported.
	24	Receive Flit Overrun	If 1, sender sent too many QPI flits to the receiver.
	25	Received Failed Response	If 1, indicates that sender sent a failed response to receiver.
	26	Receiver Clock Jitter	If 1, clock jitter detected in the internal QPI clocking.
56:27	Reserved	Reserved	
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

Table 17-9. Intel® QPI Machine Check Error Codes for IA32_MCO_MISC and IA32_MC1_MISC

Type	Bit No.	Bit Function	Bit Description
Model Specific Errors ¹	7:0	QPI Opcode	Message class and opcode from the packet with the error.
	13:8	RTID	QPI Request Transaction ID.
	15:14	Reserved	Reserved
	18:16	RHNID	QPI Requestor/Home Node ID.
	23:19	Reserved	Reserved
	24	IIB	QPI Interleave/Head Indication Bit.

NOTES:

1. Which of these fields are valid depends on the error type.

17.3.2 Internal Machine Check Errors

Table 17-10. Machine Check Error Codes for IA32_MC7_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	
Model Specific Errors	23:16	Reserved	Reserved
	31:24	Reserved, except for the following	00H: No error. 03H: Reset firmware did not complete. 08H: Received an invalid CMPD. 0AH: Invalid Power Management Request. 0DH: Invalid S-state transition. 11H: VID controller does not match POC controller selected. 1AH: MSID from POC does not match CPU MSID.
	56:32	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.3.3 Memory Controller Errors

Table 17-11. Incremental Memory Controller Error Codes of Machine Check for IA32_MC8_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory error format: 1MMMCCCC
Model Specific Errors	16	Read ECC Error	If 1, ECC occurred on a read.
	17	RAS ECC Error	If 1, ECC occurred on a scrub.
	18	Write Parity Error	If 1, bad parity on a write.
	19	Redundancy Loss	if 1, error in half of redundant memory.
	20	Reserved	Reserved
	21	Memory Range Error	If 1, memory access out of range.
	22	RTID Out of Range	If 1, Internal ID invalid.
	23	Address Parity Error	If 1, bad address parity.
	24	Byte Enable Parity Error	If 1, bad enable parity.
Other Information	37:25	Reserved	Reserved
	52:38	CORE_ERR_CNT	Corrected error count.
	56:53	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

Table 17-12. Incremental Memory Controller Error Codes of Machine Check for IA32_MC8_MISC

Type	Bit No.	Bit Function	Bit Description
Model Specific Errors ¹	7:0	RTID	Transaction Tracker ID.
	15:8	Reserved	Reserved
	17:16	DIMM	DIMM ID which received the error.
	19:18	Channel	Channel ID which received the error.
	31:20	Reserved	Reserved
	63:32	Syndrome	ECC Syndrome.

NOTES:

1. Which of these fields are valid depends on the error type.

17.4 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

Table 17-13 through Table 17-15 provide information for interpreting additional model-specific fields for memory controller errors relating to the Intel® Xeon® processor E5 Family with CPUID DisplayFamily_DisplaySignature 06_2DH, which supports Intel QuickPath Interconnect links. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC6 and IA32_MC7, incremental error codes for internal machine check error from PCU controller are reported in the register bank IA32_MC4, and incremental error codes for the memory controller unit are reported in the register banks IA32_MC8–IA32_MC11.

17.4.1 Internal Machine Check Errors

Table 17-13. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	
Model Specific Errors	19:16	Reserved, except for the following	0000b: No Error 0001b: Non_Mem_Sel 0010b: L_Parity_Error 0011b: Bad_OpCode 0100b: L_Stack_Underflow 0101b: L_Stack_Overflow 0110b: D_Stack_Underflow 0111b: D_Stack_Overflow 1000b: Non-DMem_Sel 1001b: D_Parity_Error

Table 17-13. Machine Check Error Codes for IA32_MC4_STATUS (Contd.)

Type	Bit No.	Bit Function	Bit Description
	23:20	Reserved	Reserved
	31:24	Reserved, except for the following	00H: No Error 0DH: MC_IMC_FORCE_SR_S3_TIMEOUT 0EH: MC_CPD_UNCPD_ST_TIMEOUT 0FH: MC_PKGS_SAFE_WP_TIMEOUT 43H: MC_PECI_MAILBOX QUIESCE_TIMEOUT 5CH: MC_MORE_THAN_ONE_LT_AGENT 60H: MC_INVALID_PKGS_REQ_PCH 61H: MC_INVALID_PKGS_REQ_QPI 62H: MC_INVALID_PKGS_RES_QPI 63H: MC_INVALID_PKGC_RES_PCH 64H: MC_INVALID_PKG_STATE_CONFIG 70H: MC_WATCHDG_TIMEOUT_PKGC_SECONDARY 71H: MC_WATCHDG_TIMEOUT_PKGC_MAIN 72H: MC_WATCHDG_TIMEOUT_PKGS_MAIN 7AH: MC_HA_FAILSTS_CHANGE_DETECTED 81H: MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.4.2 Intel® QPI Machine Check Errors

Table 17-14. Intel® QPI MC Error Codes for IA32_MC6_STATUS and IA32_MC7_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model Specific Errors	56:16	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.4.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the IA32_MC8_STATUS–IA32_MC11_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.” MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in the IA32_M-Ci_STATUS and IA32_MCi_MISC, where i = 8, 11.

Table 17-15. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 8, 11)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model Specific Errors	31:16	Reserved, except for the following	001H: Address parity error. 002H: HA Wrt buffer Data parity error. 004H: HA Wrt byte enable parity error. 008H: Corrected patrol scrub error. 010H: Uncorrected patrol scrub error. 020H: Corrected spare error. 040H: Uncorrected spare error.
	36:32	Other Info	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first device error when corrected error is detected during normal read.
	37	Reserved	Reserved
	56:38		See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

Table 17-16. Intel IMC MC Error Codes for IA32_MCi_MISC (i= 8, 11)

Type	Bit No.	Bit Function	Bit Description
MCA Addr Info ¹	8:0		See Chapter 16, "Machine-Check Architecture."
Model Specific Errors	13:9		<ul style="list-style-type: none"> When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second device error when corrected error is detected during normal read. Otherwise, contains parity error if MCI_Status indicates HA_WB_Data or HA_W_BE parity error.
	29:14	ErrMask_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error bit mask.
	45:30	ErrMask_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error bit mask.
	50:46	FailRank_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing rank.
	55:51	FailRank_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing rank.
	58:56	Reserved	Reserved
	61:59	Reserved	Reserved
	62	Valid_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data from the first correctable error in a memory device.
	63	Valid_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data due to a second correctable error in a memory device. Use this information only after there is valid first error information indicated by bit 62.

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.5 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 V2 AND INTEL® XEON® PROCESSOR E7 V2 FAMILIES, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor E5 v2 family and the Intel® Xeon® processor E7 v2 family are based on the Ivy Bridge-EP microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_3EH. Incremental error codes for internal machine check error from the PCU controller is reported in the register bank IA32_MC4; Table 17-17 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS. Incremental MC error codes related to the Intel QPI links are reported in the register bank IA32_MC5. Information listed in Table 17-14 for QPI MC error codes apply to IA32_MC5_STATUS. Incremental error codes for the memory controller unit are reported in the register banks IA32_MC9–IA32_MC16. Table 17-18 lists model-specific error codes that apply to IA32_MCi_STATUS, where i = 9-16.

17.5.1 Internal Machine Check Errors

Table 17-17. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	
Model Specific Errors	19:16	Reserved, except for the following	0000b: No Error 0001b: Non_Mem_Sel 0010b: L_Parity_Error 0011b: Bad_OpCode 0100b: L_Stack_Underflow 0101b: L_Stack_Overflow 0110b: D_Stack_Underflow 0111b: D_Stack_Overflow 1000b: Non-DMem_Sel 1001b: D_Parity_Error
	23:20	Reserved	Reserved
	31:24	Reserved, except for the following	00H: No Error 0DH: MC_IMC_FORCE_SR_S3_TIMEOUT 0EH: MC_CPD_UNCPD_ST_TIMEOUT 0FH: MC_PKGS_SAFE_WP_TIMEOUT 43H: MC_PECI_MAILBOX QUIESCE_TIMEOUT 44H: MC_CRITICAL_VR_FAILED 45H: MC_ICC_MAX-NOTSUPPORTED 5CH: MC_MORE_THAN_ONE_LT_AGENT 60H: MC_INVALID_PKGS_REQ_PCH 61H: MC_INVALID_PKGS_REQ_QPI 62H: MC_INVALID_PKGS_RES_QPI 63H: MC_INVALID_PKGC_RES_PCH 64H: MC_INVALID_PKG_STATE_CONFIG 70H: MC_WATCHDG_TIMEOUT_PKGC_SECONDARY 71H: MC_WATCHDG_TIMEOUT_PKGC_MAIN 72H: MC_WATCHDG_TIMEOUT_PKGS_MAIN

Table 17-17. Machine Check Error Codes for IA32_MC4_STATUS (Contd.)

Type	Bit No.	Bit Function	Bit Description
			7AH: MC_HA_FAILSTS_CHANGE_DETECTED 7BH: MC_PCIE_R2PCIE-RW_BLOCK_ACK_TIMEOUT 81H: MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.5.2 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the IA32_MC9_STATUS–IA32_MC16_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MC*i*_STATUS and IA32_MC*i*_MISC, where *i* = 9–16.

IA32_MC*i*_STATUS (*i*=9–12) logs errors from the first memory controller. The second memory controller logs errors into IA32_MC*i*_STATUS (*i*=13–16).

Table 17-18. Intel IMC MC Error Codes for IA32_MC*i*_STATUS (*i*= 9–16)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory Controller error format: 000F 0000 1MMM CCCC
Model Specific Errors	31:16	Reserved, except for the following	001H: Address parity error. 002H: HA Wrt buffer data parity error. 004H: HA Wrt byte enable parity error. 008H: Corrected patrol scrub error.
			010H: Uncorrected patrol scrub error. 020H: Corrected spare error. 040H: Uncorrected spare error. 080H: Corrected memory read error. (Only applicable with iMC’s “Additional Error logging” Mode-1 enabled.) 100H - iMC, WDB, parity errors
	36:32	Other Info	When MSR_ERROR_CONTROL.[1] is set, logs an encoded value from the first error device.
	37	Reserved	Reserved
	56:38		See Chapter 16, “Machine-Check Architecture.”
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

Table 17-19. Intel IMC MC Error Codes for IA32_MCi_MISC (i= 9–16)

Type	Bit No.	Bit Function	Bit Description
MCA Addr Info ¹	8:0		See Chapter 16, “Machine-Check Architecture.”
Model Specific Errors	13:9		If the error logged is a MCWrDataPar error or a MCWrBEPAr error, this field is the WDB ID that has the parity error; OR if the second error logged is a correctable read error, MC logs the second error device in this field.
	29:14	ErrMask_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error bit mask.
	45:30	ErrMask_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error bit mask.
	50:46	FailRank_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing rank.
	55:51	FailRank_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing rank.
	61:56		Reserved
	62	Valid_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data from a correctable error from memory read associated with first error device.
	63	Valid_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data due to a second correctable error in a memory device. Use this information only after there is valid first error info indicated by bit 62.

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.5.3 Home Agent Machine Check Errors

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs errors to the IA32_MC8_{STATUS,ADDR,MISC} registers.

17.6 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 V3 FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor E5 v3 family is based on the Haswell-E microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_3FH. Incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 17-20 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC5, IA32_MC20, and IA32_MC21. Table 17-21 contains information for QPI MC error codes. Incremental error codes for the memory controller unit are reported in the register banks IA32_MC9–IA32_MC16. Table 17-22 lists model-specific error codes that apply to IA32_MCi_STATUS, where i = 9–16.

17.6.1 Internal Machine Check Errors

Table 17-20. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	
MCACOD ²	15:0	Internal Errors	0402H: PCU internal errors. 0403H: PCU internal errors. 0406H: Intel TXT errors 0407H: Other UBOX internal errors. An IERR caused by a core 3-strike the IA32_MC3_STATUS (MLC) is copied to the IA32_MC4_STATUS. After a 3-strike, the core MCA banks will be unavailable.
Model Specific Errors	19:16	Reserved, except for the following	0000b: No error. 00xxb: PCU internal error.
	23:20	Reserved	Reserved
	31:24	Reserved, except for the following	00H: No Error 09H: MC_MESSAGE_CHANNEL_TIMEOUT 13H: MC_DMI_TRAINING_TIMEOUT 15H: MC_DMI_CPU_RESET_ACK_TIMEOUT 1EH: MC_VR_ICC_MAX_LT_FUSED_ICC_MAX 25H: MC_SVID_COMMAND_TIMEOUT 29H: MC_VR_VOUT_MAC_LT_FUSED_SVID 2BH: MC_PKGC_WATCHDOG_HANG_CBZ_DOWN 2CH: MC_PKGC_WATCHDOG_HANG_CBZ_UP 44H: MC_CRITICAL_VR_FAILED 46H: MC_VID_RAMP_DOWN_FAILED 49H: MC_SVID_WRITE_REG_VOUT_MAX_FAILED 4BH: MC_BOOT_VID_TIMEOUT; timeout setting boot VID for DRAM 0. 4FH: MC_SVID_COMMAND_ERROR 52H: MC_FIVR_CATAS_OVERVOL_FAULT 53H: MC_FIVR_CATAS_OVERCUR_FAULT 57H: MC_SVID_PKGC_REQUEST_FAILED 58H: MC_SVID_IMON_REQUEST_FAILED 59H: MC_SVID_ALERT_REQUEST_FAILED 62H: MC_INVALID_PKGS_RSP_QPI 64H: MC_INVALID_PKG_STATE_CONFIG 67H: MC_HA_IMC_RW_BLOCK_ACK_TIMEOUT 6AH: MC_MSGCH_PMREQ_CMP_TIMEOUT 72H: MC_WATCHDG_TIMEOUT_PKGS_MASTER 81H: MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.
2. The internal error codes may be model-specific.

17.6.2 Intel® QPI Machine Check Errors

MC error codes associated with the Intel QPI agents are reported in the IA32_MC5_STATUS, IA32_MC20_STATUS, and IA32_MC21_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1PPTRRRRIILL**; see Chapter 16, “Machine-Check Architecture.”

Table 17-21 lists model-specific fields to interpret error codes applicable to IA32_MC5_STATUS, IA32_MC20_STATUS, and IA32_MC21_STATUS.

Table 17-21. Intel® QPI MC Error Codes for IA32_MCi_STATUS (i = 5, 20, 21)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model Specific Errors	31:16	MSCOD	02H: Intel QPI physical layer detected drift buffer alarm.
			03H: Intel QPI physical layer detected latency buffer rollover.
			10H: Intel QPI link layer detected control error from R3QPI.
			11H: Rx entered LLR abort state on CRC error.
			12H: Unsupported or undefined packet.
			13H: Intel QPI link layer control error.
			15H: RBT used un-initialized value.
			20H: Intel QPI physical layer detected a QPI in-band reset but aborted initialization.
			21H: Link failover data self-healing.
			22H: Phy detected in-band reset (no width change).
			23H: Link failover clock failover.
			30H: Rx detected CRC error; successful LLR after Phy re-init.
			31H: Rx detected CRC error; successful LLR without Phy re-init.
			All other values are reserved.
	37:32	Reserved	Reserved
	52:38	Corrected Error Cnt	
	56:53	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.6.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the IA32_MC9_STATUS–IA32_MC16_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC, where i = 9–16.

IA32_MCi_STATUS (i=9–12) logs errors from the first memory controller. The second memory controller logs errors into IA32_MCi_STATUS (i=13–16).

Table 17-22. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 9–16)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model Specific Errors	31:16	Reserved, except for the following	0001H: DDR3 address parity error. 0002H: Uncorrected HA write data error. 0004H: Uncorrected HA data byte enable error. 0008H: Corrected patrol scrub error. 0010H: Uncorrected patrol scrub error. 0020H: Corrected spare error. 0040H: Uncorrected spare error. 0080H: Corrected memory read error. (Only applicable with iMC's "Additional Error logging" Mode-1 enabled.) 0100H: iMC, write data buffer parity errors. 0200H: DDR4 command address parity error.
	36:32	Other Info	When MSR_ERROR_CONTROL.[1] is set, logs an encoded value from the first error device.
	37	Reserved	Reserved
	56:38		See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

Table 17-23. Intel IMC MC Error Codes for IA32_MCi_MISC (i= 9–16)

Type	Bit No.	Bit Function	Bit Description
MCA Addr Info ¹	8:0		See Chapter 16, "Machine-Check Architecture."
Model Specific Errors	13:9		If the error logged is an MCWrDataPar error or an MCWrBEPAr error, this field is the WDB ID that has the parity error; OR if the second error logged is a correctable read error, MC logs the second error device in this field.
	29:14	ErrMask_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error bit mask.
	45:30	ErrMask_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error bit mask.
	50:46	FailRank_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing rank.
	55:51	FailRank_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing rank.
	61:56	Reserved	Reserved
	62	Valid_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data from a correctable error from a memory read associated with first error device.
	63	Valid_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data due to a second correctable error in a memory device. Use this information only after there is valid first error information indicated by bit 62.

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.6.4 Home Agent Machine Check Errors

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs errors in the IA32_MC8_{STATUS,ADDR,MISC} registers.

17.7 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR D FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor D family is based on the Broadwell microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_56H. Incremental error codes for internal machine check error from the PCU controller are reported in the register bank IA32_MC4. Table 17-24 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS. Incremental error codes for the memory controller unit are reported in the register banks IA32_MC9–IA32_MC10. Table 17-18 lists model-specific error codes that apply to IA32_M-Ci_STATUS, where i = 9–10.

17.7.1 Internal Machine Check Errors

Table 17-24. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	
MCACOD ²	15:0	Internal Errors	0402H: PCU internal errors. 0403H: Internal errors. 0406H: Intel TXT errors. 0407H: Other UBOX internal errors. On an IERR caused by a core 3-strike, the IA32_MC3_STATUS (MLC) is copied to the IA32_MC4_STATUS. After a 3-strike, the core MCA banks will be unavailable.
Model Specific Errors	19:16	Reserved, except for the following	0000b: No error. 00x1b: PCU internal error. 001xb: PCU internal error.
	23:20	Reserved, except for the following	x1xxb: UBOX error.
	31:24	Reserved, except for the following	00H: No Error 09H: MC_MESSAGE_CHANNEL_TIMEOUT 13H: MC_DMI_TRAINING_TIMEOUT 15H: MC_DMI_CPU_RESET_ACK_TIMEOUT 1EH: MC_VR_ICC_MAX_LT_FUSED_ICC_MAX 25H: MC_SVID_COMMAND_TIMEOUT 26H: MCA_PKGC_DIRECT_WAKE_RING_TIMEOUT 29H: MC_VR_VOUT_MAC_LT_FUSED_SVID 2BH: MC_PKGC_WATCHDOG_HANG_CBZ_DOWN 2CH: MC_PKGC_WATCHDOG_HANG_CBZ_UP 44H: MC_CRITICAL_VR_FAILED 46H: MC_VID_RAMP_DOWN_FAILED 49H: MC_SVID_WRITE_REG_VOUT_MAX_FAILED

Table 17-24. Machine Check Error Codes for IA32_MC4_STATUS (Contd.)

Type	Bit No.	Bit Function	Bit Description
			4BH: MC_PP1_BOOT_VID_TIMEOUT. Timeout setting boot VID for DRAM 0. 4FH: MC_SVID_COMMAND_ERROR. 52H: MC_FIVR_CATAS_OVERVOL_FAULT. 53H: MC_FIVR_CATAS_OVERCUR_FAULT. 57H: MC_SVID_PKG_REQUEST_FAILED 58H: MC_SVID_IMON_REQUEST_FAILED 59H: MC_SVID_ALERT_REQUEST_FAILED 62H: MC_INVALID_PKGS_RSP_QPI 64H: MC_INVALID_PKG_STATE_CONFIG 67H: MC_HA_IMC_RW_BLOCK_ACK_TIMEOUT 6AH: MC_MSGCH_PMREQ_CMP_TIMEOUT 72H: MC_WATCHDG_TIMEOUT_PKGS_MASTER 81H: MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.
2. The internal error codes may be model-specific.

17.7.2 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the IA32_MC9_STATUS–IA32_MC10_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC, where i = 9–10.

Table 17-25. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 9–10)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model Specific Errors	31:16	Reserved, except for the following	0001H: DDR3 address parity error. 0002H: Uncorrected HA write data error. 0004H: Uncorrected HA data byte enable error. 0008H: Corrected patrol scrub error. 0010H: Uncorrected patrol scrub error. 0100H: iMC, write data buffer parity errors. 0200H: DDR4 command address parity error.
	36:32	Other Info	Reserved
	37	Reserved	Reserved
	56:38		See Chapter 16, “Machine-Check Architecture.”
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.8 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 V4 FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor E5 v4 family is based on the Broadwell microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_4FH. Incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 17-20 in Section 17.6.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC5, IA32_MC20, and IA32_MC21. Information listed in Table 17-21 of Section 17.6.1 covers QPI MC error codes.

17.8.1 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the IA32_MC9_STATUS–IA32_MC16_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

Table 17-26 lists model-specific error codes that apply to IA32_MCi_STATUS, where i = 9–16.

IA32_MCi_STATUS (i=9–12) logs errors from the first memory controller. The second memory controller logs errors into IA32_MCi_STATUS (i=13–16).

Table 17-26. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 9–16)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model Specific Errors	31:16	Reserved, except for the following	0001H: DDR3 address parity error. 0002H: Uncorrected HA write data error. 0004H: Uncorrected HA data byte enable error. 0008H: Corrected patrol scrub error. 0010H: Uncorrected patrol scrub error. 0020H: Corrected spare error. 0040H: Uncorrected spare error. 0100H: iMC, write data buffer parity errors. 0200H: DDR4 command address parity error.
	36:32	Other Info	Reserved
	37	Reserved	Reserved
	56:38		See Chapter 16, “Machine-Check Architecture.”
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.8.2 Home Agent Machine Check Errors

MC error codes associated with mirrored memory corrections are reported in the IA32_MC7_MISC and IA32_MC8_MISC MSRs. Table 17-27 lists model-specific error codes that apply to IA32_MCi_MISC, where i = 7, 8.

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs errors in the IA32_MC8_{STATUS,ADDR,MISC} registers.

Table 17-27. Intel HA MC Error Codes for IA32_MCi_MISC (i= 7, 8)

Bit No.	Bit Function	Bit Description
5:0	LSB	See Figure 16-8.
8:6	Address Mode	See Table 16-3.
40:9	Reserved	Reserved
41	Failover	Error occurred at a pair of mirrored memory channels. Error was corrected by mirroring with channel failover.
42	Mirrorcorr	Error was corrected by mirroring and primary channel scrubbed successfully.
63:43	Reserved	Reserved

17.9 INCREMENTAL DECODING INFORMATION: INTEL® XEON® SCALABLE PROCESSOR FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

In the Intel® Xeon® Scalable Processor Family with CUID DisplayFamily_DisplaySignature 06_55H, incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 17-28 in Section 17.9.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

17.9.1 Internal Machine Check Errors

Table 17-28. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	
MCACOD ²	15:0	Internal Errors	0402H: PCU internal errors. 0403H: PCU internal errors. 0406H: Intel TXT errors. 0407H: Other UBOX internal errors. On an IERR caused by a core 3-strike, the IA32_MC3_STATUS (MLC) is copied to the IA32_MC4_STATUS. After a 3-strike, the core MCA banks will be unavailable.
Model Specific Errors	19:16	Reserved, except for the following	0000b: No error. 00xxb: PCU internal error.

Table 17-28. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
	23:20	Reserved	Reserved
	31:24	Reserved, except for the following	00H: No Error 0DH: MCA_DMI_TRAINING_TIMEOUT 0FH: MCA_DMI_CPU_RESET_ACK_TIMEOUT 10H: MCA_MORE_THAN_ONE_LT_AGENT 1EH: MCA_BIOS_RST_CPL_INVALID_SEQ 1FH: MCA_BIOS_INVALID_PKG_STATE_CONFIG 25H: MCA_MESSAGE_CHANNEL_TIMEOUT 27H: MCA_MSGCH_PMREQ_CMP_TIMEOUT 30H: MCA_PKGC_DIRECT_WAKE_RING_TIMEOUT 31H: MCA_PKGC_INVALID_RSP_PCH 33H: MCA_PKGC_WATCHDOG_HANG_CBZ_DOWN 34H: MCA_PKGC_WATCHDOG_HANG_CBZ_UP 38H: MCA_PKGC_WATCHDOG_HANG_C3_UP_SF 40H: MCA_SVID_VCCIN_VR_ICC_MAX_FAILURE 41H: MCA_SVID_COMMAND_TIMEOUT 42H: MCA_SVID_VCCIN_VR_VOUT_MAX_FAILURE 43H: MCA_SVID_CPU_VR_CAPABILITY_ERROR 44H: MCA_SVID_CRITICAL_VR_FAILED 45H: MCA_SVID_SA_ITD_ERROR 46H: MCA_SVID_READ_REG_FAILED 47H: MCA_SVID_WRITE_REG_FAILED 48H: MCA_SVID_PKGC_INIT_FAILED 49H: MCA_SVID_PKGC_CONFIG_FAILED 4AH: MCA_SVID_PKGC_REQUEST_FAILED 4BH: MCA_SVID_IMON_REQUEST_FAILED 4CH: MCA_SVID_ALERT_REQUEST_FAILED 4DH: MCA_SVID_MCP_VP_ABSENT_OR_RAMP_ERROR 4EH: MCA_SVID_UNEXPECTED_MCP_VP_DETECTED 51H: MCA_FIVR_CATAS_OVERVOL_FAULT 52H: MCA_FIVR_CATAS_OVERCUR_FAULT 58H: MCA_WATCHDG_TIMEOUT_PKGC_SECONDARY 59H: MCA_WATCHDG_TIMEOUT_PKGC_MAIN 5AH: MCA_WATCHDG_TIMEOUT_PKGS_MAIN 61H: MCA_PKGS_CPD_UNPCD_TIMEOUT 63H: MCA_PKGS_INVALID_REQ_PCH 64H: MCA_PKGS_INVALID_REQ_INTERNAL 65H: MCA_PKGS_INVALID_RSP_INTERNAL 6BH: MCA_PKGS_SMBUS_VPP_PAUSE_TIMEOUT 81H: MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	52:32	Reserved	Reserved
	54:53	CORR_ERR_STATUS	Reserved

Table 17-28. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
	56:55	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.
2. The internal error codes may be model-specific.

17.9.2 Interconnect Machine Check Errors

MC error codes associated with the link interconnect agents are reported in the IA32_MC5_STATUS, IA32_MC12_STATUS, and IA32_MC19_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1PPTRRRRIILL**; see Chapter 16, “Machine-Check Architecture.”

Table 17-29 lists model-specific fields to interpret error codes applicable to IA32_MCi_STATUS, i= 5, 12, 19.

Table 17-29. Interconnect MC Error Codes for IA32_MCi_STATUS (i = 5, 12, 19)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Bus error format: 1PPTRRRRIILL The two supported compound error codes: <ul style="list-style-type: none"> ▪ 0x0COF: Unsupported/Undefined Packet. ▪ 0x0EOF: For all other corrected and uncorrected errors.
Model Specific Errors	21:16	MSCOD	The encoding of Uncorrectable (UC) errors are: 00H: UC Phy Initialization Failure. 01H: UC Phy detected drift buffer alarm. 02H: UC Phy detected latency buffer rollover. 10H: UC link layer Rx detected CRC error; unsuccessful LLR entered abort state. 11H: UC LL Rx unsupported or undefined packet. 12H: UC LL or Phy control error. 13H: UC LL Rx parameter exchange exception. 1FH: UC LL detected control error from the link-mesh interface. The encoding of correctable (COR) errors are: 20H: COR Phy initialization abort. 21H: COR Phy reset. 22H: COR Phy lane failure, recovery in x8 width. 23H: COR Phy L0c error corrected without Phy reset. 24H: COR Phy L0c error triggering Phy reset. 25H: COR Phy L0p exit error corrected with Phy reset. 30H: COR LL Rx detected CRC error; successful LLR without Phy re-init. 31H: COR LL Rx detected CRC error; successful LLR with Phy re-init. All other values are reserved.

Table 17-29. Interconnect MC Error Codes for IA32_MCi_STATUS (i = 5, 12, 19)

Type	Bit No.	Bit Function	Bit Description
	31:22	MSCOD_SPARE	The definition below applies to MSCOD 12h (UC LL or Phy Control Errors) [Bit 22] : Phy Control Error. [Bit 23] : Unexpected Retry.Ack flit. [Bit 24] : Unexpected Retry.Req flit. [Bit 25] : RF parity error. [Bit 26] : Routeback Table error. [Bit 27] : Unexpected Tx Protocol flit (EOP, Header or Data). [Bit 28] : Rx Header-or-Credit BGF credit overflow/underflow. [Bit 29] : Link Layer Reset still in progress when Phy enters LO (Phy training should not be enabled until after LL reset is complete as indicated by KTILCL.LinkLayerReset going back to 0). [Bit 30] : Link Layer reset initiated while protocol traffic not idle. [Bit 31] : Link Layer Tx Parity Error.
	37:32	Reserved	Reserved
	52:38	Corrected Error Cnt	
	56:53	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.9.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the IA32_MC13_STATUS–IA32_MC18_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

IA32_MCi_STATUS (i=13,14,17) logs errors from the first memory controller. The second memory controller logs errors into IA32_MCi_STATUS (i=15,16,18).

Table 17-30. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13–18)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model Specific Errors	31:16	Reserved, except for the following	0001H: Address parity error.
			0002H: HA write data parity error.
			0004H: HA write byte enable parity error.
			0008H: Corrected patrol scrub error.
			0010H: Uncorrected patrol scrub error.
			0020H: Corrected spare error.
			0040H: Uncorrected spare error.
			0080H: Any HA read error.
			0100H: WDB read parity error.
			0200H: DDR4 command address parity error.
			0400H: Uncorrected address parity error.
			0800H: Unrecognized request type.
			0801H: Read response to an invalid scoreboard entry.
			0802H: Unexpected read response.
			0803H: DDR4 completion to an invalid scoreboard entry.
			0804H: Completion to an invalid scoreboard entry.
			0805H: Completion FIFO overflow.
			0806H: Correctable parity error.
			0807H: Uncorrectable error.
			0808H: Interrupt received while outstanding interrupt was not ACKed.
			0809H: ERID FIFO overflow.
			080AH: Error on Write credits.
			080BH: Error on Read credits.
			080CH: Scheduler error.
			080DH: Error event.
	36:32	Other Info	MC logs the first error device. This is an encoded 5-bit value of the device.
	37	Reserved	Reserved
	56:38		See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.9.4 M2M Machine Check Errors

MC error codes associated with M2M are reported in the IA32_MC7_STATUS and IA32_MC8_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, "Machine-Check Architecture."

Table 17-31. M2M MC Error Codes for IA32_MCi_STATUS (i= 7, 8)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Compound error format: 0000 0000 1MMM CCCC
Model Specific Errors	16	MscodDataRdErr	Logged an MC read data error.
	17	Reserved	Reserved
	18	MscodPtlWrErr	Logged an MC partial write data error.
	19	MscodFullWrErr	Logged a full write data error.
	20	MscodBgfErr	Logged an M2M clock-domain-crossing buffer (BGF) error.
	21	MscodTimeOut	Logged an M2M time out.
	22	MscodParErr	Logged an M2M tracker parity error.
	23	MscodBucket1Err	Logged a fatal Bucket1 error.
	31:24	Reserved	Reserved
	36:32	Other Info	MC logs the first error device. This is an encoded 5-bit value of the device.
	37	Reserved	Reserved
	56:38		See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.9.5 Home Agent Machine Check Errors

MC error codes associated with mirrored memory corrections are reported in the IA32_MC7_MISC and IA32_MC8_MISC MSRs. Table 17-32 lists model-specific error codes that apply to IA32_MCi_MISC, where i = 7, 8.

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs errors in the IA32_MC8_{STATUS,ADDR,MISC} registers.

Table 17-32. Intel HA MC Error Codes for IA32_MCi_MISC (i= 7, 8)

Bit No.	Bit Function	Bit Description
5:0	LSB	See Figure 16-8.
8:6	Address Mode	See Table 16-3.
40:9	Reserved	Reserved
61:41	Reserved	Reserved
62	Mirrorcorr	Error was corrected by mirroring and primary channel scrubbed successfully.
63	Failover	Error occurred at a pair of mirrored memory channels. Error was corrected by mirroring with channel failover.

17.10 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY WITH CPUID DISPLAYFAMILY_DISPLAYMODEL SIGNATURE 06_5FH, MACHINE ERROR CODES FOR MACHINE CHECK

In Intel Atom[®] processors based on Goldmont Microarchitecture with CPUID DisplayFamily_DisplaySignature 06_5FH (Denverton), incremental error codes for the memory controller unit are reported in the register banks IA32_MC6 and IA32_MC7. Table 17-33 in Section 17.10.1 lists model-specific fields to interpret error codes applicable to IA32_MCi_STATUS, where i = 6, 7.

17.10.1 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the IA32_MC6_STATUS and IA32_MC7_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

Table 17-33. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 6, 7)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	
Model Specific Errors	31:16	Reserved, except for the following	01H: Cmd/Addr parity. 02H: Corrected Demand/Patrol Scrub error. 04H: Uncorrected patrol scrub error. 08H: Uncorrected demand read error. 10H: WDB read ECC.
	36:32	Other Info	
	37	Reserved	Reserved
	56:38		See Chapter 16, “Machine-Check Architecture.”
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.11 INCREMENTAL DECODING INFORMATION: 3RD GENERATION INTEL[®] XEON[®] SCALABLE PROCESSOR FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

In the 3rd generation Intel[®] Xeon[®] Scalable Processor Family with CPUID DisplayFamily_DisplaySignatures of 06_6AH and 06_6CH, incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 17-34 in Section 17.11.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

17.11.1 Internal Machine Check Errors

Table 17-34. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
Machine Check Error Codes ¹	15:0	MCCOD	
MCCOD	15:0	Internal Errors	The value of this field will be 0402H for the PCU and 0406H for internal firmware errors. This applies for any logged error.
Model Specific Errors	19:16	Reserved, except for the following	Model specific error code bits 19:16. This logs the type of HW UC (PCU/VCU) error that has occurred. There are 7 errors defined. 01H: Instruction address out of valid space. 02H: Double bit RAM error on Instruction Fetch. 03H: Invalid OpCode seen. 04H: Stack Underflow. 05H: Stack Overflow. 06H: Data address out of valid space. 07H: Double bit RAM error on Data Fetch.
	23:20	Reserved, except for the following	Model specific error code bits 23:20. This logs the type of HW FSM error that has occurred. There are 3 errors defined. 04H: Clock/power IP response timeout. 05H: SMBus controller raised SMI. 09H: PM controller received invalid transaction.
	31:24	Reserved, except for the following	0DH: MCA_LLC_BIST_ACTIVE_TIMEOUT 0EH: MCA_DMI_TRAINING_TIMEOUT 0FH: MCA_DMI_STRAP_SET_ARRIVAL_TIMEOUT 10H: MCA_DMI_CPU_RESET_ACK_TIMEOUT 11H: MCA_MORE_THAN_ONE_LT_AGENT 14H: MCA_INCOMPATIBLE_PCH_TYPE 1EH: MCA_BIOS_RST_CPL_INVALID_SEQ 1FH: MCA_BIOS_INVALID_PKG_STATE_CONFIG 2DH: MCA_PCU_PMAX_CALIB_ERROR 2EH: MCA_TSC100_SYNC_TIMEOUT 3AH: MCA_GPSB_TIMEOUT 3BH: MCA_PMSB_TIMEOUT 3EH: MCA_IOSFSB_PMREQ_CMP_TIMEOUT 40H: MCA_SVID_VCCIN_VR_ICC_MAX_FAILURE 42H: MCA_SVID_VCCIN_VR_VOUT_FAILURE 43H: MCA_SVID_CPU_VR_CAPABILITY_ERROR 44H: MCA_SVID_CRITICAL_VR_FAILED 45H: MCA_SVID_SA_ITD_ERROR 46H: MCA_SVID_READ_REG_FAILED 47H: MCA_SVID_WRITE_REG_FAILED 4AH: MCA_SVID_PKGC_REQUEST_FAILED

Table 17-34. Machine Check Error Codes for IA32_MC4_STATUS (Contd.)

Type	Bit No.	Bit Function	Bit Description
			4BH: MCA_SVID_IMON_REQUEST_FAILED 4CH: MCA_SVID_ALERT_REQUEST_FAILED 4DH: MCA_SVID_MCP_VR_RAMP_ERROR 56H: MCA_FIVR_PD_HARDERR 58H: MCA_WATCHDOG_TIMEOUT_PKGC_SECONDARY 59H: MCA_WATCHDOG_TIMEOUT_PKGC_MAIN 5AH: MCA_WATCHDOG_TIMEOUT_PKGS_MAIN 5BH: MCA_WATCHDOG_TIMEOUT_MSG_CH_FSM 5CH: MCA_WATCHDOG_TIMEOUT_BULK_CR_FSM 5DH: MCA_WATCHDOG_TIMEOUT_IOSFSB_FSM 60H: MCA_PKGS_SAFE_WP_TIMEOUT 61H: MCA_PKGS_CPD_UNCPD_TIMEOUT 62H: MCA_PKGS_INVALID_REQ_PCH 63H: MCA_PKGS_INVALID_REQ_INTERNAL 64H: MCA_PKGS_INVALID_RSP_INTERNAL 65H-7AH: MCA_PKGS_RESET_PREP_TIMEOUT 7BH: MCA_PKGS_SMBUS_VPP_PAUSE_TIMEOUT 7CH: MCA_PKGS_SMBUS_MCP_PAUSE_TIMEOUT 7DH: MCA_PKGS_SMBUS_SPD_PAUSE_TIMEOUT 80H: MCA_PKGC_DISP_BUSY_TIMEOUT 81H: MCA_PKGC_INVALID_RSP_PCH 83H: MCA_PKGC_WATCHDOG_HANG_CBZ_DOWN 84H: MCA_PKGC_WATCHDOG_HANG_CBZ_UP 87H: MCA_PKGC_WATCHDOG_HANG_C2_BLKMASTER 88H: MCA_PKGC_WATCHDOG_HANG_C2_PSLIMIT 89H: MCA_PKGC_WATCHDOG_HANG_SETDISP 8BH: MCA_PKGC_ALLOW_L1_ERROR 90H: MCA_RECOVERABLE_DIE_THERMAL_TOO_HOT A0H: MCA_ADR_SIGNAL_TIMEOUT A1H: MCA_BCLK_FREQ_OC_ABOVE_THRESHOLD B0H: MCA_DISPATCHER_RUN_BUSY_TIMEOUT
	37:32	ENH_MCA_AVAIL0	Available when Enhanced MCA is in use.
	52:38	CORR_ERR_COUNT	Correctable error count.
	54:53	CORRERRORSTATUSIND	These bits are used to indicate when the number of corrected errors has exceeded the safe threshold to the point where an uncorrected error has become more likely to happen. Table 3 shows the encoding of these bits.
	56:55	ENH_MCA_AVAIL1	Available when Enhanced MCA is in use.
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.11.2 Interconnect Machine Check Errors

MC error codes associated with the link interconnect agents are reported in the IA32_MC5_STATUS, IA32_MC7_STATUS, and IA32_MC8_STATUS MSRs. The supported error codes follow the architectural MCACOD definition type **1PPTRRRRIILL**; see Chapter 16, "Machine-Check Architecture."

NOTE

The interconnect machine check errors in this section apply only to the 3rd generation Intel Xeon Scalable Processor Family with a CPUID DisplayFamily_DisplaySignature of 06_6AH. These do not apply to the 3rd generation Intel Xeon Scalable Processor Family with a CPUID DisplayFamily_DisplaySignature of 06_6CH.

Table 17-35 lists model-specific fields to interpret error codes applicable to IA32_MCi_STATUS, where i= 5, 7, 8.

Table 17-35. Interconnect MC Error Codes for IA32_MCi_STATUS (i = 5, 7, 8)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Bus error format: 1PPTRRRRIILL The two supported compound error codes: <ul style="list-style-type: none"> ▪ 0x0COF: Unsupported/Undefined Packet. ▪ 0x0E0F: For all other corrected and uncorrected errors.
Model Specific Errors	21:16	MSCOD	The encoding of Uncorrectable (UC) errors are: 00H: Phy Initialization Failure (NumInit). 01H: Phy Detected Drift Buffer Alarm. 02H: Phy Detected Latency Buffer Rollover. 10H: LL Rx detected CRC error: unsuccessful LLR (entered Abort state). 11H: LL Rx Unsupported/Undefined packet. 12H: LL or Phy Control Error. 13H: LL Rx Parameter Exception. 1FH: LL Detected Control Error. The encoding of correctable (COR) errors are: 20H: Phy Initialization Abort. 21H: Phy Inband Reset. 22H: Phy Lane failure, recovery in x8 width. 23H: Phy L0c error corrected without Phy reset. 24H: Phy L0c error triggering Phy reset. 25H: Phy L0p exit error corrected with reset. 30H: LL Rx detected CRC error: successful LLR without Phy Re-init. 31H: LL Rx detected CRC error: successful LLR with Phy Re-init. 32H: Tx received LLR. All other values are reserved.

Table 17-35. Interconnect MC Error Codes for IA32_MCi_STATUS (i = 5, 7, 8) (Contd.)

Type	Bit No.	Bit Function	Bit Description
	31:22	MSCOD_SPARE	The definition below applies to MSCOD 12h (UC LL or Phy Control Errors). [Bit 22] : Phy Control Error. [Bit 23] : Unexpected Retry.Ack flit. [Bit 24] : Unexpected Retry.Req flit. [Bit 25] : RF parity error. [Bit 26] : Routeback Table error. [Bit 27] : Unexpected Tx Protocol flit (EOP, Header or Data). [Bit 28] : Rx Header-or-Credit BGF credit overflow/underflow. [Bit 29] : Link Layer Reset still in progress when Phy enters LO (Phy training should not be enabled until after LL reset is complete as indicated by KTILCL.LinkLayerReset going back to 0). [Bit 30] : Link Layer reset initiated while protocol traffic not idle. [Bit 31] : Link Layer Tx Parity Error.
	37:32	OTHER_INFO	Other Info.
	56:38	Corrected Error Cnt	See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.11.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers for the 3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture are defined in Table 17-37.

The MSRs reporting MC error codes differ depending on the CPUID DisplayFamily_DisplaySignature of the processor. See Table 17-36 for details.

Table 17-36. MSRs Reporting MC Error Codes by CPUID DisplayFamily_DisplaySignature

Processor	CPUID DisplayFamily_DisplaySignature	MSRs Reporting MC Error Codes
3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture	06_6AH	IA32_MC13_STATUS–IA32_MC14_STATUS IA32_MC17_STATUS–IA32_MC18_STATUS IA32_MC21_STATUS–IA32_MC22_STATUS IA32_MC25_STATUS–IA32_MC26_STATUS
3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture	06_6CH	IA32_MC13_STATUS–IA32_MC14_STATUS IA32_MC17_STATUS–IA32_MC18_STATUS

The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

Table 17-37. Intel IMC MC Error Codes for IA32_MCI_STATUS (i= 13–14, 17–18, 21–22, 25–26)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model Specific Errors	27:16	Error Codes	0000H: Uncorrectable spare error. 0001H: End to End address parity error. 0002H: Write data parity error. 0003H: End to End uncorrectable/correctable write data ECC error. 0004H: Write byte enable parity error. 0007H: Transaction ID parity error. 0008H: Correctable patrol scrub error. 0010H: Uncorrectable patrol scrub error. 0020H: Correctable spare error. 0080H: Transient or correctable error for demand or underfill reads or read 2LM metadata error. 00A0H: Uncorrectable error for demand or underfill reads. 0100H: WDB read parity error. 0108H: DDR/DDRT link failure. 0111H: PCLS address CSR parity error. 0112H: PCLS illegal ADDDC configuration error. 0200H: DDR4 command / address parity error. 0400H: RPQ scheduler address parity error. 0800H: 2LM unrecognized request type. 0801H: 2LM read response to an invalid scoreboard entry. 0802H: 2LM unexpected read response. 0803H: 2LM DDR4 completion to an invalid scoreboard entry. 0804H: 2LM DDRT completion to an invalid scoreboard entry. 0805H: 2LM completion FIFO overflow. 0806H: DDRT link parity error. 0807H: DDRT RID uncorrectable error. 0809H: DDRT RID FIFO overflow. 080AH: DDRT error on FNV write credits. 080BH: DDRT error on FNV read credits. 080CH: DDRT scheduler error. 080DH: DDRT FNV error. 080EH: DDRT FNV thermal error. 080FH: DDRT unexpected data packet during CMI idle. 0810H: DDRT RPQ request parity error. 0811H: DDRT WPQ request parity error. 0812H: 2LM NmFillWr CAM multiple hit error. 0813H: CMI credit oversubscription error. 0814H: CMI total credit count error. 0815H: CMI reserved credit pool error. 0816H: DDRT link ECC error.

Table 17-37. Intel IMC MC Error Codes for IA32_MCI_STATUS (i= 13–14, 17–18, 21–22, 25–26) (Contd.)

Type	Bit No.	Bit Function	Bit Description
			0817H: WDB FIFO overflow or underflow errors. 0818H: CMI request FIFO overflow error. 0819H: CMI request FIFO underflow error. 081AH: CMI response FIFO overflow error. 081BH: CMI response FIFO underflow error. 081CH: CMI miscellaneous credit errors. 081DH: CMI MC arbiter errors. 081EH: DDRT write completion FIFO overflow error. 081FH: DDRT write completion FIFO underflow error. 0820H: CMI read completion FIFO overflow error. 0821H: CMI read completion FIFO underflow error. 0822H: TME key RF parity error. 0823H: TME miscellaneous CMI errors. 0824H: TME CMI overflow error. 0825H: TME CMI underflow error. 0826H: Intel® SGX TEM secure bit mismatch detected on demand read. 0827H: TME detected underfill read completion data parity error. 0828H: 2LM Scoreboard Overflow Error. 1008H: Correctable patrol scrub error (mirror secondary example).
	28	Mirror secondary error.	Mirror secondary error.
	31:29	Reserved	Reserved
	37:32	Other Info	Other Info.
	56:38		See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

Additional information is reported in the IA32_MC13_MISC–IA32_MC14_MISC, IA32_MC17_MISC–IA32_MC18_MISC, IA32_MC21_MISC–IA32_MC22_MISC, and IA32_MC25_MISC–IA32_MC26_MISC MSRs. Table 17-38 lists the information reported in IA32_MCi_MISC, where i = 13–14, 17–18, 21–22, and 25–26.

Table 17-38. Additional Information Reported in IA32_MCi_MISC (i= 13–14, 17–18, 21–22, 25–26)

Bit No.	Bit Function	Bit Description
5:0	LSB	See Figure 16-8.
8:6	Address Mode	See Table 16-3.
18:9	Column	Component of sub-DIMM address. Bits 18-17: Reserved. Bit 16: Column 9. Bit 15: Column 8. Bit 14: Column 7. Bit 13: Column 6. Bit 12: Column 5. Bit 11: Column 4. Bit 10: Column 3. Bit 9: Reserved.
39:19	Row	Component of sub-DIMM address.
45:40	Bank	Component of sub-DIMM address. Bit 45: Reserved. Bit 44: Bank group 2. Bit 43: Bank address 1. Bit 42: Bank address 0. Bit 41: Bank group 1. Bit 40: Bank group 0.
51:46	Failed Device	Failing device for correctable error (not valid for uncorrectable or transient errors).
55:52	CBit	CBit
58:56	Chip Select	Chip Select
62:59	ECC Mode	0000b: SDDC 2LM. 0001b: SDDC 1LM. 0010b: SDDC + 1 2LM. 0011b: SDDC + 1 1LM. 0100b: ADDDC 2LM. 0101b: ADDDC 1LM. 0110b: ADDDC + 1 2LM. 0111b: ADDDC + 1 1LM. 1000b: Read from DDRT. 1001b: x8 SDDC. 1010b: x8 SDDC + 1. 1011b: Not a valid ECC mode. Other values: Reserved.
63	Transient	0b: 1b: Error was transient.

17.11.4 M2M Machine Check Errors

MC error codes associated with M2M for the 3rd generation Intel Xeon Scalable Processor Family with a CPUID DisplayFamily_DisplaySignature of 06_6AH are reported in the IA32_MC12_STATUS, IA32_MC16_STATUS, IA32_MC20_STATUS, and IA32_MC24_STATUS MSRs.

MC error codes associated with M2M for the 3rd generation Intel Xeon Scalable Processor Family with a CPUID DisplayFamily_DisplaySignature of 06_6CH are reported in the IA32_MC12_STATUS and IA32_MC16_STATUS MSRs.

The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

Table 17-39. M2M MC Error Codes for IA32_MCi_STATUS (i= 12, 16, 20, 24)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Compound error format: 0000 0000 1MMM CCCC
Model Specific Errors	23:16	MSCOD	Logged an MC error.
	25:24	MscodDDRTtype	Logged a DDR/DDRT specific error.
	26	MscodFailoverWhileResetPrep	Logged a failover specific error while preparing to reset.
	31:27	Reserved	Reserved
	37:32	Other Info	Other information.
	56:38		See Chapter 16, “Machine-Check Architecture.”
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

MC error codes associated with mirrored memory corrections are reported in the IA32_MC12_MISC, IA32_MC16_MISC, IA32_MC20_MISC, and IA32_MC24_MISC MSRs. The model-specific error codes listed in Table 17-32 also apply to IA32_MCi_MISC, where i = 12, 16, 20, 24.

17.12 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY WITH CPUID DISPLAYFAMILY_DISPLAYMODEL SIGNATURE 06_86H, MACHINE ERROR CODES FOR MACHINE CHECK

In Intel Atom[®] processors based on Tremont microarchitecture with CPUID DisplayFamily_DisplaySignature 06_86H, incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 17-34 in Section 17.11.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

17.12.1 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC13_STATUS–IA32_MC15_STATUS. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

The IA32_MCi_STATUS MSR (where i = 13, 14, 15) contains information related to a machine check error if its VAL(valid) flag is set. Bit definitions are the same as those found in Table 17-37 “Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13–14, 17–18, 21–22, 25–26).”

The IA32_MCi_MISC MSR (where i = 13, 14, 15) contains information related memory corrections. Bit definitions are the same as those found in Table 17-38 “Additional Information Reported in IA32_MCi_MISC (i= 13–14, 17–18, 21–22, 25–26).”

17.12.2 M2M Machine Check Errors

MC error codes associated with M2M are reported in the IA32_MC12_STATUS MSR. The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

Bit definitions are the same as those found in Table 17-39 “M2M MC Error Codes for IA32_MCi_STATUS (i= 12, 16, 20, 24).”

17.13 INCREMENTAL DECODING INFORMATION: 4TH GENERATION INTEL® XEON® SCALABLE PROCESSOR FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

In the 4th generation Intel® Xeon® Scalable Processor Family with CPUID DisplayFamily_DisplaySignature of 06_8FH, incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 17-40 in Section 17.13.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

17.13.1 Internal Machine Check Errors

Table 17-40. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCACOD ¹	15:0	Internal Errors	The value of this field will be 0402H for the PCU and 0406H for internal firmware errors. This applies for any logged error.
Model Specific Errors	19:16	Reserved, except for the following	Model specific error code bits 19:16. If MACOD = 40CH, MSCOD encoding should be interpreted as: 01H: MCE when CR4.MCE is clear. 02H: MCE when MCIP bit is set. 03H: MCE under WPS. 04H: Unrecoverable error during security flow execution. 05H: Software triple fault shutdown. 06H: VMX-exit-consistency-check failures. 07H: RSM-consistency-check failures. 08H: Invalid conditions on protected mode SMM entry. 09H: Unrecoverable error during security flow execution. For all other MACOD values, MSCOD logs the type of hardware UC (PCU/VCU) error that has occurred. There are seven errors defined: 01H: Instruction address out of valid space. 02H: Double bit RAM error on Instruction Fetch. 03H: Invalid OpCode seen. 04H: Stack Underflow. 05H: Stack Overflow. 06H: Data address out of valid space. 07H: Double bit RAM error on Data Fetch.

Table 17-40. Machine Check Error Codes for IA32_MC4_STATUS (Contd.)

Type	Bit No.	Bit Function	Bit Description
	23:20	Reserved, except for the following	Model specific error code bits 23:20. This logs the type of HW FSM error that has occurred. There are 3 errors defined: 04H: Clock/power IP response timeout. 05H: SMBus controller raised SMI. 09H: PM controller received invalid transaction.
	31:24	Reserved, except for the following	0DH: MCA_LLC_BIST_ACTIVE_TIMEOUT 0EH: MCA_DMI_TRAINING_TIMEOUT 0FH: MCA_DMI_STRAP_SET_ARRIVAL_TIMEOUT 10H: MCA_DMI_CPU_RESET_ACK_TIMEOUT 11H: MCA_MORE_THAN_ONE_LT_AGENT 14H: MCA_INCOMPATIBLE_PCH_TYPE 1EH: MCA_BIOS_RST_CPL_INVALID_SEQ 1FH: MCA_BIOS_INVALID_PKG_STATE_CONFIG 2DH: MCA_PCU_PMAX_CALIB_ERROR 2EH: MCA_TSC100_SYNC_TIMEOUT 3AH: MCA_GPSB_TIMEOUT 3BH: MCA_PMSB_TIMEOUT 3EH: MCA_IOSFSB_PMREQ_CMP_TIMEOUT 40H: MCA_SVID_VCCIN_VR_ICC_MAX_FAILURE 42H: MCA_SVID_VCCIN_VR_VOUT_FAILURE 43H: MCA_SVID_CPU_VR_CAPABILITY_ERROR 44H: MCA_SVID_CRITICAL_VR_FAILED 45H: MCA_SVID_SA_ITD_ERROR 46H: MCA_SVID_READ_REG_FAILED 47H: MCA_SVID_WRITE_REG_FAILED 4AH: MCA_SVID_PKGC_REQUEST_FAILED 4BH: MCA_SVID_IMON_REQUEST_FAILED 4CH: MCA_SVID_ALERT_REQUEST_FAILED 4DH: MCA_SVID_MCP_VR_RAMP_ERROR 56H: MCA_FIVR_PD_HARDERR 58H: MCA_WATCHDOG_TIMEOUT_PKGC_SECONDARY 59H: MCA_WATCHDOG_TIMEOUT_PKGC_MAIN 5AH: MCA_WATCHDOG_TIMEOUT_PKGS_MAIN 5BH: MCA_WATCHDOG_TIMEOUT_MSG_CH_FSM 5CH: MCA_WATCHDOG_TIMEOUT_BULK_CR_FSM 5DH: MCA_WATCHDOG_TIMEOUT_IOSFSB_FSM 60H: MCA_PKGS_SAFE_WP_TIMEOUT 61H: MCA_PKGS_CPD_UNCPD_TIMEOUT 62H: MCA_PKGS_INVALID_REQ_PCH 63H: MCA_PKGS_INVALID_REQ_INTERNAL 64H: MCA_PKGS_INVALID_RSP_INTERNAL 65H-7AH: MCA_PKGS_RESET_PREP_TIMEOUT 7BH: MCA_PKGS_SMBUS_VPP_PAUSE_TIMEOUT

Table 17-40. Machine Check Error Codes for IA32_MC4_STATUS (Contd.)

Type	Bit No.	Bit Function	Bit Description
			7CH: MCA_PKGS_SMBUS_MCP_PAUSE_TIMEOUT 7DH: MCA_PKGS_SMBUS_SPD_PAUSE_TIMEOUT 80H: MCA_PKGC_DISP_BUSY_TIMEOUT 81H: MCA_PKGC_INVALID_RSP_PCH 83H: MCA_PKGC_WATCHDOG_HANG_CBZ_DOWN 84H: MCA_PKGC_WATCHDOG_HANG_CBZ_UP 87H: MCA_PKGC_WATCHDOG_HANG_C2_BLKMASTER 88H: MCA_PKGC_WATCHDOG_HANG_C2_PSLIMIT 89H: MCA_PKGC_WATCHDOG_HANG_SETDISP 8BH: MCA_PKGC_ALLOW_L1_ERROR 90H: MCA_RECOVERABLE_DIE_THERMAL_TOO_HOT A0H: MCA_ADR_SIGNAL_TIMEOUT A1H: MCA_BCLK_FREQ_OC_ABOVE_THRESHOLD B0H: MCA_DISPATCHER_RUN_BUSY_TIMEOUT C0H: MCA_DISPATCHER_RUN_BUSY_TIMEOUT
	37:32	ENH_MCA_AVAIL0	Available when Enhanced MCA is in use.
	52:38	CORR_ERR_COUNT	Correctable error count.
	54:53	CORRERRORSTATUSIND	These bits are used to indicate when the number of corrected errors has exceeded the safe threshold to the point where an uncorrected error has become more likely to happen. Table 3 shows the encoding of these bits.
	56:55	ENH_MCA_AVAIL1	Available when Enhanced MCA is in use.
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

17.13.2 Interconnect Machine Check Errors

MC error codes associated with the link interconnect agents are reported in the IA32_MC5_STATUS MSR. The supported error codes follow the architectural MCACOD definition type **1PPTRRRIILL**; see Chapter 16, “Machine-Check Architecture.”

Table 17-41 lists model-specific fields to interpret error codes applicable to IA32_MC5_STATUS.

Table 17-41. Interconnect MC Error Codes for IA32_MC5_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Bus error format: 1PPTRRRIILL The two supported compound error codes: <ul style="list-style-type: none"> ▪ 0x0COF: Unsupported/Undefined Packet. ▪ 0x0E0F: For all other corrected and uncorrected errors.
Model Specific Errors	21:16	MSCOD	The encoding of Uncorrectable (UC) errors are: 00H: UC Phy Initialization Failure. 01H: UC Phy Detected Drift Buffer Alarm. 02H: UC Phy Detected Latency Buffer Rollover. 10H: UC LL Rx detected CRC error: unsuccessful LLR (entered Abort state). 11H: UC LL Rx Unsupported/Undefined packet. 12H: UC LL or Phy Control Error. 13H: UC LL Rx Parameter Exception. 15H: UC LL Rx SGX MAC Error. 1FH: UC LL Detected Control Error. The encoding of correctable (COR) errors are: 20H: COR Phy Initialization Abort. 21H: COR Phy Inband Reset. 22H: COR Phy Lane failure, recovery in x8 width. 23H: COR Phy L0c error corrected without Phy reset. 24H: COR Phy L0c error triggering Phy reset. 25H: COR Phy L0p exit error corrected with reset. 30H: COR LL Rx detected CRC error: successful LLR without Phy Re-init. 31H: COR LL Rx detected CRC error: successful LLR with Phy Re-init. All other values are reserved.
	31:22	MSCOD_SPARE	The definition below applies to MSCOD 12H (UC LL or Phy Control Errors). [Bit 22]: Phy Control Error. [Bit 23]: Unexpected Retry.Ack flit. [Bit 24]: Unexpected Retry.Req flit. [Bit 25]: RF parity error. [Bit 26]: Routeback Table error. [Bit 27]: Unexpected Tx Protocol flit (EOP, Header, or Data). [Bit 28]: Rx Header-or-Credit BGF credit overflow/underflow. [Bit 29]: Link Layer Reset still in progress when Phy enters L0 (Phy training should not be enabled until after LL reset is complete as indicated by KTILCL.LinkLayerReset going back to 0). [Bit 30]: Link Layer reset initiated while protocol traffic not idle. [Bit 31]: Link Layer Tx Parity Error.
	37:32	OTHER_INFO	Other Info.
	56:38	Corrected Error Cnt	See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.13.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers for the 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture are reported in the IA32_MC13_STATUS–IA32_MC20_STATUS MSRs.

The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

Table 17-42. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13–20)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model Specific Errors	31:16	Reserved, except for the following	0001H: Address parity error. 0002H: Data parity error. 0003H: Data ECC error. 0004H: Data byte enable parity error. 0007H: Transaction ID parity error. 0008H: Corrected patrol scrub error. 0010H: Uncorrected patrol scrub error. 0020H: Corrected spare error. 0040H: Uncorrected spare error. 0080H: Corrected read error. 00A0H: Uncorrected read error. 00C0H: Uncorrected metadata. 0100H: WDB read parity error. 0108H: DDR link failure. 0200H: DDR5 command / address parity error. 0400H: RPQO parity (primary) error. 0800H: DDR-T bad request. 0801H: DDR Data response to an invalid entry. 0802H: DDR data response to an entry not expecting data. 0803H: DDR5 completion to an invalid entry. 0804H: DDR-T completion to an invalid entry. 0805H: DDR data/completion FIFO overflow. 0806H: DDR-T ERID correctable parity error. 0807H: DDR-T ERID uncorrectable error. 0808H: DDR-T interrupt received while outstanding interrupt was not ACKed. 0809H: ERID FIFO overflow. 080AH: DDR-T error on FNV write credits. 080BH: DDR-T error on FNV read credits. 080CH: DDR-T scheduler error. 080DH: DDR-T FNV error event. 080EH: DDR-T FNV thermal event. 080FH: CMI packet while idle. 0810H: DDR_T_RPQ_REQ_PARITY_ERR. 0811H: DDR_T_WPQ_REQ_PARITY_ERR. 0812H: 2LM_NMFILLWR_CAM_ERR.

Table 17-42. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13–20)

Type	Bit No.	Bit Function	Bit Description
			0813H: CMI_CREDIT_OVERSUB_ERR. 0814H: CMI_CREDIT_TOTAL_ERR. 0815H: CMI_CREDIT_RSVD_POOL_ERR. 0816H: DDR_T_RD_ERROR. 0817H: WDB_FIFO_ERR. 0818H: CMI_REQ_FIFO_OVERFLOW. 0819H: CMI_REQ_FIFO_UNDERFLOW. 081AH: CMI_RSP_FIFO_OVERFLOW. 081BH: CMI_RSP_FIFO_UNDERFLOW. 081CH: CMI_MISC_MC_CRDT_ERRORS. 081DH: CMI_MISC_MC_ARB_ERRORS. 081EH: DDR_T_WR_CMPL_FIFO_OVERFLOW. 081FH: DDR_T_WR_CMPL_FIFO_UNDERFLOW. 0820H: CMI_RD_CPL_FIFO_OVERFLOW. 0821H: CMI_RD_CPL_FIFO_UNDERFLOW. 0822H: TME_KEY_PAR_ERR. 0823H: TME_CMI_MISC_ERR. 0824H: TME_CMI_OVFL_ERR. 0825H: TME_CMI_UFL_ERR. 0826H: TME_TEM_SECURE_ERR. 0827H: TME_UFILL_PAR_ERR. 0829H: INTERNAL_ERR. 082AH: TME_INTEGRITY_ERR. 082BH: TME_TDX_ERR 082CH: TME_UFILL_TEM_SECURE_ERR. 082DH: TME_KEY_POISON_ERR. 082EH: TME_SECURITY_ENGINE_ERR. 1008H: CORR_PATSCRUB_MIRR2ND_ERR. 1010H: UC_PATSCRUB_MIRR2ND_ERR. 1020H: COR_SPARE_MIRR2ND_ERR. 1040H: UC_SPARE_MIRR2ND_ERR. 1080H: HA_RD_MIRR2ND_ERR. 10A0H: HA_UNCORR_RD_MIRR2ND_ERR.
	37:32	Other Info	Other Info.
	56:38		See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

Additional information is reported in the IA32_MC13_MISC–IA32_MC20_MISC MSRs. Table 17-43 lists the information reported in IA32_MCi_MISC, where i = 13–20.

Table 17-43. Additional Information Reported in IA32_MCi_MISC (i= 13–20)

Bit No.	Bit Function	Bit Description
5:0	LSB	See Figure 16-8.
8:6	Address Mode	See Table 16-3.
18:9	Column	Column address for the last retry. To get the real column address from this field, shift the value left by 2.
36:19	Row	Component of sub-DIMM address.
42:37	Bank ID	Component of sub-DIMM address. Bit 42: Reserved. Bit 41: Bank group 2. Bit 40: Bank address 1. Bit 39: Bank address 0. Bit 38: Bank group 1. Bit 37: Bank group 0.
48:43	Failed Device	Failing device for correctable error (not valid for uncorrectable or transient errors).
50:49	Reserved	Reserved
55:51	Failed Device Number	In HBM mode, holds the failed device number for upper 32 bytes.
55:52	CBit	In DDR mode, bits 54-52: sub_rank[2:0]; bit 55: reserved.
58:56	Chip Select	Chip Select
62:59	ECC Mode	0000b: SDDC 2LM. 0001b: SDDC 1LM. 0010b: SDDC + 1 2LM. 0011b: SDDC + 1 1LM. 0100b: ADDDC 2LM. 0101b: ADDDC 1LM. 0110b: ADDDC + 1 2LM. 0111b: ADDDC + 1 1LM. 1000b: Read from DDRT. 1011b: Not a valid ECC mode. For HBM mode: 0001b: 64B read. 1001b: 32B read. Other values: Reserved.
63	Transient	Indicates if the error was a transient error. A transient error is only indicated for demand reads, underfill reads, and patrol. If there was a WDBParity Error, this field indicates the WDB ID bit 6.

17.13.4 M2M Machine Check Errors

MC error codes associated with M2M for the 4th generation Intel Xeon Scalable Processor Family with a CPUID DisplayFamily_DisplaySignature of 06_8FH are reported in the IA32_MC12_STATUS MSR.

The supported error codes follow the architectural MCACOD definition type **1MMMCCCC**; see Chapter 16, “Machine-Check Architecture.”

Table 17-44. M2M MC Error Codes for IA32_MC12_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes ¹	15:0	MCACOD	Compound error format: 0000 0000 1MMM CCCC
Model Specific Errors	23:16	MscodDataRdErr	00H: No error (default). 01H: Read ECC error (MemSpecRd; MemRd; MemRdData; MemRdXto*; MemInv; MemInvXto*; MemInvItoX). 02H: Bucket1 error. 03H: RdTrkr Parity error. 05H: Prefetch channel mismatch. 07H: Read completion parity error. 08H: Response parity error. 09H: Timeout error. 0AH: CMI reserved credit pool error. 0BH: CMI total credit count error. 0CH: CMI credit oversubscription error.
	25:24	MscodDDRTType	00: Not logged, whether error on DDR4 or DDRT. 01: HBM errors.
	31:26	Reserved	Reserved
	37:32	Other Info	Other Info.
	56:38		See Chapter 16, "Machine-Check Architecture."
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

17.13.5 High Bandwidth Memory Machine Check Errors

MC error codes associated with high bandwidth memory for the 4th generation Intel Xeon Scalable Processor Family are reported in the IA32_MC29_STATUS–IA32_MC31_STATUS MSRs.

17.14 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY 0FH, MACHINE ERROR CODES FOR MACHINE CHECK

Table 17-45 provides information for interpreting additional family 0FH model-specific fields for external bus errors. These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally as compound errors with a general form of **0000 1PPT RRRR IILL** in the MCA error code field. See Chapter 16 for information on the interpretation of compound error codes.

Table 17-45. Incremental Decoding Information: Processor Family 0FH, Machine Error Codes for Machine Check

Type	BitNo.	Bit Function	Bit Description
MCA Error Codes ¹	15:0		
Model-Specific Error Codes	16	FSB Address Parity	Address parity error detected: 1: Address parity error detected. 0: No address parity error.
	17	Response Hard Fail	Hardware failure detected on response.
	18	Response Parity	Parity error detected on response.
	19	PIC and FSB Data Parity	Data Parity detected on either PIC or FSB access.
	20	Processor Signature = 00000F04H: Invalid PIC Request All other processors: Reserved	Processor Signature = 00000F04H: Indicates error due to an invalid PIC request access was made to PIC space with WB memory): 1: Invalid PIC request error. 0: No Invalid PIC request error. Reserved
	21	Pad State Machine	The state machine that tracks P and N data-strobe relative timing has become unsynchronized or a glitch has been detected.
	22	Pad Strobe Glitch	Data strobe glitch.
	23	Pad Address Glitch	Address strobe glitch.
Other Information	56:24	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, “Machine-Check Architecture,” for more information.

Table 17-10 provides information on interpreting additional family 0FH model specific fields for cache hierarchy errors. These errors are reported in one of the IA32_MCi_STATUS MSRs. These errors are reported, architecturally, as compound errors with a general form of **0000 0001 RRRR TTLL** in the MCA error code field. See Chapter 16 for how to interpret the compound error code.

17.14.1 Model-Specific Machine Check Error Codes for the Intel® Xeon® Processor MP 7100 Series

The Intel Xeon processor MP 7100 series has five register banks which contain information related to Machine Check Errors. MCI_STATUS[63:0] refers to all five register banks. MC0_STATUS[63:0] through MC3_STATUS[63:0] is the same as previous generations of Intel Xeon processors within Family 0FH. MC4_STATUS[63:0] is the main error logging for the processor’s L3 and front side bus errors. It supports the L3 Errors, Bus and Interconnect Errors Compound Error Codes in the MCA Error Code Field.

Table 17-46. MCI_STATUS Register Bit Definition

Bit Field Name	Bits	Description
MCA_Error_Code	15:0	This field specifies the machine check architecture defined error code for the machine check error condition detected. The machine check architecture defined error codes are guaranteed to be the same for all Intel Architecture processors that implement the machine check architecture. See tables below.
Model_Specific_Error_Code	31:16	This field specifies the model specific error code that uniquely identifies the machine check error condition detected. The model specific error codes may differ among Intel Architecture processors for the same Machine Check Error condition. See tables below.
Other_Info	56:32	The functions of the bits in this field are implementation specific and are not part of the machine check architecture. Software that is intended to be portable among Intel Architecture processors should not rely on the values in this field.
PCC	57	The Processor Context Corrupt flag indicates that the state of the processor might have been corrupted by the error condition detected and that reliable restarting of the processor may not be possible. When clear, this flag indicates that the error did not affect the processor's state. This bit will always be set for MC errors, which are not corrected.
ADDRV	58	The MC_ADDR register valid flag indicates that the MC_ADDR register contains the address where the error occurred. When clear, this flag indicates that the MC_ADDR register does not contain the address where the error occurred. The MC_ADDR register should not be read if the ADDRv bit is clear.
MISCV	59	The MC_MISC register valid flag indicates that the MC_MISC register contains additional information regarding the error. When clear, this flag indicates that the MC_MISC register does not contain additional information regarding the error. MC_MISC should not be read if the MISCV bit is not set.
EN	60	The error enabled flag indicates that reporting of the machine check exception for this error was enabled by the associated flag bit of the MC_CTL register. Note that correctable errors do not have associated enable bits in the MC_CTL register so the EN bit should be clear when a correctable error is logged.
UC	61	The error uncorrected flag indicates that the processor did not correct the error condition. When clear, this flag indicates that the processor was able to correct the event condition.
OVER	62	The machine check overflow flag indicates that a machine check error occurred while the results of a previous error were still in the register bank (i.e., the VAL bit was already set in the MC_STATUS register). The processor sets the OVER flag and software is responsible for clearing it. Enabled errors are written over disabled errors, and uncorrected errors are written over corrected events. Uncorrected errors are not written over previous valid uncorrected errors.
VAL	63	The MC_STATUS register valid flag indicates that the information within the MC_STATUS register is valid. When this flag is set, the processor follows the rules given for the OVER flag in the MC_STATUS register when overwriting previously valid entries. The processor sets the VAL flag and software is responsible for clearing it.

17.14.1.1 Processor Machine Check Status Register MCA Error Code Definition

The Intel Xeon processor MP 7100 series uses compound MCA Error Codes for logging its CBC internal machine check errors, L3 Errors, and Bus/Interconnect Errors. It defines additional Machine Check error types (IA32_MC4_STATUS[15:0]) beyond those defined in Chapter 16. Table 17-47 lists these model-specific MCA error codes. Error code details are specified in MC4_STATUS [31:16]; see Section 17.14.3, the "Model Specific Error Code" field. The information in the "Other_Info" field (MC4_STATUS[56:32]) is common to the three processor error types and contains a correctable event count and specifies the MC4_MISC register format.

Table 17-47. Incremental MCA Error Code for Intel® Xeon® Processor MP 7100

Processor MCA_Error_Code (MC4_STATUS[15:0])			
Type	Error Code	Binary Encoding	Meaning
C	Internal Error	0000 0100 0000 0000	Internal Error Type Code.
A	L3 Tag Error	0000 0001 0000 1011	L3 Tag Error Type Code.
B	Bus and Interconnect Error	0000 100x 0000 1111	Not used, but this encoding is reserved for compatibility with other MCA implementations.
		0000 101x 0000 1111	Not used, but this encoding is reserved for compatibility with other MCA implementations.
		0000 110x 0000 1111	Not used, but this encoding is reserved for compatibility with other MCA implementations.
		0000 1110 0000 1111	Bus and Interconnection Error Type Code.
		0000 1111 0000 1111	Not used, but this encoding is reserved for compatibility with other MCA implementations.

The **bold faced** binary encodings are the only encodings used by the processor for MC4_STATUS[15:0].

17.14.2 Other_Info Field (All MCA Error Types)

The MC4_STATUS[56:32] field is common to the processor's three MCA error types (A, B, and C).

Table 17-48. Other Information Field Bit Definition

Bit Field Name	Bits	Description
39:32	8-bit Correctable Event Count	This field holds a count of the number of correctable events since cold reset. This is a saturating counter; the counter begins at 1 (with the first error) and saturates at a count of 255.
41:40	MC4_MISC Format Type	The value in this field specifies the format of information in the MC4_MISC register. Currently, only two values are defined. Valid only when MISCV is asserted.
43:42	Reserved	Reserved
51:44	ECC Syndrome	ECC syndrome value for a correctable ECC event when the "Valid ECC syndrome" bit is asserted.
52	Valid ECC Syndrome	Set when a correctable ECC event supplies the ECC syndrome.
54:53	Threshold-Based Error Status	00: No tracking. No hardware status tracking is provided for the structure reporting this event. 01: Green. Status tracking is provided for the structure posting the event; the current status is green (below threshold). 10: Yellow. Status tracking is provided for the structure posting the event; the current status is yellow (above threshold). 11: Reserved for future use. Valid only if Valid bit (bit 63) is set. Undefined if the UC bit (bit 61) is set.
56:55	Reserved	Reserved

17.14.3 Processor Model Specific Error Code Field

17.14.3.1 MCA Error Type A: L3 Error

Note: The Model Specific Error Code field in MC4_STATUS (bits 31:16).

Table 17-49. Type A: L3 Error Codes

Bit Num	Sub-Field Name	Description	Legal Value(s)
18:16	L3 Error Code	Describes the L3 error encountered	000: No error. 001: More than one way reporting a correctable event. 010: More than one way reporting an uncorrectable error. 011: More than one way reporting a tag hit. 100: No error. 101: One way reporting a correctable event. 110: One way reporting an uncorrectable error. 111: One or more ways reporting a correctable event while one or more ways are reporting an uncorrectable error.
20:19	---	Reserved	00
31:21	---	Fixed pattern	0010_0000_000

17.14.3.2 Processor Model Specific Error Code Field Type B: Bus and Interconnect Error

Note: The Model Specific Error Code field in MC4_STATUS (bits 31:16).

Table 17-50. Type B: Bus and Interconnect Error Codes

Bit Num	Sub-Field Name	Description
16	FSB Request Parity	Parity error detected during FSB request phase.
17	Core0 Addr Parity	Parity error detected on Core 0 request's address field.
18	Core1 Addr Parity	Parity error detected on Core 1 request's address field.
19	Reserved	Reserved
20	FSB Response Parity	Parity error on FSB response field detected.
21	FSB Data Parity	FSB data parity error on inbound data detected.
22	Core0 Data Parity	Data parity error on data received from Core 0 detected.
23	Core1 Data Parity	Data parity error on data received from Core 1 detected.
24	IDS Parity	Detected an Enhanced Defer parity error (phase A or phase B).
25	FSB Inbound Data ECC	Data ECC event to error on inbound data (correctable or uncorrectable).
26	FSB Data Glitch	Pad logic detected a data strobe 'glitch' (or sequencing error).
27	FSB Address Glitch	Pad logic detected a request strobe 'glitch' (or sequencing error).
31:28	Reserved	Reserved

Exactly one of the bits defined in the preceding table will be set for a Bus and Interconnect Error. The Data ECC can be correctable or uncorrectable; the MC4_STATUS.UC bit distinguishes between correctable and uncorrectable cases with the Other_Info field possibly providing the ECC Syndrome for correctable errors. All other errors for this processor MCA Error Type are uncorrectable.

17.14.3.3 Processor Model Specific Error Code Field Type C: Cache Bus Controller Error

Table 17-51. Type C: Cache Bus Controller Error Codes

MC4_STATUS[31:16] (MSCE) Value	Error Description
0000_0000_0000_0001 0001H	Inclusion Error from Core 0.
0000_0000_0000_0010 0002H	Inclusion Error from Core 1.
0000_0000_0000_0011 0003H	Write Exclusive Error from Core 0.
0000_0000_0000_0100 0004H	Write Exclusive Error from Core 1.
0000_0000_0000_0101 0005H	Inclusion Error from FSB.
0000_0000_0000_0110 0006H	SNP Stall Error from FSB.
0000_0000_0000_0111 0007H	Write Stall Error from FSB.
0000_0000_0000_1000 0008H	FSB Arb Timeout Error.
0000_0000_0000_1001 0009H	CBC OOD Queue Underflow/overflow.
0000_0001_0000_0000 0100H	Enhanced Intel SpeedStep Technology TM1-TM2 Error.
0000_0010_0000_0000 0200H	Internal Timeout Error.
0000_0011_0000_0000 0300H	Internal Timeout Error.
0000_0100_0000_0000 0400H	Intel® Cache Safe Technology Queue Full Error or Disabled-ways-in-a-set overflow.
1100_0000_0000_0001 C001H	Correctable ECC event on outgoing FSB data.
1100_0000_0000_0010 C002H	Correctable ECC event on outgoing Core 0 data.
1100_0000_0000_0100 C004H	Correctable ECC event on outgoing Core 1 data.
1110_0000_0000_0001 E001H	Uncorrectable ECC error on outgoing FSB data.
1110_0000_0000_0010 E002H	Uncorrectable ECC error on outgoing Core 0 data.
1110_0000_0000_0100 E004H	Uncorrectable ECC error on outgoing Core 1 data.
— All other encodings —	Reserved

All errors, except for the correctable ECC types, in this table are uncorrectable. The correctable ECC events may supply the ECC syndrome in the Other_Info field of the MC4_STATUS MSR.

Table 17-52. Decoding Family 0FH Machine Check Codes for Cache Hierarchy Errors

Type	Bit No.	Bit Function	Bit Description
MCA error codes ¹	15:0		
Model Specific Error Codes	17:16	Tag Error Code	Contains the tag error code for this machine check error: 00: No error detected. 01: Parity error on tag miss with a clean line. 10: Parity error/multiple tag match on tag hit. 11: Parity error/multiple tag match on tag miss.
	19:18	Data Error Code	Contains the data error code for this machine check error: 00: No error detected. 01: Single bit error. 10: Double bit error on a clean line. 11: Double bit error on a modified line.
	20	L3 Error	This bit is set if the machine check error originated in the L3 (it can be ignored for invalid PIC request errors): 1: L3 error. 0: L2 error.
	21	Invalid PIC Request	Indicates error due to invalid PIC request access was made to PIC space with WB memory: 1: Invalid PIC request error. 0: No invalid PIC request error.
	31:22	Reserved	Reserved
Other Information	39:32	8-bit Error Count	Holds a count of the number of errors since reset. The counter begins at 0 for the first error and saturates at a count of 255.
	56:40	Reserved	Reserved
Status Register Validity Indicators ¹	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 16, "Machine-Check Architecture," for more information.

10. Updates to Chapter 18, Volume 3B

Change bars and green text show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- Update to the following figures to add the bus-lock detected bit: Figure 18-1, "Debug Registers," Figure 18-2, "DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture," and Figure 18-3, "IA32_DEBUGCTL MSR for Processors Based on Intel® Core™ Microarchitecture."
- Update to Section 18.2.3, "Debug Status Register (DR6)," to add bus-lock detected bit information.
- New Section 18.3.1.6, "OS Bus-Lock Detection," added.
- Update to Section 18.4.1, "IA32_DEBUGCTL MSR," to add the bus-lock detection bit.

CHAPTER 18

DEBUG, BRANCH PROFILE, TSC, AND INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FEATURES

NOTE

This chapter makes numerous references to last-branch recording (LBR) facilities. Unless noted otherwise, all such references in this chapter are to an earlier non-architectural form of the feature. Chapter 19 defines an architectural form of last-branch recording that is supported on newer processors.

Intel 64 and IA-32 architectures provide debug facilities for use in debugging code and monitoring performance. These facilities are valuable for debugging application software, system software, and multitasking operating systems. Debug support is accessed using debug registers (DR0 through DR7) and model-specific registers (MSRs):

- Debug registers hold the addresses of memory and I/O locations called breakpoints. Breakpoints are user-selected locations in a program, a data-storage area in memory, or specific I/O ports. They are set where a programmer or system designer wishes to halt execution of a program and examine the state of the processor by invoking debugger software. A debug exception (#DB) is generated when a memory or I/O access is made to a breakpoint address.
- MSRs monitor branches, interrupts, and exceptions; they record addresses of the last branch, interrupt or exception taken and the last branch taken before an interrupt or exception.
- Time stamp counter is described in Section 18.17, "Time-Stamp Counter."
- Features which allow monitoring of shared platform resources such as the L3 cache are described in Section 18.18, "Intel® Resource Director Technology (Intel® RDT) Monitoring Features."
- Features which enable control over shared platform resources are described in Section 18.19, "Intel® Resource Director Technology (Intel® RDT) Allocation Features."

18.1 OVERVIEW OF DEBUG SUPPORT FACILITIES

The following processor facilities support debugging and performance monitoring:

- **Debug exception (#DB)** — Transfers program control to a debug procedure or task when a debug event occurs.
- **Breakpoint exception (#BP)** — See breakpoint instruction (INT3) below.
- **Breakpoint-address registers (DR0 through DR3)** — Specifies the addresses of up to 4 breakpoints.
- **Debug status register (DR6)** — Reports the conditions that were in effect when a debug or breakpoint exception was generated.
- **Debug control register (DR7)** — Specifies the forms of memory or I/O access that cause breakpoints to be generated.
- **T (trap) flag, TSS** — Generates a debug exception (#DB) when an attempt is made to switch to a task with the T flag set in its TSS.
- **RF (resume) flag, EFLAGS register** — Suppresses multiple exceptions to the same instruction.
- **TF (trap) flag, EFLAGS register** — Generates a debug exception (#DB) after every execution of an instruction.
- **Breakpoint instruction (INT3)** — Generates a breakpoint exception (#BP) that transfers program control to the debugger procedure or task. This instruction is an alternative way to set instruction breakpoints. It is especially useful when more than four breakpoints are desired, or when breakpoints are being placed in the source code.

- **Last branch recording facilities** — Store branch records in the last branch record (LBR) stack MSR for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consist of a branch-from and a branch-to instruction address. Send branch records out on the system bus as branch trace messages (BTMs).

These facilities allow a debugger to be called as a separate task or as a procedure in the context of the current program or task. The following conditions can be used to invoke the debugger:

- Task switch to a specific task.
- Execution of the breakpoint instruction.
- Execution of any instruction.
- Execution of an instruction at a specified address.
- Read or write to a specified memory address/range.
- Write to a specified memory address/range.
- Input from a specified I/O address/range.
- Output to a specified I/O address/range.
- Attempt to change the contents of a debug register.

18.2 DEBUG REGISTERS

Eight debug registers (see Figure 18-1 for 32-bit operation and Figure 18-2 for 64-bit operation) control the debug operation of the processor. These registers can be written to and read using the move to/from debug register form of the MOV instruction. A debug register may be the source or destination operand for one of these instructions.

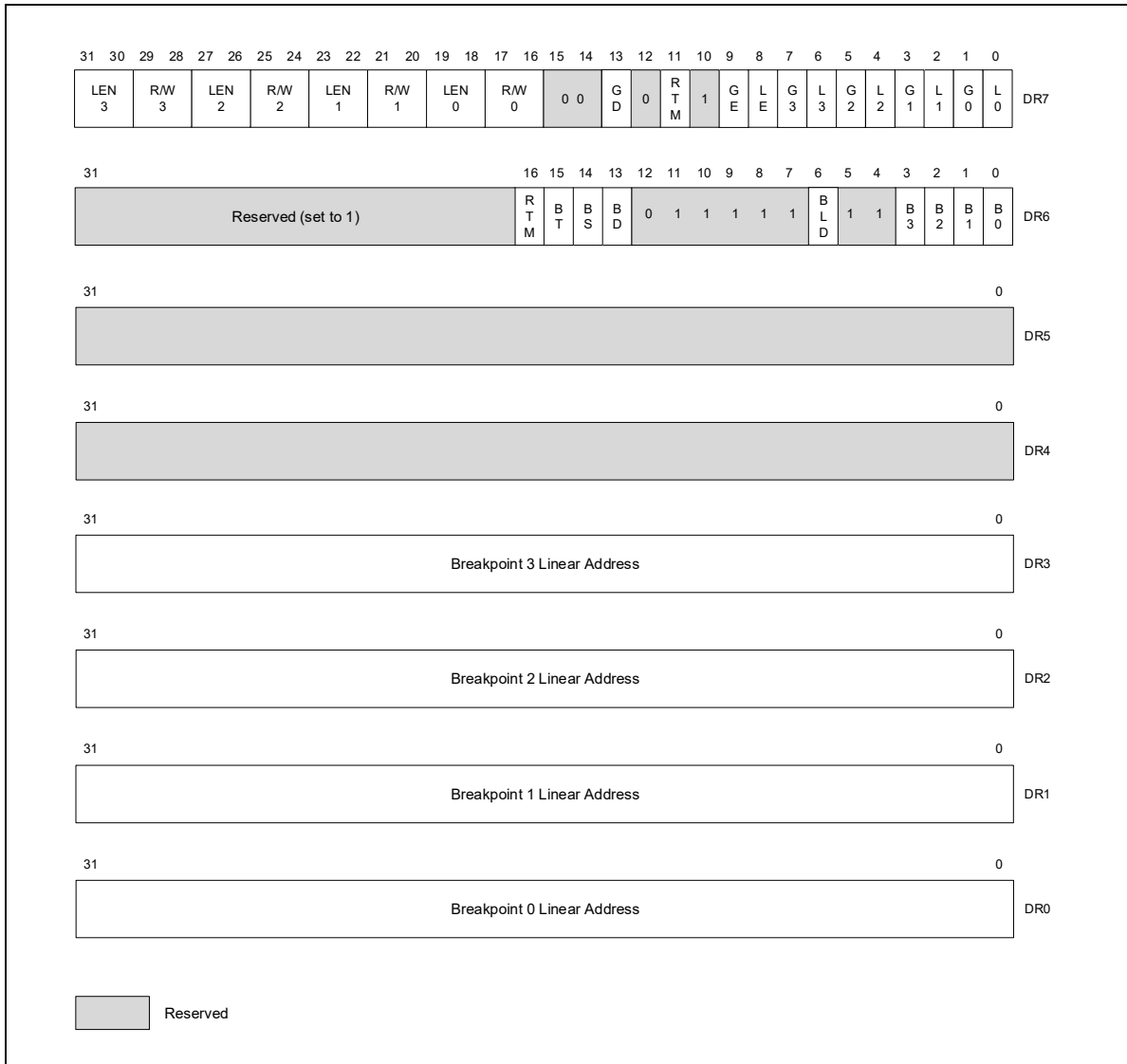


Figure 18-1. Debug Registers

Debug registers are privileged resources; a MOV instruction that accesses these registers can only be executed in real-address mode, in SMM or in protected mode at a CPL of 0. An attempt to read or write the debug registers from any other privilege level generates a general-protection exception (#GP).

The primary function of the debug registers is to set up and monitor from 1 to 4 breakpoints, numbered 0 through 3. For each breakpoint, the following information can be specified:

- The linear address where the breakpoint is to occur.
- The length of the breakpoint location: 1, 2, 4, or 8 bytes (refer to the notes in Section 18.2.4).
- The operation that must be performed at the address for a debug exception to be generated.
- Whether the breakpoint is enabled.
- Whether the breakpoint condition was present when the debug exception was generated.

The following paragraphs describe the functions of flags and fields in the debug registers.

18.2.1 Debug Address Registers (DR0-DR3)

Each of the debug-address registers (DR0 through DR3) holds the 32-bit linear address of a breakpoint (see Figure 18-1). Breakpoint comparisons are made before physical address translation occurs. The contents of debug register DR7 further specifies breakpoint conditions.

18.2.2 Debug Registers DR4 and DR5

Debug registers DR4 and DR5 are reserved when debug extensions are enabled (when the DE flag in control register CR4 is set) and attempts to reference the DR4 and DR5 registers cause invalid-opcode exceptions (#UD). When debug extensions are not enabled (when the DE flag is clear), these registers are aliased to debug registers DR6 and DR7.

18.2.3 Debug Status Register (DR6)

The debug status register (DR6) reports debug conditions that were sampled at the time the last debug exception was generated (see Figure 18-1). Updates to this register only occur when an exception is generated. The flags in this register show the following information:

- **B0 through B3 (breakpoint condition detected) flags (bits 0 through 3)** — Indicates (when set) that its associated breakpoint condition was met when a debug exception was generated. These flags are set if the condition described for each breakpoint by the LEN_n and R/W_n flags in debug control register DR7 is true. They may or may not be set if the breakpoint is not enabled by the Ln or the Gn flags in register DR7. Therefore on a #DB, a debug handler should check only those B0-B3 bits which correspond to an enabled breakpoint.
- **BLD (bus-lock detected) flag (bit 11)** — Indicates (when **clear**) that the debug exception was triggered by the assertion of a bus lock when $CPL > 0$ and OS bus-lock detection was enabled (see Section 18.3.1.6). Other debug exceptions do not modify this bit. To avoid confusion in identifying debug exceptions, software debug-exception handlers should set bit 11 to 1 before returning. (Software that never enables OS bus-lock detection need not do this as $DR6[11] = 1$ following reset.) This bit is always 1 if the processor does not support OS bus-lock detection.
- **BD (debug register access detected) flag (bit 13)** — Indicates that the next instruction in the instruction stream accesses one of the debug registers (DR0 through DR7). This flag is enabled when the GD (general detect) flag in debug control register DR7 is set. See Section 18.2.4, “Debug Control Register (DR7),” for further explanation of the purpose of this flag.
- **BS (single step) flag (bit 14)** — Indicates (when set) that the debug exception was triggered by the single-step execution mode (enabled with the TF flag in the EFLAGS register). The single-step mode is the highest-priority debug exception. When the BS flag is set, any of the other debug status bits also may be set.
- **BT (task switch) flag (bit 15)** — Indicates (when set) that the debug exception resulted from a task switch where the T flag (debug trap flag) in the TSS of the target task was set. See Section 8.2.1, “Task-State Segment (TSS),” for the format of a TSS. There is no flag in debug control register DR7 to enable or disable this exception; the T flag of the TSS is the only enabling flag.
- **RTM (restricted transactional memory) flag (bit 16)** — Indicates (when **clear**) that a debug exception (#DB) or breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 18.3.3). This bit is set for any other debug exception (including all those that occur when advanced debugging of RTM transactional regions is not enabled). This bit is always 1 if the processor does not support RTM.

Certain debug exceptions may clear bits 0-3. The remaining contents of the DR6 register are never cleared by the processor. To avoid confusion in identifying debug exceptions, debug handlers should clear the register (except bit 16, which they should set) before returning to the interrupted task.

18.2.4 Debug Control Register (DR7)

The debug control register (DR7) enables or disables breakpoints and sets breakpoint conditions (see Figure 18-1). The flags and fields in this register control the following things:

- **L0 through L3 (local breakpoint enable) flags (bits 0, 2, 4, and 6)** — Enables (when set) the breakpoint condition for the associated breakpoint for the current task. When a breakpoint condition is detected and its associated L_n flag is set, a debug exception is generated. The processor automatically clears these flags on every task switch to avoid unwanted breakpoint conditions in the new task.
- **G0 through G3 (global breakpoint enable) flags (bits 1, 3, 5, and 7)** — Enables (when set) the breakpoint condition for the associated breakpoint for all tasks. When a breakpoint condition is detected and its associated G_n flag is set, a debug exception is generated. The processor does not clear these flags on a task switch, allowing a breakpoint to be enabled for all tasks.
- **LE and GE (local and global exact breakpoint enable) flags (bits 8, 9)** — This feature is not supported in the P6 family processors, later IA-32 processors, and Intel 64 processors. When set, these flags cause the processor to detect the exact instruction that caused a data breakpoint condition. For backward and forward compatibility with other Intel processors, we recommend that the LE and GE flags be set to 1 if exact breakpoints are required.
- **RTM (restricted transactional memory) flag (bit 11)** — Enables (when set) advanced debugging of RTM transactional regions (see Section 18.3.3). This advanced debugging is enabled only if IA32_DEBUGCTL.RTM is also set.
- **GD (general detect enable) flag (bit 13)** — Enables (when set) debug-register protection, which causes a debug exception to be generated prior to any MOV instruction that accesses a debug register. When such a condition is detected, the BD flag in debug status register DR6 is set prior to generating the exception. This condition is provided to support in-circuit emulators.

When the emulator needs to access the debug registers, emulator software can set the GD flag to prevent interference from the program currently executing on the processor.

The processor clears the GD flag upon entering to the debug exception handler, to allow the handler access to the debug registers.

- **R/W0 through R/W3 (read/write) fields (bits 16, 17, 20, 21, 24, 25, 28, and 29)** — Specifies the breakpoint condition for the corresponding breakpoint. The DE (debug extensions) flag in control register CR4 determines how the bits in the R/W_n fields are interpreted. When the DE flag is set, the processor interprets bits as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Break on I/O reads or writes.
- 11 — Break on data reads or writes but not instruction fetches.

When the DE flag is clear, the processor interprets the R/W_n bits the same as for the Intel386™ and Intel486™ processors, which is as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Undefined.
- 11 — Break on data reads or writes but not instruction fetches.

- **LEN0 through LEN3 (Length) fields (bits 18, 19, 22, 23, 26, 27, 30, and 31)** — Specify the size of the memory location at the address specified in the corresponding breakpoint address register (DR0 through DR3). These fields are interpreted as follows:

- 00 — 1-byte length.
- 01 — 2-byte length.
- 10 — Undefined (or 8 byte length, see note below).
- 11 — 4-byte length.

If the corresponding RW_n field in register DR7 is 00 (instruction execution), then the LEN_n field should also be 00. The effect of using other lengths is undefined. See Section 18.2.5, “Breakpoint Field Recognition,” below.

NOTES

For Pentium® 4 and Intel® Xeon® processors with a CPUID signature corresponding to family 15 (model 3, 4, and 6), break point conditions permit specifying 8-byte length on data read/write with an of encoding 10B in the LEN_n field.

Encoding 10B is also supported in processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture, the respective CPUID signatures corresponding to family 6, model 15, and family 6, DisplayModel value 23 (see the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). The Encoding 10B is supported in processors based on Intel Atom® microarchitecture, with CPUID signature of family 6, DisplayModel value 1CH. The encoding 10B is undefined for other processors.

18.2.5 Breakpoint Field Recognition

Breakpoint address registers (debug registers DR0 through DR3) and the LEN_n fields for each breakpoint define a range of sequential byte addresses for a data or I/O breakpoint. The LEN_n fields permit specification of a 1-, 2-, 4- or 8-byte range, beginning at the linear address specified in the corresponding debug register (DR_n). Two-byte ranges must be aligned on word boundaries; 4-byte ranges must be aligned on doubleword boundaries, 8-byte ranges must be aligned on quadword boundaries. I/O addresses are zero-extended (from 16 to 32 bits, for comparison with the breakpoint address in the selected debug register). These requirements are enforced by the processor; it uses LEN_n field bits to mask the lower address bits in the debug registers. Unaligned data or I/O breakpoint addresses do not yield valid results.

A data breakpoint for reading or writing data is triggered if any of the bytes participating in an access is within the range defined by a breakpoint address register and its LEN_n field. Table 18-1 provides an example setup of debug registers and data accesses that would subsequently trap or not trap on the breakpoints.

A data breakpoint for an unaligned operand can be constructed using two breakpoints, where each breakpoint is byte-aligned and the two breakpoints together cover the operand. The breakpoints generate exceptions only for the operand, not for neighboring bytes.

Instruction breakpoint addresses must have a length specification of 1 byte (the LEN_n field is set to 00). Instruction breakpoints for other operand sizes are undefined. The processor recognizes an instruction breakpoint address only when it points to the first byte of an instruction. If the instruction has prefixes, the breakpoint address must point to the first prefix.

Table 18-1. Breakpoint Examples

Debug Register Setup			
Debug Register	R/W _n	Breakpoint Address	LEN _n
DR0	R/W0 = 11 (Read/Write)	A0001H	LEN0 = 00 (1 byte)
DR1	R/W1 = 01 (Write)	A0002H	LEN1 = 00 (1 byte)
DR2	R/W2 = 11 (Read/Write)	B0002H	LEN2 = 01) (2 bytes)
DR3	R/W3 = 01 (Write)	C0000H	LEN3 = 11 (4 bytes)
Data Accesses			
Operation		Address	Access Length (In Bytes)
Data operations that trap			
- Read or write		A0001H	1
- Read or write		A0001H	2
- Write		A0002H	1
- Write		A0002H	2
- Read or write		B0001H	4
- Read or write		B0002H	1
- Read or write		B0002H	2
- Write		C0000H	4
- Write		C0001H	2
- Write		C0003H	1

Table 18-1. Breakpoint Examples (Contd.)

Debug Register Setup			
Debug Register	R/Wn	Breakpoint Address	LENn
Data operations that do not trap			
- Read or write		A0000H	1
- Read		A0002H	1
- Read or write		A0003H	4
- Read or write		B0000H	2
- Read		C0000H	2
- Read or write		C0004H	4

18.2.6 Debug Registers and Intel® 64 Processors

For Intel 64 architecture processors, debug registers DR0–DR7 are 64 bits. In 16-bit or 32-bit modes (protected mode and compatibility mode), writes to a debug register fill the upper 32 bits with zeros. Reads from a debug register return the lower 32 bits. In 64-bit mode, MOV DRn instructions read or write all 64 bits. Operand-size prefixes are ignored.

In 64-bit mode, the upper 32 bits of DR6 and DR7 are reserved and must be written with zeros. Writing 1 to any of the upper 32 bits results in a #GP(0) exception (see Figure 18-2). All 64 bits of DR0–DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0–DR3 are in the linear-address limits of the processor implementation (address matching is supported only on valid addresses generated by the processor implementation). Breakpoint conditions for 8-byte memory read/writes are supported in all modes.

18.3 DEBUG EXCEPTIONS

The Intel 64 and IA-32 architectures dedicate two interrupt vectors to handling debug exceptions: vector 1 (debug exception, #DB) and vector 3 (breakpoint exception, #BP). The following sections describe how these exceptions are generated and typical exception handler operations.

18.3.1 Debug Exception (#DB)—Interrupt Vector 1

The debug-exception handler is usually a debugger program or part of a larger software system. The processor generates a debug exception for any of several conditions. The debugger checks flags in the DR6 and DR7 registers to determine which condition caused the exception and which other conditions might apply. Table 18-2 shows the states of these flags following the generation of each kind of breakpoint condition.

Instruction-breakpoint and general-detect condition (see Section 18.3.1.3, “General-Detect Exception Condition”) result in faults; other debug-exception conditions result in traps. The debug exception may report one or both at one time. The following sections describe each class of debug exception.

The INT1 instruction generates a debug exception as a trap. Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

See also: Chapter 6, “Interrupt 1—Debug Exception (#DB),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

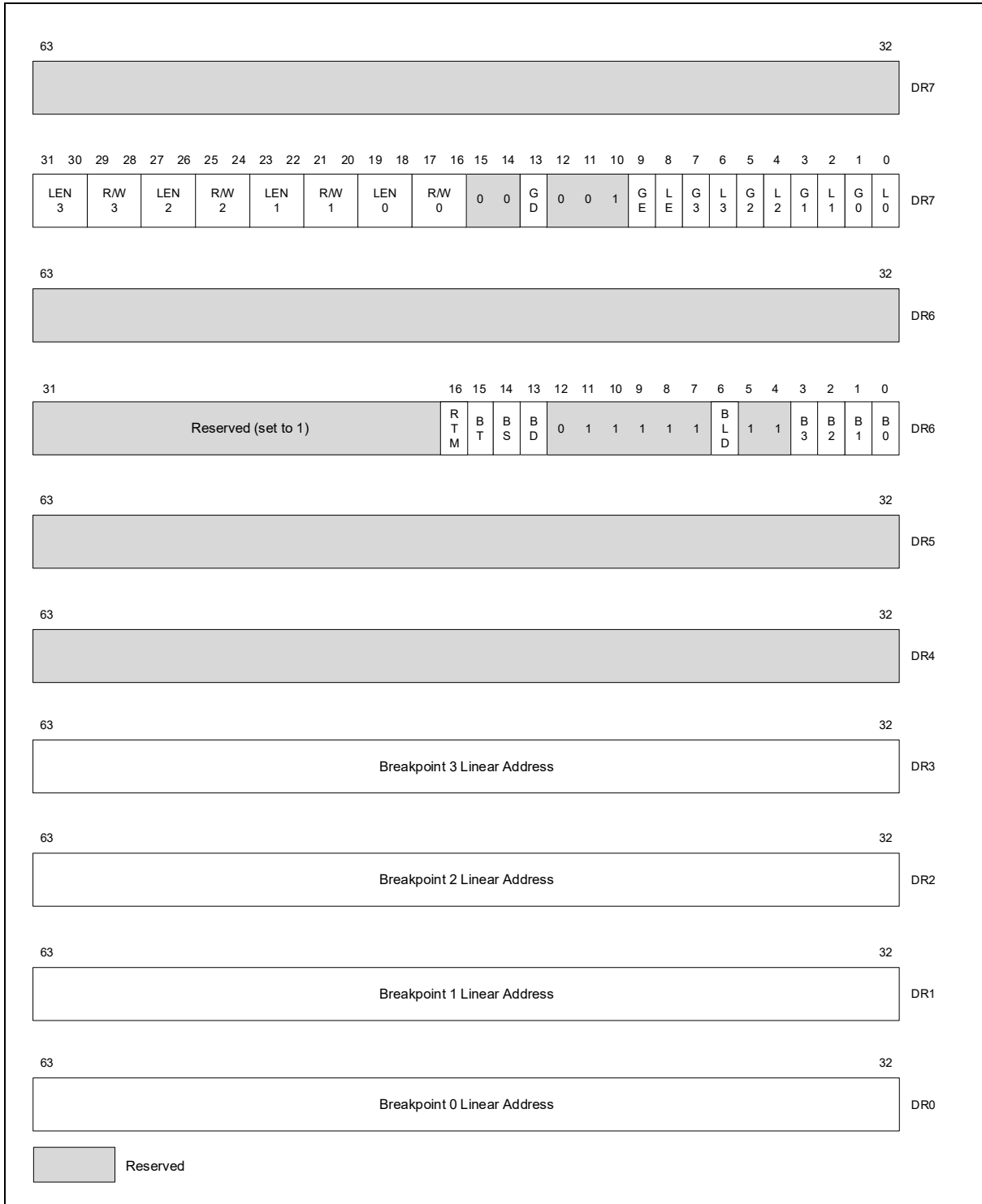


Figure 18-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

Table 18-2. Debug Exception Conditions

Debug or Breakpoint Condition	DR6 Flags Tested	DR7 Flags Tested	Exception Class
Single-step trap	BS = 1		Trap
Instruction breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 0	Fault
Data write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 1	Trap
I/O read or write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 2	Trap
Data read or write (but not instruction fetches), at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 3	Trap
General detect fault, resulting from an attempt to modify debug registers (usually in conjunction with in-circuit emulation)	BD = 1	None	Fault
Task switch	BT = 1	None	Trap
INT1 instruction	None	None	Trap

18.3.1.1 Instruction-Breakpoint Exception Condition

The processor reports an instruction breakpoint when it attempts to execute an instruction at an address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect instruction execution (R/W flag is set to 0). Upon reporting the instruction breakpoint, the processor generates a fault-class, debug exception (#DB) before it executes the target instruction for the breakpoint.

Instruction breakpoints are the highest priority debug exceptions. They are serviced before any other exceptions detected during the decoding or execution of an instruction. However, if an instruction breakpoint is placed on an instruction located immediately after a POP SS/MOV SS instruction, the breakpoint will be suppressed as if EFLAGS.RF were 1 (see the next paragraph and Section 6.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).

Because the debug exception for an instruction breakpoint is generated before the instruction is executed, if the instruction breakpoint is not removed by the exception handler; the processor will detect the instruction breakpoint again when the instruction is restarted and generate another debug exception. To prevent looping on an instruction breakpoint, the Intel 64 and IA-32 architectures provide the RF flag (resume flag) in the EFLAGS register (see Section 2.3, “System Flags and Fields in the EFLAGS Register,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). When the RF flag is set, the processor ignores instruction breakpoints.

All Intel 64 and IA-32 processors manage the RF flag as follows. The RF Flag is cleared at the start of the instruction after the check for instruction breakpoints, CS limit violations, and FP exceptions. Task Switches and IRETD/IRETQ instructions transfer the RF image from the TSS/stack to the EFLAGS register.

When calling an event handler, Intel 64 and IA-32 processors establish the value of the RF flag in the EFLAGS image pushed on the stack:

- For any fault-class exception except a debug exception generated in response to an instruction breakpoint, the value pushed for RF is 1.
- For any interrupt arriving after any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For any trap-class exception generated by any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For other cases, the value pushed for RF is the value that was in EFLAG.RF at the time the event handler was called. This includes:
 - Debug exceptions generated in response to instruction breakpoints
 - Hardware-generated interrupts arriving between instructions (including those arriving after the last iteration of a repeated string instruction)

- Trap-class exceptions generated after an instruction completes (including those generated after the last iteration of a repeated string instruction)
- Software-generated interrupts (RF is pushed as 0, since it was cleared at the start of the software interrupt)

As noted above, the processor does not set the RF flag prior to calling the debug exception handler for debug exceptions resulting from instruction breakpoints. The debug exception handler can prevent recurrence of the instruction breakpoint by setting the RF flag in the EFLAGS image on the stack. If the RF flag in the EFLAGS image is set when the processor returns from the exception handler, it is copied into the RF flag in the EFLAGS register by IRETD/IRETQ or a task switch that causes the return. The processor then ignores instruction breakpoints for the duration of the next instruction. (Note that the POPF, POPFD, and IRET instructions do not transfer the RF image into the EFLAGS register.) Setting the RF flag does not prevent other types of debug-exception conditions (such as, I/O or data breakpoints) from being detected, nor does it prevent non-debug exceptions from being generated.

For the Pentium processor, when an instruction breakpoint coincides with another fault-type exception (such as a page fault), the processor may generate one spurious debug exception after the second exception has been handled, even though the debug exception handler set the RF flag in the EFLAGS image. To prevent a spurious exception with Pentium processors, all fault-class exception handlers should set the RF flag in the EFLAGS image.

18.3.1.2 Data Memory and I/O Breakpoint Exception Conditions

Data memory and I/O breakpoints are reported when the processor attempts to access a memory or I/O address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect data or I/O accesses (R/W flag is set to 1, 2, or 3). The processor generates the exception after it executes the instruction that made the access, so these breakpoint condition causes a trap-class exception to be generated.

Because data breakpoints are traps, an instruction that writes memory overwrites the original data before the debug exception generated by a data breakpoint is generated. If a debugger needs to save the contents of a write breakpoint location, it should save the original contents before setting the breakpoint. The handler can report the saved value after the breakpoint is triggered. The address in the debug registers can be used to locate the new value stored by the instruction that triggered the breakpoint.

If a data breakpoint is detected during an iteration of a string instruction executed with fast-string operation (see Section 7.3.9.3 of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1), delivery of the resulting debug exception may be delayed until completion of the corresponding group of iterations.

Intel486 and later processors ignore the GE and LE flags in DR7. In Intel386 processors, exact data breakpoint matching does not occur unless it is enabled by setting the LE and/or the GE flags.

For repeated INS and OUTS instructions that generate an I/O-breakpoint debug exception, the processor generates the exception after the completion of the first iteration. Repeated INS and OUTS instructions generate a data-breakpoint debug exception after the iteration in which the memory address breakpoint location is accessed.

If an execution of the MOV or POP instruction loads the SS register and encounters a data breakpoint, the resulting debug exception is delivered after completion of the next instruction (the one after the MOV or POP).

Any pending data or I/O breakpoints are lost upon delivery of an exception. For example, if a machine-check exception (#MC) occurs following an instruction that encounters a data breakpoint (but before the resulting debug exception is delivered), the data breakpoint is lost. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, the data breakpoint is lost if the next instruction causes a fault.

Delivery of events due to INT n , INT3, or INTO does not cause a loss of data breakpoints. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, and the next instruction is software interrupt (INT n , INT3, or INTO), a debug exception (#DB) resulting from a data breakpoint will be delivered after the transition to the software-interrupt handler. The #DB handler should account for the fact that the #DB may have been delivered after a invocation of a software-interrupt handler, and in particular that the CPL may have changed between recognition of the data breakpoint and delivery of the #DB.

18.3.1.3 General-Detect Exception Condition

When the GD flag in DR7 is set, the general-detect debug exception occurs when a program attempts to access any of the debug registers (DR0 through DR7) at the same time they are being used by another application, such as an emulator or debugger. This protection feature guarantees full control over the debug registers when required. The

debug exception handler can detect this condition by checking the state of the BD flag in the DR6 register. The processor generates the exception before it executes the MOV instruction that accesses a debug register, which causes a fault-class exception to be generated.

18.3.1.4 Single-Step Exception Condition

The processor generates a single-step debug exception if (while an instruction is being executed) it detects that the TF flag in the EFLAGS register is set. The exception is a trap-class exception, because the exception is generated after the instruction is executed. The processor will not generate this exception after the instruction that sets the TF flag. For example, if the POPF instruction is used to set the TF flag, a single-step trap does not occur until after the instruction that follows the POPF instruction.

The processor clears the TF flag before calling the exception handler. If the TF flag was set in a TSS at the time of a task switch, the exception occurs after the first instruction is executed in the new task.

The TF flag normally is not cleared by privilege changes inside a task. The INT *n*, INT3, and INTO instructions, however, do clear this flag. Therefore, software debuggers that single-step code must recognize and emulate INT *n* or INTO instructions rather than executing them directly. To maintain protection, the operating system should check the CPL after any single-step trap to see if single stepping should continue at the current privilege level.

The interrupt priorities guarantee that, if an external interrupt occurs, single stepping stops. When both an external interrupt and a single-step interrupt occur together, the single-step interrupt is processed first. This operation clears the TF flag. After saving the return address or switching tasks, the external interrupt input is examined before the first instruction of the single-step handler executes. If the external interrupt is still pending, then it is serviced. The external interrupt handler does not run in single-step mode. To single step an interrupt handler, single step an INT *n* instruction that calls the interrupt handler.

If an occurrence of the MOV or POP instruction loads the SS register executes with EFLAGS.TF = 1, no single-step debug exception occurs following the MOV or POP instruction.

18.3.1.5 Task-Switch Exception Condition

The processor generates a debug exception after a task switch if the T flag of the new task's TSS is set. This exception is generated after program control has passed to the new task, and prior to the execution of the first instruction of that task. The exception handler can detect this condition by examining the BT flag of the DR6 register.

If entry 1 (#DB) in the IDT is a task gate, the T bit of the corresponding TSS should not be set. Failure to observe this rule will put the processor in a loop.

18.3.1.6 OS Bus-Lock Detection

OS bus-lock detection is a feature that causes the processor to generate a debug exception (called a **bus-lock detection debug exception**) if it detects that a bus lock has been asserted (see Section 9.1.2). Such an exception is a trap-class exception, because it is generated after execution of an instruction that asserts a bus lock. The exception thus does not prevent assertion of the bus lock. Delivery of a bus-lock detection debug exception clears DR6.BLD.

Software can enable OS bus-lock detection by setting IA32_DEBUGCTL.BLD[bit 2]. Bus-lock detection debug exceptions occur only if CPL > 0.

18.3.2 Breakpoint Exception (#BP)—Interrupt Vector 3

The breakpoint exception (interrupt 3) is caused by execution of an INT3 instruction. See Chapter 6, "Interrupt 3—Breakpoint Exception (#BP)." Debuggers use breakpoint exceptions in the same way that they use the breakpoint registers; that is, as a mechanism for suspending program execution to examine registers and memory locations. With earlier IA-32 processors, breakpoint exceptions are used extensively for setting instruction breakpoints.

With the Intel386 and later IA-32 processors, it is more convenient to set breakpoints with the breakpoint-address registers (DR0 through DR3). However, the breakpoint exception still is useful for breakpointing debuggers,

because a breakpoint exception can call a separate exception handler. The breakpoint exception is also useful when it is necessary to set more breakpoints than there are debug registers or when breakpoints are being placed in the source code of a program under development.

18.3.3 Debug Exceptions, Breakpoint Exceptions, and Restricted Transactional Memory (RTM)

Chapter 16, “Programming with Intel® Transactional Synchronization Extensions,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1 describes Restricted Transactional Memory (RTM). This is an instruction-set interface that allows software to identify **transactional regions** (or critical sections) using the XBEGIN and XEND instructions.

Execution of an RTM transactional region begins with an XBEGIN instruction. If execution of the region successfully reaches an XEND instruction, the processor ensures that all memory operations performed within the region appear to have occurred instantaneously when viewed from other logical processors. Execution of an RTM transaction region does not succeed if the processor cannot commit the updates atomically. When this happens, the processor rolls back the execution, a process referred to as a **transactional abort**. In this case, the processor discards all updates performed in the region, restores architectural state to appear as if the execution had not occurred, and resumes execution at a fallback instruction address that was specified with the XBEGIN instruction.

If debug exception (#DB) or breakpoint exception (#BP) occurs within an RTM transaction region, a transactional abort occurs, the processor sets EAX[4], and no exception is delivered.

Software can enable **advanced debugging of RTM transactional regions** by setting DR7.RTM[bit 11] and IA32_DEBUGCTL.RTM[bit 15]. If these bits are both set, the transactional abort caused by a #DB or #BP within an RTM transaction region does **not** resume execution at the fallback instruction address specified with the XBEGIN instruction that begin the region. Instead, execution is resumed at that XBEGIN instruction, and a #DB is delivered. (A #DB is delivered even if the transactional abort was caused by a #BP.) Such a #DB will clear DR6.RTM[bit 16] (all other debug exceptions set DR6[16]).

18.4 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING OVERVIEW

P6 family processors introduced the ability to set breakpoints on taken branches, interrupts, and exceptions, and to single-step from one branch to the next. This capability has been modified and extended in the Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors to allow logging of branch trace messages in a branch trace store (BTS) buffer in memory.

See the following sections for processor specific implementation of last branch, interrupt, and exception recording:

- Section 18.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)”
- Section 18.6, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Microarchitecture”
- Section 18.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture”
- Section 18.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture”
- Section 18.11, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Haswell Microarchitecture”
- Section 18.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture”
- Section 18.14, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors)”
- Section 18.15, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors)”
- Section 18.16, “Last Branch, Interrupt, and Exception Recording (P6 Family Processors)”

The following subsections of Section 18.4 describe common features of profiling branches. These features are generally enabled using the IA32_DEBUGCTL MSR (older processor may have implemented a subset or model-specific features, see definitions of MSR_DEBUGCTLA, MSR_DEBUGCTLB, MSR_DEBUGCTL).

18.4.1 IA32_DEBUGCTL MSR

The **IA32_DEBUGCTL** MSR provides bit field controls to enable debug trace interrupts, debug trace stores, trace messages enable, single stepping on branches, last branch record recording, and to control freezing of LBR stack or performance counters on a PMI request. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 18-3 for the MSR layout and the bullets below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the Section 18.5.1, “LBR Stack” (Intel® Core™2 Duo and Intel Atom® processor family) and Section 18.9.1, “LBR Stack” (processors based on Nehalem microarchitecture).
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 18.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **BLD (bus-lock detection) flag (bit 2)** — If this bit is set, OS bus-lock detection is enabled when CPL > 0. See Section 18.3.1.6.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bit 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

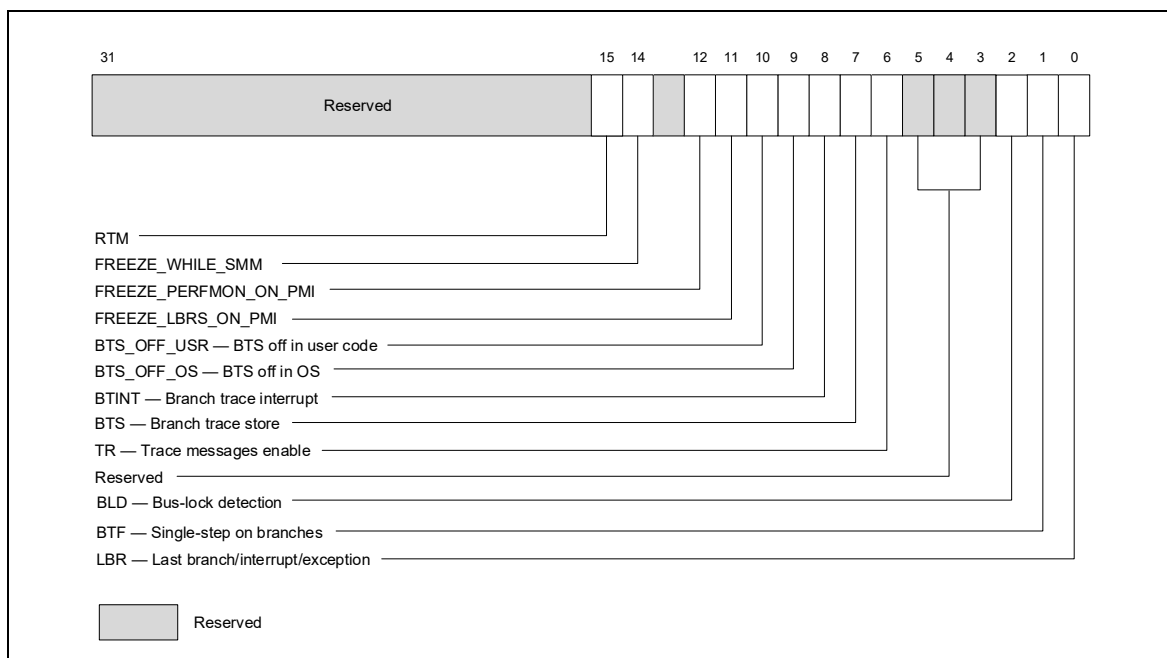


Figure 18-3. IA32_DEBUGCTL MSR for Processors Based on Intel® Core™ Microarchitecture

- **BTS_OFF_OS (branch trace off in privileged code) flag (bit 9)** — When set, BTS or BTM is skipped if CPL is 0. See Section 18.13.2.
- **BTS_OFF_USR (branch trace off in user code) flag (bit 10)** — When set, BTS or BTM is skipped if CPL is greater than 0. See Section 18.13.2.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — When set, the LBR stack is frozen on a hardware PMI request (e.g., when a counter overflows and is configured to trigger PMI). See Section 18.4.7 for details.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — When set, the performance counters (IA32_PMCx and IA32_FIXED_CTRx) are frozen on a PMI request. See Section 18.4.7 for details.
- **FREEZE_WHILE_SMM (bit 14)** — If this bit is set, upon the delivery of an SMI, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler. If Intel Thread Director support was enabled before transferring control to the SMI handler, then the processor will also reset the Intel Thread Director history (see Section 15.6.11 for more details about Intel Thread Director enable, reset, and history reset operations).
Subsequently, the enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its service. If Intel Thread Director support is enabled when RSM is executed, then the processor resets the Intel Thread Director history.
Note that system software must check if the processor supports the IA32_DEBUGCTL.FREEZE_WHILE_SMM control bit. IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 20.8 for details of detecting the presence of IA32_PERF_CAPABILITIES MSR.
- **RTM (bit 15)** — If this bit is set, advanced debugging of RTM transactional regions is enabled if DR7.RTM is also set. See Section 18.3.3.

18.4.2 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag (bit 0) in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions) in the LBR stack MSRs.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler. This action does not clear previously stored LBR stack MSRs.

A debugger can use the linear addresses in the LBR stack to re-set breakpoints in the breakpoint address registers (DR0 through DR3). This allows a backward trace from the manifestation of a particular bug toward its source.

On some processors, if the LBR flag is cleared and TR flag in the IA32_DEBUGCTL MSR remains set, the processor will continue to update LBR stack MSRs. This is because those processors use the entries in the LBR stack in the process of generating BTM/BTS records. A #DB does not automatically clear the TR flag.

18.4.3 Single-Stepping on Branches

When software sets both the BTF flag (bit 1) in the IA32_DEBUGCTL MSR and the TF flag in the EFLAGS register, the processor generates a single-step debug exception only after instructions that cause a branch.¹ This mechanism allows a debugger to single-step on control transfers caused by branches. This “branch single stepping” helps isolate a bug to a particular block of code before instruction single-stepping further narrows the search. The processor clears the BTF flag when it generates a debug exception. The debugger must set the BTF flag before resuming program execution to continue single-stepping on branches.

1. Executions of CALL, IRET, and JMP that cause task switches never cause single-step debug exceptions (regardless of the value of the BTF flag). A debugger desiring debug exceptions on switches to a task should set the T flag (debug trap flag) in the TSS of that task. See Section 8.2.1, “Task-State Segment (TSS).”

18.4.4 Branch Trace Messages

Setting the TR flag (bit 6) in the IA32_DEBUGCTL MSR enables branch trace messages (BTMs). Thereafter, when the processor detects a branch, exception, or interrupt, it sends a branch record out on the system bus as a BTM. A debugging device that is monitoring the system bus can read these messages and synchronize operations with taken branch, interrupt, and exception events.

When interrupts or exceptions occur in conjunction with a taken branch, additional BTMs are sent out on the bus, as described in Section 18.4.2, “Monitoring Branches, Exceptions, and Interrupts.”

For the P6 processor family, Pentium M processor family, and processors based on Intel Core microarchitecture, TR and LBR bits can not be set at the same time due to hardware limitation. The content of LBR stack is undefined when TR is set.

For processors with Intel NetBurst microarchitecture, Intel Atom processors, and Intel Core and related Intel Xeon processors both starting with the Nehalem microarchitecture, the processor can collect branch records in the LBR stack and at the same time send/store BTMs when both the TR and LBR flags are set in the IA32_DEBUGCTL MSR (or the equivalent MSR_DEBUGCTLA, MSR_DEBUGCTLB).

The following exception applies:

- BTM may not be observable on Intel Atom processor families that do not provide an externally visible system bus (i.e., processors based on the Silvermont microarchitecture or later).

18.4.4.1 Branch Trace Message Visibility

Branch trace message (BTM) visibility is implementation specific and limited to systems with a front side bus (FSB). BTMs may not be visible to newer system link interfaces or a system bus that deviates from a traditional FSB.

18.4.5 Branch Trace Store (BTS)

A trace of taken branches, interrupts, and exceptions is useful for debugging code by providing a method of determining the decision path taken to reach a particular code location. The LBR flag (bit 0) of IA32_DEBUGCTL provides a mechanism for capturing records of taken branches, interrupts, and exceptions and saving them in the last branch record (LBR) stack MSRs, setting the TR flag for sending them out onto the system bus as BTMs. The branch trace store (BTS) mechanism provides the additional capability of saving the branch records in a memory-resident BTS buffer, which is part of the DS save area. The BTS buffer can be configured to be circular so that the most recent branch records are always available or it can be configured to generate an interrupt when the buffer is nearly full so that all the branch records can be saved. The BTINT flag (bit 8) can be used to enable the generation of interrupt when the BTS buffer is full. See Section 18.4.9.2, “Setting Up the DS Save Area.” for additional details.

Setting this flag (BTS) alone can greatly reduce the performance of the processor. CPL-qualified branch trace storing mechanism can help mitigate the performance impact of sending/logging branch trace messages.

18.4.6 CPL-Qualified Branch Trace Mechanism

CPL-qualified branch trace mechanism is available to a subset of Intel 64 and IA-32 processors that support the branch trace storing mechanism. The processor supports the CPL-qualified branch trace mechanism if `CPUID.01H:ECX[bit 4] = 1`.

The CPL-qualified branch trace mechanism is described in Section 18.4.9.4. System software can selectively specify CPL qualification to not send/store Branch Trace Messages associated with a specified privilege level. Two bit fields, `BTS_OFF_USR` (bit 10) and `BTS_OFF_OS` (bit 9), are provided in the debug control register to specify the CPL of BTMs that will not be logged in the BTS buffer or sent on the bus.

18.4.7 Freezing LBR and Performance Counters on PMI

Many issues may generate a performance monitoring interrupt (PMI); a PMI service handler will need to determine cause to handle the situation. Two capabilities that allow a PMI service routine to improve branch tracing and performance monitoring are available for processors supporting architectural performance monitoring version 2 or

greater (i.e., CPUID.0AH:EAX[7:0] > 1). These capabilities provides the following interface in IA32_DEBUGCTL to reduce runtime overhead of PMI servicing, profiler-contributed skew effects on analysis or counter metrics:

- **Freezing LBRs on PMI (bit 11)**— Allows the PMI service routine to ensure the content in the LBR stack are associated with the target workload and not polluted by the branch flows of handling the PMI. Depending on the version ID enumerated by CPUID.0AH:EAX.ArchPerfMonVerID[bits 7:0], two flavors are supported:
 - Legacy Freeze_LBR_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID >1. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the LBR is frozen on the overflowed condition of the buffer area, the processor clears the LBR bit (bit 0) in IA32_DEBUGCTL. Software must then re-enable IA32_DEBUGCTL.LBR to resume recording branches. When using this feature, software should be careful about writes to IA32_DEBUGCTL to avoid re-enabling LBRs by accident if they were just disabled.
 - Streamlined Freeze_LBR_on_PMI is supported for ArchPerfMonVerID >= 4. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.LBR_Frz =1 to disable recording, but does not change the LBR bit (bit 0) in IA32_DEBUGCTL. The LBRs are frozen on the overflowed condition of the buffer area.
- **Freezing PMCs on PMI (bit 12)** — Allows the PMI service routine to ensure the content in the performance counters are associated with the target workload and not polluted by the PMI and activities within the PMI service routine. Depending on the version ID enumerated by CPUID.0AH:EAX.ArchPerfMonVerID[bits 7:0], two flavors are supported:
 - Legacy Freeze_Perfmon_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID >1. If IA32_DEBUGCTL.Freeze_Perfmon_On_PMI = 1, the performance counters are frozen on the counter overflowed condition when the processor clears the IA32_PERF_GLOBAL_CTRL MSR (see Figure 20-3). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 20.6.2.1, “Fixed-function Performance Counters”). Software must re-enable counts by writing 1s to the corresponding enable bits in IA32_PERF_GLOBAL_CTRL before leaving a PMI service routine to continue counter operation.
 - Streamlined Freeze_Perfmon_on_PMI is supported for ArchPerfMonVerID >= 4. The processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.CTR_Frz =1 to disable counting on a counter overflow condition, but does not change the IA32_PERF_GLOBAL_CTRL MSR.

Freezing LBRs and PMCs on PMIs (both legacy and streamlined operation) occur when one of the following applies:

- A performance counter had an overflow and was programmed to signal a PMI in case of an overflow.
 - For the general-purpose counters; enabling PMI is done by setting bit 20 of the IA32_PERFEVTSELx register.
 - For the fixed-function counters; enabling PMI is done by setting the 3rd bit in the corresponding 4-bit control field of the MSR_PERF_FIXED_CTR_CTRL register (see Figure 20-1) or IA32_FIXED_CTR_CTRL MSR (see Figure 20-2).
- The PEBS buffer is almost full and reaches the interrupt threshold.
- The BTS buffer is almost full and reaches the interrupt threshold.

Table 18-3 compares the interaction of the processor with the PMI handler using the legacy versus streamlined Freeza_Perfmon_On_PMI interface.

Table 18-3. Legacy and Streamlined Operation with Freeze_Perfmon_On_PMI = 1, Counter Overflowed

Legacy Freeze_Perfmon_On_PMI	Streamlined Freeze_Perfmon_On_PMI	Comment
Processor freezes the counters on overflow	Processor freezes the counters on overflow	Unchanged
Processor clears IA32_PERF_GLOBAL_CTRL	Processor set IA32_PERF_GLOBAL_STATUS.CTR_FTZ	
Handler reads IA32_PERF_GLOBAL_STATUS(0x38E) to examine which counter(s) overflowed	mask = RDMSR(0x38E)	Similar
Handler services the PMI	Handler services the PMI	Unchanged
Handler writes 1s to IA32_PERF_GLOBAL_OVF_CTL (0x390)	Handler writes mask into IA32_PERF_GLOBAL_OVF_RESET (0x390)	
Processor clears IA32_PERF_GLOBAL_STATUS	Processor clears IA32_PERF_GLOBAL_STATUS	Unchanged
Handler re-enables IA32_PERF_GLOBAL_CTRL	None	Reduced software overhead

18.4.8 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel 64 and IA-32 processor families. However, the number of MSRs in the LBR stack and the valid range of TOS pointer value can vary between different processor families. Table 18-4 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily_DisplayModel encoding (see CPUID instruction in Chapter 3 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A).

Table 18-4. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Component of an LBR Entry	Range of TOS Pointer
06_5CH, 06_5FH	32	FROM_IP, TO_IP	0 to 31
06_4EH, 06_5EH, 06_8EH, 06_9EH, 06_55H, 06_66H, 06_7AH, 06_67H, 06_6AH, 06_6CH, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_A5H, 06_A6H, 06_A7H, 06_A8H, 06_86H, 06_8AH, 06_96H, 06_9CH	32	FROM_IP, TO_IP, LBR_INFO ¹	0 to 31
06_3DH, 06_47H, 06_4FH, 06_56H, 06_3CH, 06_45H, 06_46H, 06_3FH, 06_2AH, 06_2DH, 06_3AH, 06_3EH, 06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	16	FROM_IP, TO_IP	0 to 15
06_17H, 06_1DH, 06_0FH	4	FROM_IP, TO_IP	0 to 3
06_37H, 06_4AH, 06_4CH, 06_4DH, 06_5AH, 06_5DH, 06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	8	FROM_IP, TO_IP	0 to 7

NOTES:

1. See Section 18.12.

The last branch recording mechanism tracks not only branch instructions (e.g., JMP, Jcc, LOOP, and CALL instructions), but also other operations that cause a change in the instruction pointer (e.g., external interrupts, traps, and faults). The branch recording mechanisms generally employ a set of MSRs, referred to as last branch record (LBR) stack. The size and exact locations of the LBR stack are generally model-specific (see Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4 for model-specific MSR addresses).

- **Last Branch Record (LBR) Stack** — The LBR consists of N pairs of MSRs (N is listed in the LBR stack size column of Table 18-4) that store source and destination address of recent branches (see Figure 18-3):
 - MSR_LASTBRANCH_0_FROM_IP (address is model specific) through the next consecutive (N-1) MSR address store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address is model specific) through the next consecutive (N-1) MSR address store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant M bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address is model specific) contains an M-bit pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. The valid range of the M-bit POS pointer is given in Table 18-4.

18.4.8.1 LBR Stack and Intel® 64 Processors

LBR MSRs are 64-bits. In 64-bit mode, last branch records store the full address. Outside of 64-bit mode, the upper 32-bits of branch addresses will be stored as 0.

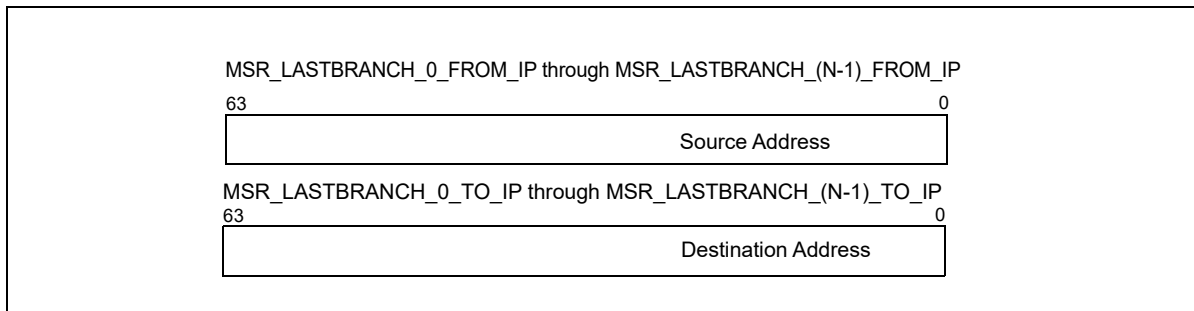


Figure 18-4. 64-bit Address Layout of LBR MSR

Software should query an architectural MSR IA32_PERF_CAPABILITIES[5:0] about the format of the address that is stored in the LBR stack. Four formats are defined by the following encoding:

- **000000B (32-bit record format)** — Stores 32-bit offset in current CS of respective source/destination,
- **000001B (64-bit LIP record format)** — Stores 64-bit linear address of respective source/destination,
- **000010B (64-bit EIP record format)** — Stores 64-bit offset (effective address) of respective source/destination.
- **000011B (64-bit EIP record format) and Flags** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction info is reported in the upper bit of 'FROM' registers in the LBR stack. See LBR stack details below for flag support and definition.
- **000100B (64-bit EIP record format), Flags, and TSX** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction and TSX info are reported in the upper bits of 'FROM' registers in the LBR stack.
- **000101B (64-bit EIP record format), Flags, TSX, and LBR_INFO** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction, TSX, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.
- **000110B (64-bit LIP record format), Flags, and Cycles** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction info is reported in the upper bits of

'FROM' registers in the LBR stack. Elapsed cycles since the last LBR update are reported in the upper 16 bits of the 'TO' registers in the LBR stack (see Section 18.6).

- **000111B (64-bit LIP record format), Flags, and LBR_INFO** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.

Processor's support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX[PERF_CAPAB_MSR] (bit 15).

18.4.8.2 LBR Stack and IA-32 Processors

The LBR MSRs in IA-32 processors introduced prior to Intel 64 architecture store the 32-bit "To Linear Address" and "From Linear Address" using the high and low half of each 64-bit MSR.

18.4.8.3 Last Exception Records and Intel 64 Architecture

Intel 64 and IA-32 processors also provide MSRs that store the branch record for the last branch taken prior to an exception or an interrupt. The location of the last exception record (LER) MSRs are model specific. The MSRs that store last exception records are 64-bits. If IA-32e mode is disabled, only the lower 32-bits of the address is recorded. If IA-32e mode is enabled, the processor writes 64-bit values into the MSR. In 64-bit mode, last exception records store 64-bit addresses; in compatibility mode, the upper 32-bits of last exception records are cleared.

18.4.9 BTS and DS Save Area

The **Debug store (DS)** feature flag (bit 21), returned by CPUID.1:EDX[21] indicates that the processor provides the debug store (DS) mechanism. The DS mechanism allows:

- BTMs to be stored in a memory-resident BTS buffer. See Section 18.4.5, "Branch Trace Store (BTS)."
- Processor event-based sampling (PEBS) also uses the DS save area provided by debug store mechanism. The capability of PEBS varies across different microarchitectures. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)," and the relevant PEBS sub-sections across the core PMU sections in Chapter 20, "Performance Monitoring."

When CPUID.1:EDX[21] is set:

- The BTS_UNAVAILABLE and PEBS_UNAVAILABLE flags in the IA32_MISC_ENABLE MSR indicate (when clear) the availability of the BTS and PEBS facilities, including the ability to set the BTS and BTINT bits in the appropriate DEBUGCTL MSR.
- The IA32_DS_AREA MSR exists and points to the DS save area.

The debug store (DS) save area is a software-designated area of memory that is used to collect the following two types of information:

- **Branch records** — When the BTS flag in the IA32_DEBUGCTL MSR is set, a branch record is stored in the BTS buffer in the DS save area whenever a taken branch, interrupt, or exception is detected.
- **PEBS records** — When a performance counter is configured for PEBS, a PEBS record is stored in the PEBS buffer in the DS save area after the counter overflow occurs. This record contains the architectural state of the processor (state of the 8 general purpose registers, EIP register, and EFLAGS register) at the next occurrence of the PEBS event that caused the counter to overflow. When the state information has been logged, the counter is automatically reset to a specified value, and event counting begins again. The content layout of a PEBS record varies across different implementations that support PEBS. See Section 20.6.2.4.2 for details of enumerating PEBS record format.

NOTES

Prior to processors based on the Goldmont microarchitecture, PEBS facility only supports a subset of implementation-specific precise events. See Section 20.5.3.1 for a PEBS enhancement that can generate records for both precise and non-precise events.

The DS save area and recording mechanism are disabled on INIT, processor Reset or transition to system-management mode (SMM) or IA-32e mode. It is similarly disabled on the generation of a machine-check exception on 45nm and 32nm Intel Atom processors and on processors with Netburst or Intel Core microarchitecture.

The BTS and PEBS facilities may not be available on all processors. The availability of these facilities is indicated by the `BTS_UNAVAILABLE` and `PEBS_UNAVAILABLE` flags, respectively, in the `IA32_MISC_ENABLE` MSR (see Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4).

The DS save area is divided into three parts: buffer management area, branch trace store (BTS) buffer, and PEBS buffer (see Figure 18-5). The buffer management area is used to define the location and size of the BTS and PEBS buffers. The processor then uses the buffer management area to keep track of the branch and/or PEBS records in their respective buffers and to record the performance counter reset value. The linear address of the first byte of the DS buffer management area is specified with the `IA32_DS_AREA` MSR.

The fields in the buffer management area are as follows:

- **BTS buffer base** — Linear address of the first byte of the BTS buffer. This address should point to a natural doubleword boundary.
- **BTS index** — Linear address of the first byte of the next BTS record to be written to. Initially, this address should be the same as the address in the BTS buffer base field.
- **BTS absolute maximum** — Linear address of the next byte past the end of the BTS buffer. This address should be a multiple of the BTS record size (12 bytes) plus 1.
- **BTS interrupt threshold** — Linear address of the BTS record on which an interrupt is to be generated. This address must point to an offset from the BTS buffer base that is a multiple of the BTS record size. Also, it must be several records short of the BTS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the BTS absolute maximum record.
- **PEBS buffer base** — Linear address of the first byte of the PEBS buffer. This address should point to a natural doubleword boundary.
- **PEBS index** — Linear address of the first byte of the next PEBS record to be written to. Initially, this address should be the same as the address in the PEBS buffer base field.
- **PEBS absolute maximum** — Linear address of the next byte past the end of the PEBS buffer. This address should be a multiple of the PEBS record size (40 bytes) plus 1.
- **PEBS interrupt threshold** — Linear address of the PEBS record on which an interrupt is to be generated. This address must point to an offset from the PEBS buffer base that is a multiple of the PEBS record size. Also, it must be several records short of the PEBS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the PEBS absolute maximum record.
- **PEBS counter reset value** — A 64-bit value that the counter is to be set to when a PEBS record is written. Bits beyond the size of the counter are ignored. This value allows state information to be collected regularly every time the specified number of events occur.

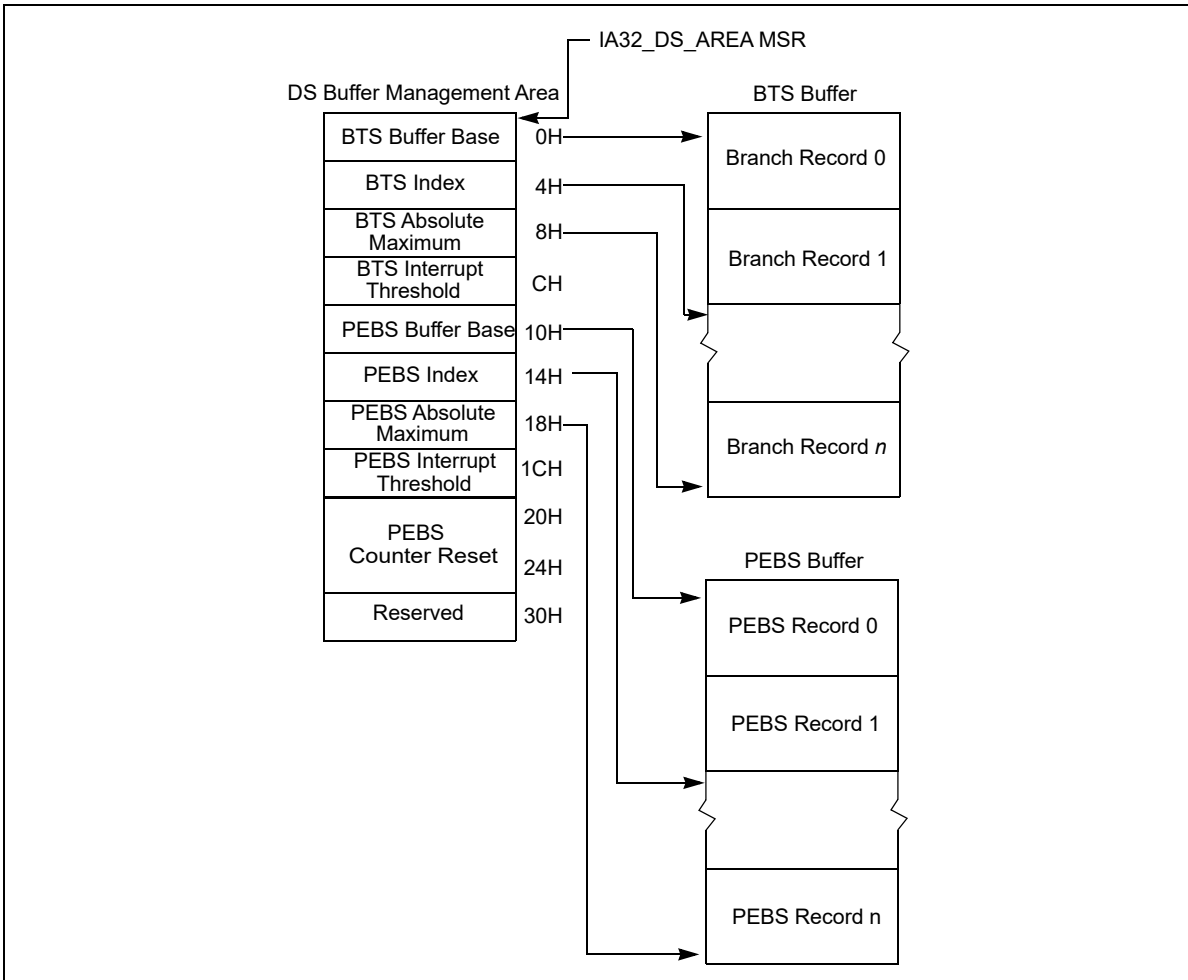


Figure 18-5. DS Save Area Example¹

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

Figure 18-6 shows the structure of a 12-byte branch record in the BTS buffer. The fields in each record are as follows:

- **Last branch from** — Linear address of the instruction from which the branch, interrupt, or exception was taken.
- **Last branch to** — Linear address of the branch target or the first instruction in the interrupt or exception service routine.
- **Branch predicted** — Bit 4 of field indicates whether the branch that was taken was predicted (set) or not predicted (clear).

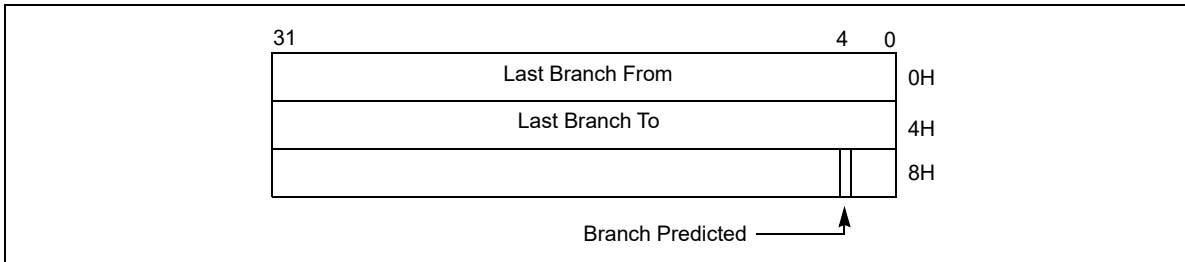


Figure 18-6. 32-bit Branch Trace Record Format

Figure 18-7 shows the structure of the 40-byte PEBS records. Nominally the register values are those at the beginning of the instruction that caused the event. However, there are cases where the registers may be logged in a partially modified state. The linear IP field shows the value in the EIP register translated from an offset into the current code segment to a linear address.

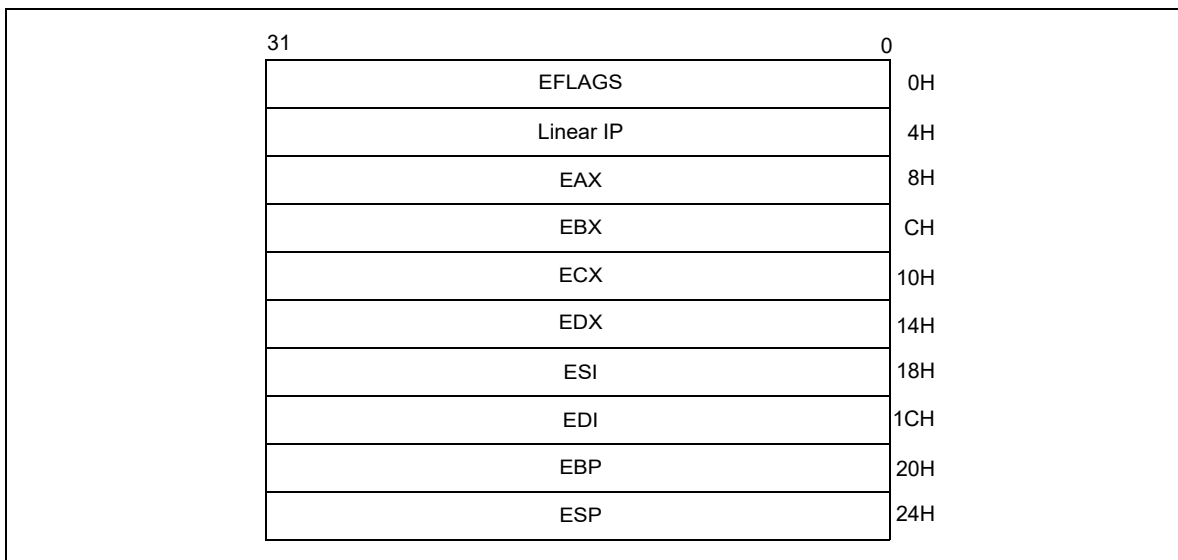


Figure 18-7. PEBS Record Format

18.4.9.1 64 Bit Format of the DS Save Area

When DTES64 = 1 (CPUID.1.ECX[2] = 1), the structure of the DS save area is shown in Figure 18-8.

When DTES64 = 0 (CPUID.1.ECX[2] = 0) and IA-32e mode is active, the structure of the DS save area is shown in Figure 18-8. If IA-32e mode is not active the structure of the DS save area is as shown in Figure 18-5.

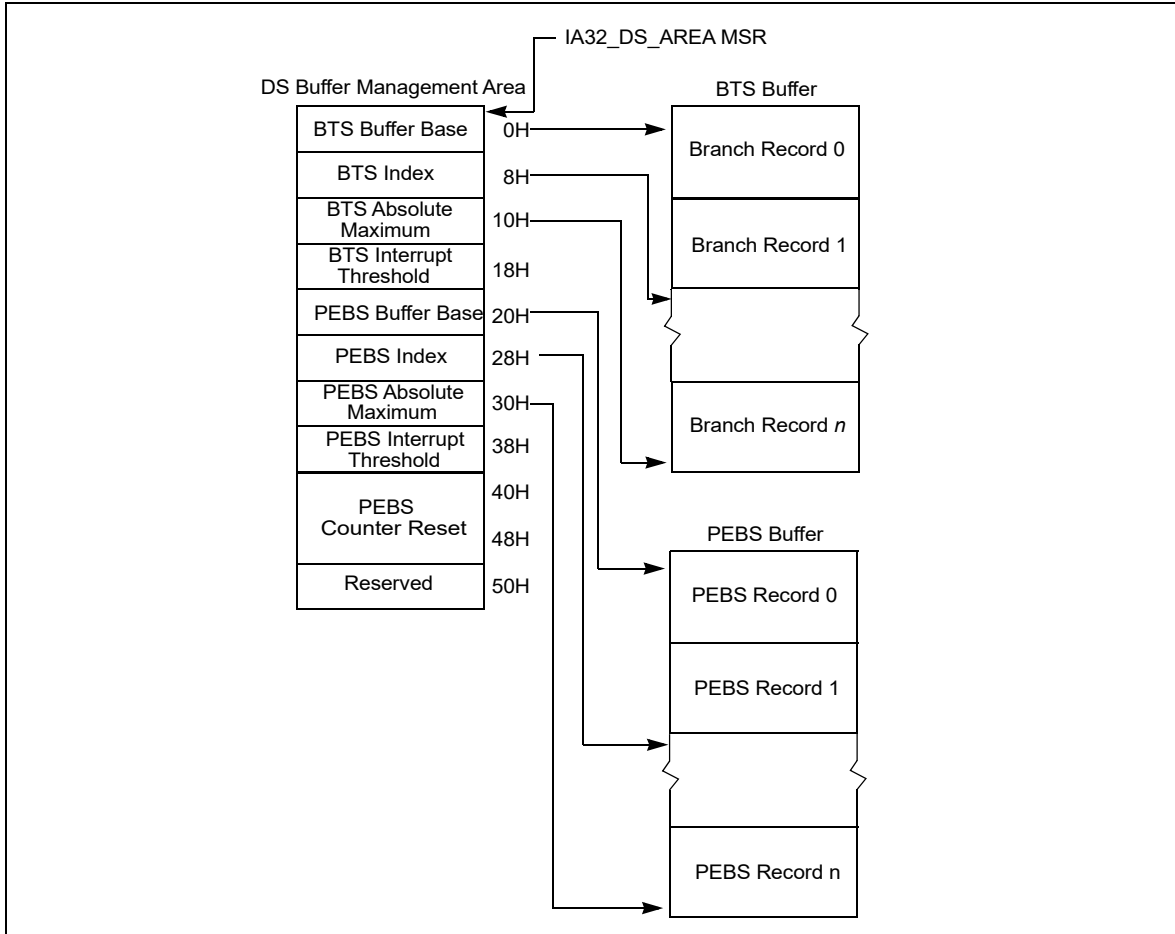


Figure 18-8. IA-32e Mode DS Save Area Example¹

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

The IA32_DS_AREA MSR holds the 64-bit linear address of the first byte of the DS buffer management area. The structure of a branch trace record is similar to that shown in Figure 18-6, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 18-9). The structure of a PEBS record is similar to that shown in Figure 18-7, but each field is 8 bytes in length and architectural states include register R8 through R15. This makes the size of a PEBS record in 64-bit mode 144 bytes (see Figure 18-10).

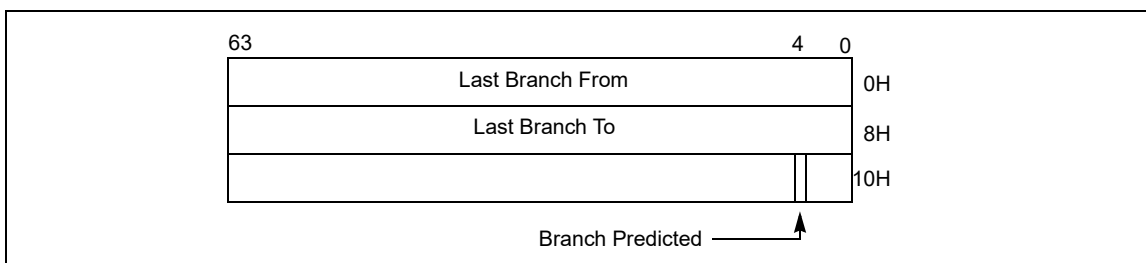


Figure 18-9. 64-bit Branch Trace Record Format

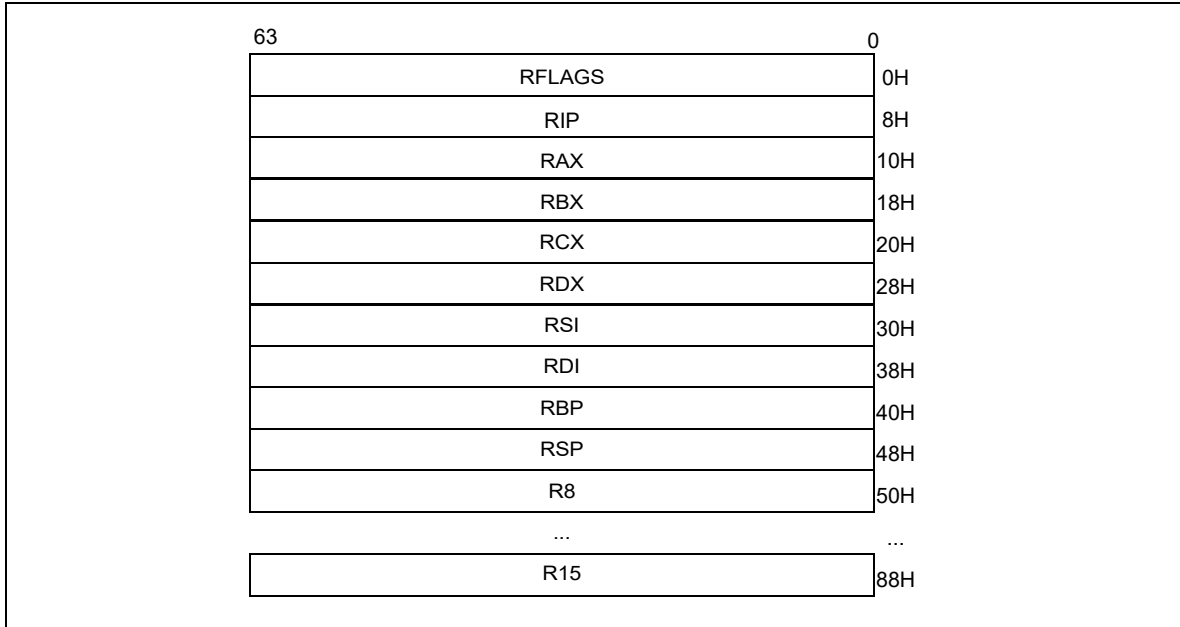


Figure 18-10. 64-bit PEBS Record Format

Fields in the buffer management area of a DS save area are described in Section 18.4.9.

The format of a branch trace record and a PEBS record are the same as the 64-bit record formats shown in Figures 18-9 and Figures 18-10, with the exception that the branch predicted bit is not supported by Intel Core microarchitecture or Intel Atom microarchitecture. The 64-bit record formats for BTS and PEBS apply to DS save area for all operating modes.

The procedures used to program IA32_DEBUGCTL MSR to set up a BTS buffer or a CPL-qualified BTS are described in Section 18.4.9.3 and Section 18.4.9.4.

Required elements for writing a DS interrupt service routine are largely the same on processors that support using DS Save area for BTS or PEBS records. However, on processors based on Intel NetBurst® microarchitecture, re-enabling counting requires writing to CCCRs. But a DS interrupt service routine on processors supporting architectural performance monitoring should:

- Re-enable the enable bits in IA32_PERF_GLOBAL_CTRL MSR if it is servicing an overflow PMI due to PEBS.
- Clear overflow indications by writing to IA32_PERF_GLOBAL_OVF_CTRL when a counting configuration is changed. This includes bit 62 (ClrOvfBuffer) and the overflow indication of counters used in either PEBS or general-purpose counting (specifically: bits 0 or 1; see Figures 20-3).

18.4.9.2 Setting Up the DS Save Area

To save branch records with the BTS buffer, the DS save area must first be set up in memory as described in the following procedure (See Section 20.6.2.4.1, “Setting up the PEBS Buffer,” for instructions for setting up a PEBS buffer, respectively, in the DS save area):

1. Create the DS buffer management information area in memory (see Section 18.4.9, “BTS and DS Save Area,” and Section 18.4.9.1, “64 Bit Format of the DS Save Area”). Also see the additional notes in this section.
2. Write the base linear address of the DS buffer management area into the IA32_DS_AREA MSR.
3. Set up the performance counter entry in the xAPIC LVT for fixed delivery and edge sensitive. See Section 11.5.1, “Local Vector Table.”
4. Establish an interrupt handler in the IDT for the vector associated with the performance counter entry in the xAPIC LVT.

5. Write an interrupt service routine to handle the interrupt. See Section 18.4.9.5, “Writing the DS Interrupt Service Routine.”

The following restrictions should be applied to the DS save area.

- The recording of branch records in the BTS buffer (or PEBS records in the PEBS buffer) may not operate properly if accesses to the linear addresses in any of the three DS save area sections cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do “lazy” page-table entry propagation for these pages. Some newer processor generations support “lazy” EPT page-table entry propagation for PEBS; see Section 20.3.9.1 and Section 20.9.5 for more information. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.
- The DS save area can be larger than a page, but the pages must be mapped to contiguous linear addresses. The buffer may share a page, so it need not be aligned on a 4-KByte boundary. For performance reasons, the base of the buffer must be aligned on a doubleword boundary and should be aligned on a cache line boundary.
- It is recommended that the buffer size for the BTS buffer and the PEBS buffer be an integer multiple of the corresponding record sizes.
- The precise event records buffer should be large enough to hold the number of precise event records that can occur while waiting for the interrupt to be serviced.
- The DS save area should be in kernel space. It must not be on the same page as code, to avoid triggering self-modifying code actions.
- There are no memory type restrictions on the buffers, although it is recommended that the buffers be designated as WB memory type for performance considerations.
- Either the system must be prevented from entering A20M mode while DS save area is active, or bit 20 of all addresses within buffer bounds must be 0.
- Pages that contain buffers must be mapped to the same physical addresses for all processes, such that any change to control register CR3 will not change the DS addresses.
- The DS save area is expected to be used only on systems with an enabled APIC. The LVT Performance Counter entry in the APCI must be initialized to use an interrupt gate instead of the trap gate.

18.4.9.3 Setting Up the BTS Buffer

Three flags in the MSR_DEBUGCTLA MSR (see Table 18-5), IA32_DEBUGCTL (see Figure 18-3), or MSR_DEBUGCTLB (see Figure 18-16) control the generation of branch records and storing of them in the BTS buffer; these are TR, BTS, and BTINT. The TR flag enables the generation of BTMs. The BTS flag determines whether the BTMs are sent out on the system bus (clear) or stored in the BTS buffer (set). BTMs cannot be simultaneously sent to the system bus and logged in the BTS buffer. The BTINT flag enables the generation of an interrupt when the BTS buffer is full. When this flag is clear, the BTS buffer is a circular buffer.

Table 18-5. IA32_DEBUGCTL Flag Encodings

TR	BTS	BTINT	Description
0	X	X	Branch trace messages (BTMs) off
1	0	X	Generate BTMs
1	1	0	Store BTMs in the BTS buffer, used here as a circular buffer
1	1	1	Store BTMs in the BTS buffer, and generate an interrupt when the buffer is nearly full

The following procedure describes how to set up a DS Save area to collect branch records in the BTS buffer:

1. Place values in the BTS buffer base, BTS index, BTS absolute maximum, and BTS interrupt threshold fields of the DS buffer management area to set up the BTS buffer in memory.
2. Set the TR and BTS flags in the IA32_DEBUGCTL for Intel Core Solo and Intel Core Duo processors or later processors (or MSR_DEBUGCTLA MSR for processors based on Intel NetBurst Microarchitecture; or MSR_DEBUGCTLB for Pentium M processors).

- Clear the BTINT flag in the corresponding IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR; or MSR_DEBUGCTLB) if a circular BTS buffer is desired.

NOTES

If the buffer size is set to less than the minimum allowable value (i.e., BTS absolute maximum < 1 + size of BTS record), the results of BTS is undefined.

In order to prevent generating an interrupt, when working with circular BTS buffer, SW need to set BTS interrupt threshold to a value greater than BTS absolute maximum (fields of the DS buffer management area). It's not enough to clear the BTINT flag itself only.

18.4.9.4 Setting Up CPL-Qualified BTS

If the processor supports CPL-qualified last branch recording mechanism, the generation of branch records and storing of them in the BTS buffer are determined by: TR, BTS, BTS_OFF_OS, BTS_OFF_USR, and BTINT. The encoding of these five bits are shown in Table 18-6.

Table 18-6. CPL-Qualified Branch Trace Store Encodings

TR	BTS	BTS_OFF_OS	BTS_OFF_USR	BTINT	Description
0	X	X	X	X	Branch trace messages (BTMs) off
1	0	X	X	X	Generates BTMs but do not store BTMs
1	1	0	0	0	Store all BTMs in the BTS buffer, used here as a circular buffer
1	1	1	0	0	Store BTMs with CPL > 0 in the BTS buffer
1	1	0	1	0	Store BTMs with CPL = 0 in the BTS buffer
1	1	1	1	X	Generate BTMs but do not store BTMs
1	1	0	0	1	Store all BTMs in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	1	0	1	Store BTMs with CPL > 0 in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	0	1	1	Store BTMs with CPL = 0 in the BTS buffer; generate an interrupt when the buffer is nearly full

18.4.9.5 Writing the DS Interrupt Service Routine

The BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector and interrupt service routine (called the debug store interrupt service routine or DS ISR). To handle BTS, non-precise event-based sampling, and PEBS interrupts: separate handler routines must be included in the DS ISR. Use the following guidelines when writing a DS ISR to handle BTS, non-precise event-based sampling, and/or PEBS interrupts.

- The DS interrupt service routine (ISR) must be part of a kernel driver and operate at a current privilege level of 0 to secure the buffer storage area.
- Because the BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector, the DS ISR must check for all the possible causes of interrupts from these facilities and pass control on to the appropriate handler.

BTS and PEBS buffer overflow would be the sources of the interrupt if the buffer index matches/exceeds the interrupt threshold specified. Detection of non-precise event-based sampling as the source of the interrupt is accomplished by checking for counter overflow.

- There must be separate save areas, buffers, and state for each processor in an MP system.
- Upon entering the ISR, branch trace messages and PEBS should be disabled to prevent race conditions during access to the DS save area. This is done by clearing TR flag in the IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR) and by clearing the precise event enable flag in the MSR_PEBS_ENABLE MSR. These settings should be restored to their original values when exiting the ISR.

- The processor will not disable the DS save area when the buffer is full and the circular mode has not been selected. The current DS setting must be retained and restored by the ISR on exit.
- After reading the data in the appropriate buffer, up to but not including the current index into the buffer, the ISR must reset the buffer index to the beginning of the buffer. Otherwise, everything up to the index will look like new entries upon the next invocation of the ISR.
- The ISR must clear the mask bit in the performance counter LVT entry.
- The ISR must re-enable the counters to count via IA32_PERF_GLOBAL_CTRL/IA32_PERF_GLOBAL_OVF_CTRL if it is servicing an overflow PMI due to PEBS (or via CCCR's ENABLE bit on processor based on Intel NetBurst microarchitecture).
- The Pentium 4 Processor and Intel Xeon Processor mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

18.5 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ 2 DUO AND INTEL ATOM® PROCESSORS)

The Intel Core 2 Duo processor family and Intel Xeon processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture provide last branch interrupt and exception recording. The facilities described in this section also apply to 45 nm and 32 nm Intel Atom processors. These capabilities are similar to those found in Pentium 4 processors, including support for the following facilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provide bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 18.4.1 for a description of the flags. See Figure 18-3 for the MSR layout.
- **Last branch record (LBR) stack** — There are a collection of MSR pairs that store the source and destination addresses related to recently executed branches. See Section 18.5.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts**
 - See Section 18.4.2 and Section 18.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.
 - 45 nm and 32 nm Intel Atom processors clear the TR flag when the FREEZE_LBRS_ON_PMI flag is set.
- **Branch trace messages** — See Section 18.4.4.
- **Last exception records** — See Section 18.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 18.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 18.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 18.4.7 for legacy Freeze_Perfmon_On_PMI operation.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 18.4.1.

18.5.1 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel Core 2, Intel Atom processor families, and Intel processors based on Intel NetBurst microarchitecture.

Four pairs of MSRs are supported in the LBR stack for Intel Core 2 processors families and Intel processors based on Intel NetBurst microarchitecture:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_3_FROM_IP (address 43H) store source addresses
 - MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_3_TO_IP (address 63H) store destination addresses

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 2 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

Eight pairs of MSRs are supported in the LBR stack for 45 nm and 32 nm Intel Atom processors:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_7_FROM_IP (address 47H) store source addresses
 - MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_7_TO_IP (address 67H) store destination addresses
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

The address format written in the FROM_IP/TO_IP MSRS may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 18.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

18.5.2 LBR Stack in Intel Atom® Processors based on the Silvermont Microarchitecture

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in Intel Atom processors based on the Silvermont and Airmont microarchitectures. Eight pairs of MSRs are supported in the LBR stack.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 18-11.

18.6 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Processors based on the Goldmont microarchitecture extend the capabilities described in Section 18.5.2 with the following enhancements:

- Supports new LBR format encoding 00110b in IA32_PERF_CAPABILITIES[5:0].
- Size of LBR stack increased to 32. Each entry includes MSR_LASTBRANCH_x_FROM_IP (address 0x680..0x69f) and MSR_LASTBRANCH_x_TO_IP (address 0x6c0..0x6df).
- LBR call stack filtering supported. The layout of MSR_LBR_SELECT is described in Table 18-13.
- Elapsed cycle information is added to MSR_LASTBRANCH_x_TO_IP. Format is shown in Table 18-7.
- Misprediction info is reported in the upper bits of MSR_LASTBRANCH_x_FROM_IP. MISHPRED bit format is shown in Table 18-8.
- Streamlined Freeze_LBRs_On_PMI operation; see Section 18.12.2.
- LBR MSRs may be cleared when MWAIT is used to request a C-state that is numerically higher than C1; see Section 18.12.3.

Table 18-7. MSR_LASTBRANCH_x_TO_IP for the Goldmont Microarchitecture

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 18.4.8.1 for address format.
Cycle Count (Saturating)	63:48	R/W	Elapsed core clocks since last update to the LBR stack.

18.7 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Next generation Intel Atom processors are based on the Goldmont Plus microarchitecture. Processors based on the Goldmont Plus microarchitecture extend the capabilities described in Section 18.6 with the following changes:

- Enumeration of new LBR format: encoding 00111b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 18.4.8.1.
- Each LBR stack entry consists of three MSRs:
 - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 18-9.
 - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 18-9.
 - MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data. Layout is the same as Table 18-16.

18.8 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR INTEL® XEON PHI™ PROCESSOR 7200/5200/3200

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in the Intel® Xeon Phi™ processor 7200/5200/3200 series based on the Knights Landing microarchitecture. Eight pairs of MSRs are supported in the LBR stack, per thread:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 680H) through MSR_LASTBRANCH_7_FROM_IP (address 687H) store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address 6C0H) through MSR_LASTBRANCH_7_TO_IP (address 6C7H) store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 18-11.

The address format written in the FROM_IP/TO_IP MSRS may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 18.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

18.9 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

The processors based on Nehalem microarchitecture and Westmere microarchitecture support last branch interrupt and exception recording. These capabilities are similar to those found in Intel Core 2 processors and add additional capabilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provides bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 18.4.1 for a description of the flags. See Figure 18-11 for the MSR layout.
- **Last branch record (LBR) stack** — There are 16 MSR pairs that store the source and destination addresses related to recently executed branches. See Section 18.9.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts** — See Section 18.4.2 and Section 18.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.

- **Branch trace messages** — The IA32_DEBUGCTL MSR provides bit fields for software to enable each logical processor to generate branch trace messages. See Section 18.4.4. However, not all BTM messages are observable using the Intel® QPI link.
- **Last exception records** — See Section 18.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 18.4.6 and Section 18.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 18.4.7 for legacy Freeze_LBRS_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 18.4.7 for legacy Freeze_Perfmon_On_PMI operation.
- **UNCORE_PMI_EN (bit 13)** — When set, this logical processor is enabled to receive an counter overflow interrupt form the uncore.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 18.4.1.

Processors based on Nehalem microarchitecture provide additional capabilities:

- **Independent control of uncore PMI** — The IA32_DEBUGCTL MSR provides a bit field (see Figure 18-11) for software to enable each logical processor to receive an uncore counter overflow interrupt.
- **LBR filtering** — Processors based on Nehalem microarchitecture support filtering of LBR based on combination of CPL and branch type conditions. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT.

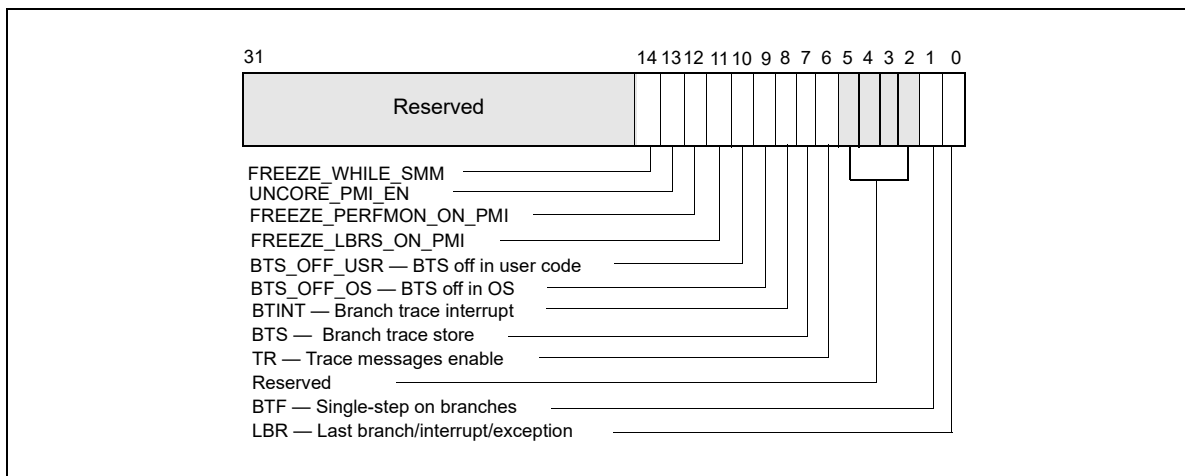


Figure 18-11. IA32_DEBUGCTL MSR for Processors Based on Nehalem Microarchitecture

18.9.1 LBR Stack

Processors based on Nehalem microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is shown in Table 18-8 and Table 18-9.

Table 18-8. MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch from” address. See Section 18.4.8.1 for address format.
SIGN_EXT	62:48	R/W	Signed extension of bit 47 of this register.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

Table 18-9. MSR_LASTBRANCH_x_TO_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 18.4.8.1 for address format
SIGN_EXT	63:48	R/W	Signed extension of bit 47 of this register.

Processors based on Nehalem microarchitecture have an LBR MSR Stack as shown in Table 18-10.

Table 18-10. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_1AH	16	0 to 15

18.9.2 Filtering of Last Branch Records

MSR_LBR_SELECT is cleared to zero at RESET, and LBR filtering is disabled, i.e., all branches will be captured. MSR_LBR_SELECT provides bit fields to specify the conditions of subsets of branches that will not be captured in the LBR. The layout of MSR_LBR_SELECT is shown in Table 18-11.

Table 18-11. MSR_LBR_SELECT for Nehalem Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

18.10 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SANDY BRIDGE MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 18.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture”, apply to processors based on Sandy Bridge microarchitecture. For processors based on Ivy Bridge microarchitecture, the same holds true.

One difference of note is that MSR_LBR_SELECT is shared between two logical processors in the same core. In Sandy Bridge microarchitecture, each logical processor has its own MSR_LBR_SELECT. The filtering semantics for “Near_ind_jmp” and “Near_rel_jmp” has been enhanced, see Table 18-12.

Table 18-12. MSR_LBR_SELECT for Sandy Bridge Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

18.11 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON HASWELL MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 18.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture”, apply to next generation processors based on Haswell microarchitecture.

The LBR facility also supports an alternate capability to profile call stack profiles. Configuring the LBR facility to conduct call stack profiling is by writing 1 to the MSR_LBR_SELECT.EN_CALLSTACK[bit 9]; see Table 18-13. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 18.10.

Table 18-13. MSR_LBR_SELECT for Haswell Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
EN_CALLSTACK ¹	9		Enable LBR stack to use LIFO filtering to capture Call stack profile
Reserved	63:10		Must be zero

NOTES:

1. Must set valid combination of bits 0-8 in conjunction with bit 9 (as described below), otherwise the contents of the LBR MSRs are undefined.

The call stack profiling capability is an enhancement of the LBR facility. The LBR stack is a ring buffer typically used to profile control flow transitions resulting from branches. However, the finite depth of the LBR stack often become less effective when profiling certain high-level languages (e.g., C++), where a transition of the execution flow is accompanied by a large number of leaf function calls, each of which returns an individual parameter to form the list

of parameters for the main execution function call. A long list of such parameters returned by the leaf functions would serve to flush the data captured in the LBR stack, often losing the main execution context.

When the call stack feature is enabled, the LBR stack will capture unfiltered call data normally, but as return instructions are executed the last captured branch record is flushed from the on-chip registers in a last-in first-out (LIFO) manner. Thus, branch information relative to leaf functions will not be captured, while preserving the call stack information of the main line execution path.

The configuration of the call stack facility is summarized below:

- Set IA32_DEBUGCTL.LBR (bit 0) to enable the LBR stack to capture branch records. The source and target addresses of the call branches will be captured in the 16 pairs of From/To LBR MSRs that form the LBR stack.
- Program the Top of Stack (TOS) MSR that points to the last valid from/to pair. This register is incremented by 1, modulo 16, before recording the next pair of addresses.
- Program the branch filtering bits of MSR_LBR_SELECT (bits 0:8) as desired.
- Program the MSR_LBR_SELECT to enable LIFO filtering of return instructions with:
 - The following bits in MSR_LBR_SELECT must be set to '1': JCC, NEAR_IND_JMP, NEAR_REL_JMP, FAR_BRANCH, EN_CALLSTACK;
 - The following bits in MSR_LBR_SELECT must be cleared: NEAR_REL_CALL, NEAR-IND_CALL, NEAR_RET;
 - At most one of CPL_EQ_0, CPL_NEQ_0 is set.

Note that when call stack profiling is enabled, “zero length calls” are excluded from writing into the LBRs. (A “zero length call” uses the attribute of the call instruction to push the immediate instruction pointer on to the stack and then pops off that address into a register. This is accomplished without any matching return on the call.)

18.11.1 LBR Stack Enhancement

Processors based on Haswell microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is enumerated by IA32_PERF_CAPABILITIES[5:0] = 04H, and is shown in Table 18-14 and Table 18-9.

Table 18-14. MSR_LASTBRANCH_x_FROM_IP with TSX Information

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch from” address. See Section 18.4.8.1 for address format.
SIGN_EXT	60:48	R/W	Signed extension of bit 47 of this register.
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region, or EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

18.12 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SKYLAKE MICROARCHITECTURE

Processors based on the Skylake microarchitecture provide a number of enhancement with storing last branch records:

- enumeration of new LBR format: encoding 00101b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 18.4.8.1.
- Each LBR stack entry consists of a triplets of MSRs:

- MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 18-9.
- MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 18-9.
- MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data.
- Size of LBR stack increased to 32.

Processors based on the Skylake microarchitecture supports the same LBR filtering capabilities as described in Table 18-13.

Table 18-15. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_4EH, 06_5EH	32	0 to 31

18.12.1 MSR_LBR_INFO_x MSR

The layout of each MSR_LBR_INFO_x MSR is shown in Table 18-16.

Table 18-16. MSR_LBR_INFO_x

Bit Field	Bit Offset	Access	Description
Cycle Count (saturating)	15:0	R/W	Elapsed core clocks since last update to the LBR stack.
Reserved	60:16	R/W	Reserved
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region OR EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

18.12.2 Streamlined Freeze_LBRs_On_PMI Operation

The FREEZE_LBRs_ON_PMI feature causes the LBRs to be frozen on a hardware request for a PMI. This prevents the LBRs from being overwritten by new branches, allowing the PMI handler to examine the control flow that preceded the PMI generation. Architectural performance monitoring version 4 and above supports a streamlined FREEZE_LBRs_ON_PMI operation for PMI service routine that replaces the legacy FREEZE_LBRs_ON_PMI operation (see Section 18.4.7).

While the legacy FREEZE_LBRs_ON_PMI clear the LBR bit in the IA32_DEBUGCTL MSR on a PMI request, the streamlined FREEZE_LBRs_ON_PMI will set the LBR_FRZ bit in IA32_PERF_GLOBAL_STATUS. Branches will not cause the LBRs to be updated when LBR_FRZ is set. Software can clear LBR_FRZ at the same time as it clears overflow bits by setting the LBR_FRZ bit as well as the needed overflow bit when writing to IA32_PERF_GLOBAL_STATUS_RESET MSR.

This streamlined behavior avoids race conditions between software and processor writes to IA32_DEBUGCTL that are possible with FREEZE_LBRs_ON_PMI clearing of the LBR enable.

18.12.3 LBR Behavior and Deep C-State

When MWAIT is used to request a C-state that is numerically higher than C1, then LBR state may be initialized to zero depending on optimized “waiting” state that is selected by the processor. The affected LBR states include the FROM, TO, INFO, LAST_BRANCH, LER, and LBR_TOS registers. The LBR enable bit and LBR_FROZEN bit are not affected. The LBR-time of the first LBR record inserted after an exit from such a C-state request will be zero.

18.13 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PROCESSORS BASED ON INTEL NETBURST® MICROARCHITECTURE)

Pentium 4 and Intel Xeon processors based on Intel NetBurst microarchitecture provide the following methods for recording taken branches, interrupts, and exceptions:

- Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consists of a branch-from and a branch-to instruction address.
- Send the branch records out on the system bus as branch trace messages (BTMs).
- Log BTMs in a memory-resident branch trace store (BTS) buffer.

To support these functions, the processor provides the following MSRs and related facilities:

- **MSR_DEBUGCTLA MSR** — Enables last branch, interrupt, and exception recording; single-stepping on taken branches; branch trace messages (BTMs); and branch trace store (BTS). This register is named DebugCtlMSR in the P6 family processors.
- **Debug store (DS) feature flag (CPUID.1:EDX.DS[bit 21])** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer.
- **CPL-qualified debug store (DS) feature flag (CPUID.1:ECX.DS-CPL[bit 4])** — Indicates that the processor provides a CPL-qualified debug store (DS) mechanism, which allows software to selectively skip sending and storing BTMs, according to specified current privilege level settings, into a memory-resident BTS buffer.
- **IA32_MISC_ENABLE MSR** — Indicates that the processor provides the BTS facilities.
- **Last branch record (LBR) stack** — The LBR stack is a circular stack that consists of four MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. The LBR stack consists of 16 MSR pairs (MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_15_FROM_IP and MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_15_TO_IP) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H].
- **Last branch record top-of-stack (TOS) pointer** — The TOS Pointer MSR contains a 2-bit pointer (0-3) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. This pointer becomes a 4-bit pointer (0-15) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H]. See also: Table 18-17, Figure 18-12, and Section 18.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **Last exception record** — See Section 18.13.3, “Last Exception Records.”

18.13.1 MSR_DEBUGCTLA MSR

The MSR_DEBUGCTLA MSR enables and disables the various last branch recording mechanisms described in the previous section. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode. A protected-mode operating system procedure is required to provide user access to this register. Figure 18-12 shows the flags in the MSR_DEBUGCTLA MSR. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. Each branch, interrupt, or exception is recorded as a 64-bit branch record. The processor clears this flag whenever a debug exception is generated (for example,

when an instruction or data breakpoint or a single-step trap occurs). See Section 18.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”

- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 18.4.3, “Single-Stepping on Branches.”
- **TR (trace message enable) flag (bit 2)** — When set, branch trace messages are enabled. Thereafter, when the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.4.4, “Branch Trace Messages.”

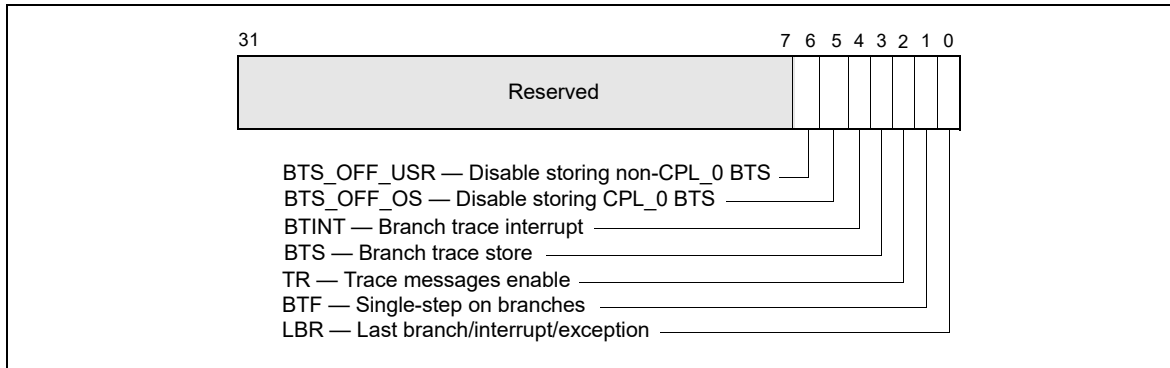


Figure 18-12. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **BTS (branch trace store) flag (bit 3)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 4)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.4.5, “Branch Trace Store (BTS).”
- **BTS_OFF_OS (disable ring 0 branch trace store) flag (bit 5)** — When set, enables the BTS facilities to skip sending/logging CPL_0 BTMs to the memory-resident BTS buffer. See Section 18.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **BTS_OFF_USR (disable ring 0 branch trace store) flag (bit 6)** — When set, enables the BTS facilities to skip sending/logging non-CPL_0 BTMs to the memory-resident BTS buffer. See Section 18.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”

NOTE

The initial implementation of BTS_OFF_USR and BTS_OFF_OS in MSR_DEBUGCTLA is shown in Figure 18-12. The BTS_OFF_USR and BTS_OFF_OS fields may be implemented on other model-specific debug control register at different locations.

See Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4 for a detailed description of each of the last branch recording MSRs.

18.13.2 LBR Stack for Processors Based on Intel NetBurst® Microarchitecture

The LBR stack is made up of LBR MSRs that are treated by the processor as a circular stack. The TOS pointer (MSR_LASTBRANCH_TOS MSR) points to the LBR MSR (or LBR MSR pair) that contains the most recent (last) branch record placed on the stack. Prior to placing a new branch record on the stack, the TOS is incremented by 1. When the TOS pointer reaches its maximum value, it wraps around to 0. See Table 18-17 and Figure 18-12.

Table 18-17. LBR MSR Stack Size and TOS Pointer Range for the Pentium® 4 and the Intel® Xeon® Processor Family

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
Family 0FH, Models 0H-02H; MSRs at locations 1DBH-1DEH.	4	0 to 3
Family 0FH, Models; MSRs at locations 680H-68FH.	16	0 to 15
Family 0FH, Model 03H; MSRs at locations 6C0H-6CFH.	16	0 to 15

The registers in the LBR MSR stack and the MSR_LASTBRANCH_TOS MSR are read-only and can be read using the RDMSR instruction.

Figure 18-13 shows the layout of a branch record in an LBR MSR (or MSR pair). Each branch record consists of two linear addresses, which represent the “from” and “to” instruction pointers for a branch, interrupt, or exception. The contents of the from and to addresses differ, depending on the source of the branch:

- **Taken branch** — If the record is for a taken branch, the “from” address is the address of the branch instruction and the “to” address is the target instruction of the branch.
- **Interrupt** — If the record is for an interrupt, the “from” address the return instruction pointer (RIP) saved for the interrupt and the “to” address is the address of the first instruction in the interrupt handler routine. The RIP is the linear address of the next instruction to be executed upon returning from the interrupt handler.
- **Exception** — If the record is for an exception, the “from” address is the linear address of the instruction that caused the exception to be generated and the “to” address is the address of the first instruction in the exception handler routine.

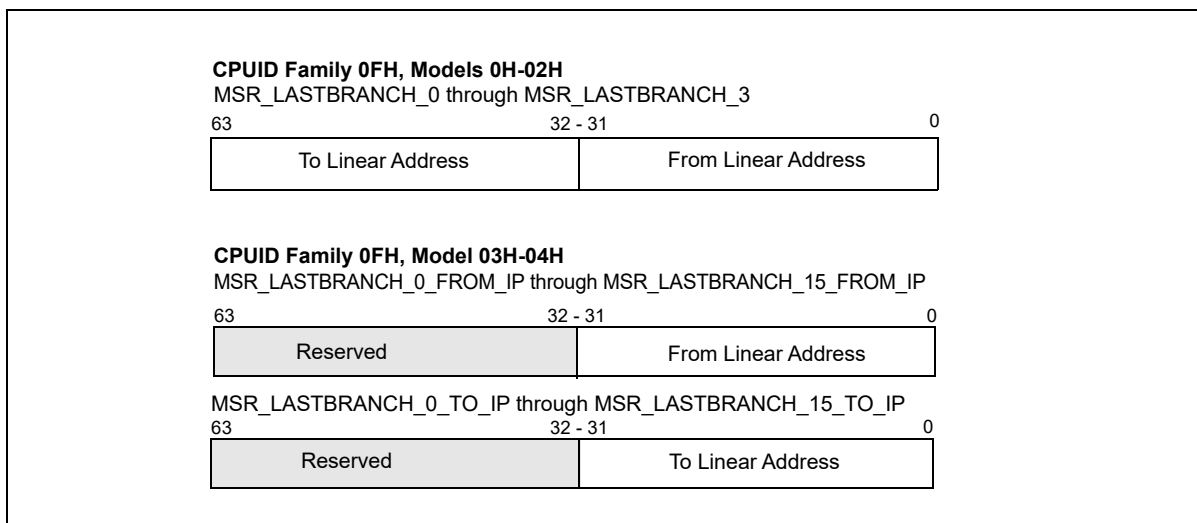


Figure 18-13. LBR MSR Branch Record Layout for the Pentium 4 and Intel® Xeon® Processor Family

Additional information is saved if an exception or interrupt occurs in conjunction with a branch instruction. If a branch instruction generates a trap type exception, two branch records are stored in the LBR stack: a branch record for the branch instruction followed by a branch record for the exception.

If a branch instruction is immediately followed by an interrupt, a branch record is stored in the LBR stack for the branch instruction followed by a record for the interrupt.

18.13.3 Last Exception Records

The Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors provide two MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate the functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

The MSR_LER_TO_LIP and MSR_LER_FROM_LIP MSRs contain a branch record for the last branch that the processor took prior to an exception or interrupt being generated.

18.14 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS)

Intel Core Solo and Intel Core Duo processors provide last branch interrupt and exception recording. This capability is almost identical to that found in Pentium 4 and Intel Xeon processors. There are differences in the stack and in some MSR names and locations.

Note the following:

- **IA32_DEBUGCTL MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 18-14 for the layout and the entries below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” below.
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 18.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

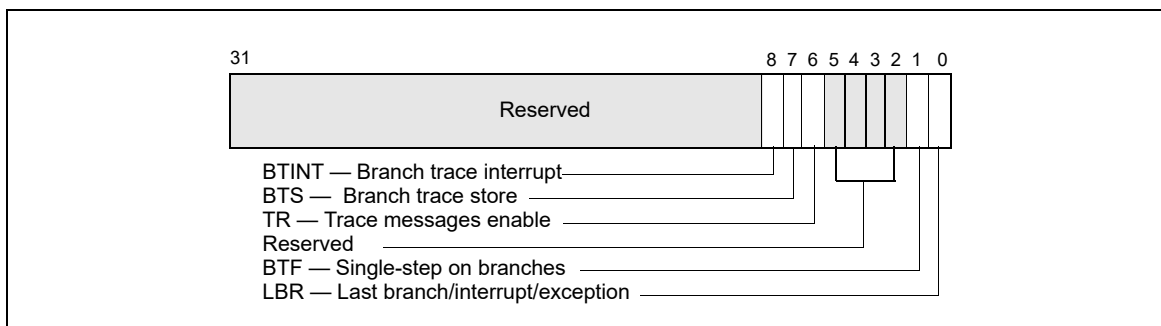


Figure 18-14. IA32_DEBUGCTL MSR for Intel® Core™ Solo and Intel® Core™ Duo Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 18.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address (MSR addresses start at 40H). See Figure 18-15.

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Intel Core Solo and Intel Core Duo processors, this MSR is located at register address 01C9H.

For compatibility, the Intel Core Solo and Intel Core Duo processors provide two 32-bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

For details, see Section 18.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture,” and Section 2.20, “MSRs In Intel® Core™ Solo and Intel® Core™ Duo Processors” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

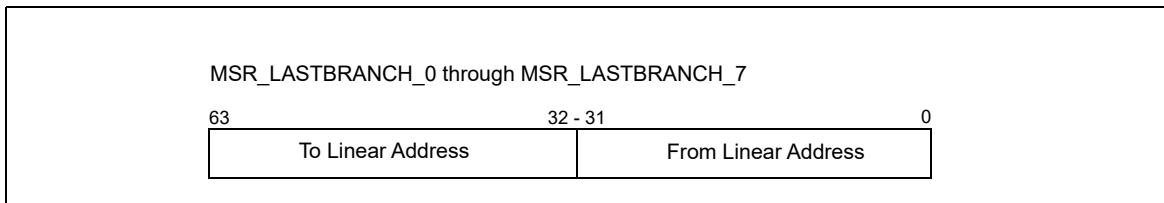


Figure 18-15. LBR Branch Record Layout for the Intel® Core™ Solo and Intel® Core™ Duo Processor

18.15 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PENTIUM M PROCESSORS)

Like the Pentium 4 and Intel Xeon processor family, Pentium M processors provide last branch interrupt and exception recording. The capability operates almost identically to that found in Pentium 4 and Intel Xeon processors. There are differences in the shape of the stack and in some MSR names and locations. Note the following:

- **MSR_DEBUGCTLB MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. For Pentium M processors, this MSR is located at register address 01D9H. See Figure 18-16 and the entries below for a description of the flags.
 - **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” bullet below.
 - **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 18.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
 - **PBi (performance monitoring/breakpoint pins) flags (bits 5-2)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BPi# pin when a breakpoint match occurs. When a PBi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
 - **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.4.4, “Branch Trace Messages,” for more information about the TR flag.
 - **BTS (branch trace store) flag (bit 7)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.4.9, “BTS and DS Save Area.”
 - **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

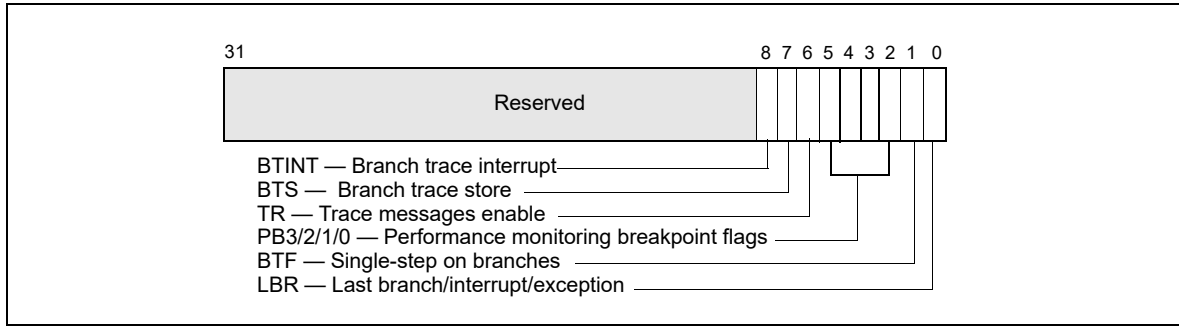


Figure 18-16. MSR_DEBUGCTLB MSR for Pentium M Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 18.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address. For Pentium M Processors, these pairs are located at register addresses 040H-047H. See Figure 18-17.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Pentium M Processors, this MSR is located at register address 01C9H.

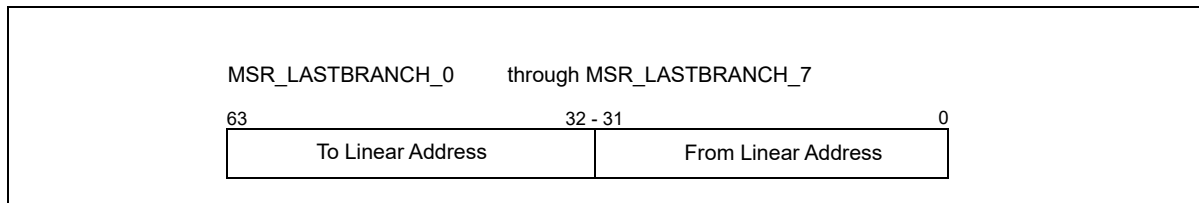


Figure 18-17. LBR Branch Record Layout for the Pentium M Processor

For more detail on these capabilities, see Section 18.13.3, “Last Exception Records,” and Section 2.21, “MSRs In the Pentium M Processor” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

18.16 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (P6 FAMILY PROCESSORS)

The P6 family processors provide five MSRs for recording the last branch, interrupt, or exception taken by the processor: DEBUGCTLMR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP. These registers can be used to collect last branch records, to set breakpoints on branches, interrupts, and exceptions, and to single-step from one branch to the next.

See Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4 for a detailed description of each of the last branch recording MSRs.

18.16.1 DEBUGCTLMR Register

The version of the DEBUGCTLMR register found in the P6 family processors enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.

A protected-mode operating system procedure is required to provide user access to this register. Figure 18-18 shows the flags in the DEBUGCTLMR register for the P6 family processors. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records the source and target addresses (in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs) for the last branch and the last exception or interrupt taken by the processor prior to a debug exception being generated. The processor clears this flag whenever a debug exception, such as an instruction or data breakpoint or single-step trap occurs.

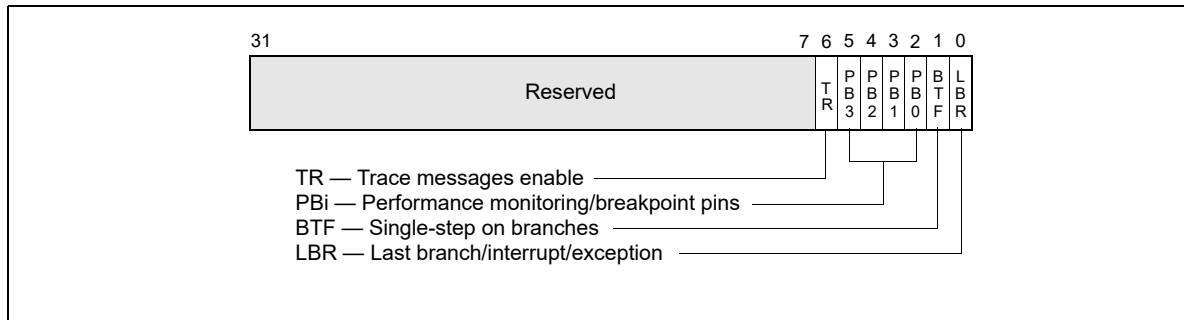


Figure 18-18. DEBUGCTLMR Register (P6 Family Processors)

- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag. See Section 18.4.3, “Single-Stepping on Branches.”
- **PB_i (performance monitoring/breakpoint pins) flags (bits 2 through 5)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding PB_i# pin when a breakpoint match occurs. When a PB_i flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
- **TR (trace message enable) flag (bit 6)** — When set, trace messages are enabled as described in Section 18.4.4, “Branch Trace Messages.” Setting this flag greatly reduces the performance of the processor. When trace messages are enabled, the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are undefined.

18.16.2 Last Branch and Last Exception MSRs

The LastBranchToIP and LastBranchFromIP MSRs are 32-bit registers for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated. When a branch occurs, the processor loads the address of the branch instruction into the LastBranchFromIP MSR and loads the target address for the branch into the LastBranchToIP MSR.

When an interrupt or exception occurs (other than a debug exception), the address of the instruction that was interrupted by the exception or interrupt is loaded into the LastBranchFromIP MSR and the address of the exception or interrupt handler that is called is loaded into the LastBranchToIP MSR.

The LastExceptionToIP and LastExceptionFromIP MSRs (also 32-bit registers) record the instruction pointers for the last branch that the processor took prior to an exception or interrupt being generated. When an exception or interrupt occurs, the contents of the LastBranchToIP and LastBranchFromIP MSRs are copied into these registers before the to and from addresses of the exception or interrupt are recorded in the LastBranchToIP and LastBranchFromIP MSRs.

These registers can be read using the RDMSR instruction.

Note that the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into the current code segment, as opposed to linear addresses, which are saved in last branch records for the Pentium 4 and Intel Xeon processors.

18.16.3 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag in the DEBUGCTLMR register is set, the processor automatically begins recording branches that it takes, exceptions that are generated (except for debug exceptions), and interrupts that are serviced. Each time a branch, exception, or interrupt occurs, the processor records the to and from instruction pointers in the LastBranchToIP and LastBranchFromIP MSRs. In addition, for interrupts and exceptions, the processor copies the contents of the LastBranchToIP and LastBranchFromIP MSRs into the LastExceptionToIP and LastExceptionFromIP MSRs prior to recording the to and from addresses of the interrupt or exception.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler, but does not touch the last branch and last exception MSRs. The addresses for the last branch, interrupt, or exception taken are thus retained in the LastBranchToIP and LastBranchFromIP MSRs and the addresses of the last branch prior to an interrupt or exception are retained in the LastExceptionToIP, and LastExceptionFromIP MSRs.

The debugger can use the last branch, interrupt, and/or exception addresses in combination with code-segment selectors retrieved from the stack to reset breakpoints in the breakpoint-address registers (DR0 through DR3), allowing a backward trace from the manifestation of a particular bug toward its source. Because the instruction pointers recorded in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into a code segment, software must determine the segment base address of the code segment associated with the control transfer to calculate the linear address to be placed in the breakpoint-address registers. The segment base address can be determined by reading the segment selector for the code segment from the stack and using it to locate the segment descriptor for the segment in the GDT or LDT. The segment base address can then be read from the segment descriptor.

Before resuming program execution from a debug-exception handler, the handler must set the LBR flag again to re-enable last branch and last exception/interrupt recording.

18.17 TIME-STAMP COUNTER

The Intel 64 and IA-32 architectures (beginning with the Pentium processor) define a time-stamp counter mechanism that can be used to monitor and identify the relative time occurrence of processor events. The counter's architecture includes the following components:

- **TSC flag** — A feature bit that indicates the availability of the time-stamp counter. The counter is available in an if the function CPUID.1:EDX.TSC[bit 4] = 1.
- **IA32_TIME_STAMP_COUNTER MSR** (called TSC MSR in P6 family and Pentium processors) — The MSR used as the counter.
- **RDTSC instruction** — An instruction used to read the time-stamp counter.
- **TSD flag** — A control register flag is used to enable or disable the time-stamp counter (enabled if CR4.TSD[bit 2] = 1).

The time-stamp counter (as implemented in the P6 family, Pentium, Pentium M, Pentium 4, Intel Xeon, Intel Core Solo and Intel Core Duo processors and later processors) is a 64-bit counter that is set to 0 following a RESET of the processor. Following a RESET, the counter increments even when the processor is halted by the HLT instruction or the external STPCLK# pin. Note that the assertion of the external DPSLP# pin may cause the time-stamp counter to stop.

Processor families increment the time-stamp counter differently:

- For Pentium M processors (family [06H], models [09H, 0DH]); for Pentium 4 processors, Intel Xeon processors (family [0FH], models [00H, 01H, or 02H]); and for P6 family processors: the time-stamp counter increments with every internal processor clock cycle.

The internal processor clock cycle is determined by the current core-clock to bus-clock ratio. Intel® SpeedStep® technology transitions may also impact the processor clock.

- For Pentium 4 processors, Intel Xeon processors (family [0FH], models [03H and higher]); for Intel Core Solo and Intel Core Duo processors (family [06H], model [0EH]); for the Intel Xeon processor 5100 series and Intel Core 2 Duo processors (family [06H], model [0FH]); for Intel Core 2 and Intel Xeon processors (family [06H], DisplayModel [17H]); for Intel Atom processors (family [06H], DisplayModel [1CH]): the time-stamp counter increments at a constant rate. That rate may be set by the maximum core-clock to bus-clock ratio of the

processor or may be set by the maximum resolved frequency at which the processor is booted. The maximum resolved frequency may differ from the processor base frequency, see Section 20.7.2 for more detail. On certain processors, the TSC frequency may not be the same as the frequency in the brand string.

The specific processor configuration determines the behavior. Constant TSC behavior ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. This is the architectural behavior moving forward.

NOTE

To determine average processor clock frequency, Intel recommends the use of performance monitoring logic to count processor core clocks over the period of time for which the average is required. See Section 20.6.4.5, “Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture,” and <https://perfmon-events.intel.com/> for more information.

The RDTSC instruction reads the time-stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for a 64-bit counter wraparound. Intel guarantees that the time-stamp counter will not wraparound within 10 years after being reset. The period for counter wrap is longer for Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Normally, the RDTSC instruction can be executed by programs and procedures running at any privilege level and in virtual-8086 mode. The TSD flag allows use of this instruction to be restricted to programs and procedures running at privilege level 0. A secure operating system would set the TSD flag during system initialization to disable user access to the time-stamp counter. An operating system that disables user access to the time-stamp counter should emulate the instruction through a user-accessible programming interface.

The RDTSC instruction is not serializing or ordered with other instructions. It does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDTSC instruction operation is performed.

The RDMSR and WRMSR instructions read and write the time-stamp counter, treating the time-stamp counter as an ordinary MSR (address 10H). In the Pentium 4, Intel Xeon, and P6 family processors, all 64-bits of the time-stamp counter are read using RDMSR (just as with RDTSC). When WRMSR is used to write the time-stamp counter on processors before family [0FH], models [03H, 04H]: only the low-order 32-bits of the time-stamp counter can be written (the high-order 32 bits are cleared to 0). For family [0FH], models [03H, 04H, 06H]; for family [06H]], model [0EH, 0FH]; for family [06H]], DisplayModel [17H, 1AH, 1CH, 1DH]: all 64 bits are writable.

18.17.1 Invariant TSC

The time stamp counter in newer processors may support an enhancement, referred to as invariant TSC. Processor’s support for invariant TSC is indicated by CPUID.80000007H:EDX[8].

The invariant TSC will run at a constant rate in all ACPI P-, C-, and T-states. This is the architectural behavior moving forward. On processors with invariant TSC support, the OS may use the TSC for wall clock timer services (instead of ACPI or HPET timers). TSC reads are much more efficient and do not incur the overhead associated with a ring transition or access to a platform resource.

18.17.2 IA32_TSC_AUX Register and RDTSCP Support

Processors based on Nehalem microarchitecture provide an auxiliary TSC register, IA32_TSC_AUX that is designed to be used in conjunction with IA32_TSC. IA32_TSC_AUX provides a 32-bit field that is initialized by privileged software with a signature value (for example, a logical processor ID).

The primary usage of IA32_TSC_AUX in conjunction with IA32_TSC is to allow software to read the 64-bit time stamp in IA32_TSC and signature value in IA32_TSC_AUX with the instruction RDTSCP in an atomic operation. RDTSCP returns the 64-bit time stamp in EDX:EAX and the 32-bit TSC_AUX signature value in ECX. The atomicity of RDTSCP ensures that no context switch can occur between the reads of the TSC and TSC_AUX values.

Support for RDTSCP is indicated by CPUID.80000011H:EDX[27]. As with RDTSC instruction, non-ring 0 access is controlled by CR4.TSD (Time Stamp Disable flag).

User mode software can use RDTSCP to detect if CPU migration has occurred between successive reads of the TSC. It can also be used to adjust for per-CPU differences in TSC values in a NUMA system.

18.17.3 Time-Stamp Counter Adjustment

Software can modify the value of the time-stamp counter (TSC) of a logical processor by using the WRMSR instruction to write to the IA32_TIME_STAMP_COUNTER MSR (address 10H). Because such a write applies only to that logical processor, software seeking to synchronize the TSC values of multiple logical processors must perform these writes on each logical processor. It may be difficult for software to do this in a way that ensures that all logical processors will have the same value for the TSC at a given point in time.

The synchronization of TSC adjustment can be simplified by using the 64-bit IA32_TSC_ADJUST MSR (address 3BH). Like the IA32_TIME_STAMP_COUNTER MSR, the IA32_TSC_ADJUST MSR is maintained separately for each logical processor. A logical processor maintains and uses the IA32_TSC_ADJUST MSR as follows:

- On RESET, the value of the IA32_TSC_ADJUST MSR is 0.
- If an execution of WRMSR to the IA32_TIME_STAMP_COUNTER MSR adds (or subtracts) value X from the TSC, the logical processor also adds (or subtracts) value X from the IA32_TSC_ADJUST MSR.
- If an execution of WRMSR to the IA32_TSC_ADJUST MSR adds (or subtracts) value X from that MSR, the logical processor also adds (or subtracts) value X from the TSC.

Unlike the TSC, the value of the IA32_TSC_ADJUST MSR changes only in response to WRMSR (either to the MSR itself, or to the IA32_TIME_STAMP_COUNTER MSR). Its value does not otherwise change as time elapses. Software seeking to adjust the TSC can do so by using WRMSR to write the same value to the IA32_TSC_ADJUST MSR on each logical processor.

Processor support for the IA32_TSC_ADJUST MSR is indicated by CPUID.(EAX=07H, ECX=0H):EBX.TSC_ADJUST (bit 1).

18.17.4 Invariant Time-Keeping

The invariant TSC is based on the invariant timekeeping hardware (called Always Running Timer or ART), that runs at the core crystal clock frequency. The ratio defined by CPUID leaf 15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX[InvariantTSC] = 1, the following linearity relationship holds between TSC and the ART hardware:

$$\text{TSC_Value} = (\text{ART_Value} * \text{CPUID.15H:EBX[31:0]}) / \text{CPUID.15H:EAX[31:0]} + K$$

Where 'K' is an offset that can be adjusted by a privileged agent¹.

When ART hardware is reset, both invariant TSC and K are also reset.

18.18 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) MONITORING FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of monitoring capabilities including Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring (MBM). The Intel® Xeon® processor E5 v3 family introduced resource monitoring capability in each logical processor to measure specific platform shared resource metrics, for example, L3 cache occupancy. The programming interface for these monitoring features is described in this section. Two features within the monitoring feature set provided are described - Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.

1. IA32_TSC_ADJUST MSR and the TSC-offset field in the VM execution controls of VMCS are some of the common interfaces that privileged software can use to manage the time stamp counter for keeping time

Cache Monitoring Technology (CMT) allows an Operating System, Hypervisor or similar system management agent to determine the usage of cache by applications running on the platform. The initial implementation is directed at L3 cache monitoring (currently the last level cache in most server platforms).

Memory Bandwidth Monitoring (MBM), introduced in the Intel® Xeon® processor E5 v4 family, builds on the CMT infrastructure to allow monitoring of bandwidth from one level of the cache hierarchy to the next - in this case focusing on the L3 cache, which is typically backed directly by system memory. As a result of this implementation, memory bandwidth can be monitored.

The monitoring mechanisms described provide the following key shared infrastructure features:

- A mechanism to enumerate the presence of the monitoring capabilities within the platform (via a CPUID feature bit).
- A framework to enumerate the details of each sub-feature (including CMT and MBM, as discussed later, via CPUID leaves and sub-leaves).
- A mechanism for the OS or Hypervisor to indicate a software-defined ID for each of the software threads (applications, virtual machines, etc.) that are scheduled to run on a logical processor. These identifiers are known as Resource Monitoring IDs (RMIDs).
- Mechanisms in hardware to monitor cache occupancy and bandwidth statistics as applicable to a given product generation on a per software-id basis.
- Mechanisms for the OS or Hypervisor to read back the collected metrics such as L3 occupancy or Memory Bandwidth for a given software ID at any point during runtime.

18.18.1 Overview of Cache Monitoring Technology and Memory Bandwidth Monitoring

The shared resource monitoring features described in this chapter provide a layer of abstraction between applications and logical processors through the use of **Resource Monitoring IDs** (RMIDs). Each logical processor in the system can be assigned an RMID independently, or multiple logical processors can be assigned to the same RMID value (e.g., to track an application with multiple threads). For each logical processor, only one RMID value is active at a time. This is enforced by the IA32_PQR_ASSOC MSR, which specifies the active RMID of a logical processor. Writing to this MSR by software changes the active RMID of the logical processor from an old value to a new value.

The underlying platform shared resource monitoring hardware tracks cache metrics such as cache utilization and misses as a result of memory accesses according to the RMIDs and reports monitored data via a counter register (IA32_QM_CTR). The specific event types supported vary by generation and can be enumerated via CPUID. Before reading back monitored data software must configure an event selection MSR (IA32_QM_EVTSEL) to specify which metric is to be reported, and the specific RMID for which the data should be returned.

Processor support of the monitoring framework and sub-features such as CMT is reported via the CPUID instruction. The resource type available to the monitoring framework is enumerated via a new leaf function in CPUID. Reading and writing to the monitoring MSRs requires the RDMSR and WRMSR instructions.

The Cache Monitoring Technology feature set provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the CMT feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- CMT-specific event codes to read occupancy for a given level of the cache hierarchy.

The Memory Bandwidth Monitoring feature provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the MBM feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- MBM-specific event codes to read bandwidth out to the next level of the hierarchy and various sub-event codes to read more specific metrics as discussed later (e.g., total bandwidth vs. bandwidth only from local memory controllers on the same package).

18.18.2 Enabling Monitoring: Usage Flow

Figure 18-19 illustrates the key steps for OS/VMM to detect support of shared resource monitoring features such as CMT and enable resource monitoring for available resource types and monitoring events.

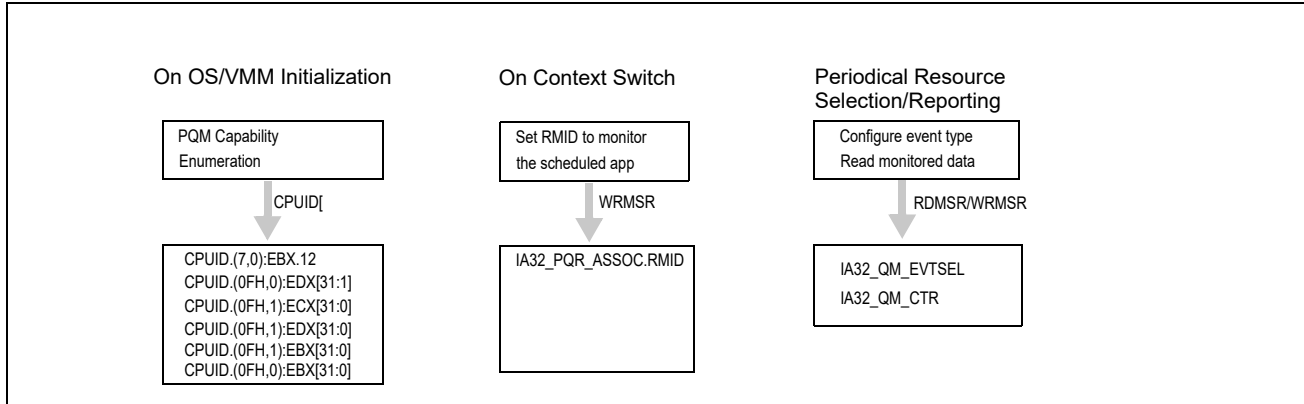


Figure 18-19. Platform Shared Resource Monitoring Usage Flow

18.18.3 Enumeration and Detecting Support of Cache Monitoring Technology and Memory Bandwidth Monitoring

Software can query processor support of shared resource monitoring features capabilities by executing CUID instruction with EAX = 07H, ECX = 0H as input. If CUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] reports 1, the processor provides the following programming interfaces for shared resource monitoring, including Cache Monitoring Technology:

- CUID leaf function 0FH (Shared Resource Monitoring Enumeration leaf) provides information on available resource types (see Section 18.18.4), and monitoring capabilities for each resource type (see Section 18.18.5). Note CMT and MBM capabilities are enumerated as separate event vectors using shared enumeration infrastructure under a given resource type.
- IA32_PQR_ASSOC.RMID: The per-logical-processor MSR, IA32_PQR_ASSOC, that OS/VMM can use to assign an RMID to each logical processor, see Section 18.18.6.
- IA32_QM_EVTSEL: This MSR specifies an Event ID (EvtID) and an RMID which the platform uses to look up and provide monitoring data in the monitoring counter, IA32_QM_CTR, see Section 18.18.7.
- IA32_QM_CTR: This MSR reports monitored resource data when available along with bits to allow software to check for error conditions and verify data validity.

Software must follow the following sequence of enumeration to discover Cache Monitoring Technology capabilities:

1. Execute CUID with EAX=0 to discover the "cpuid_maxLeaf" supported in the processor;
2. If cpuid_maxLeaf >= 7, then execute CUID with EAX=7, ECX= 0 to verify CUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] is set;
3. If CUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] = 1, then execute CUID with EAX=0FH, ECX= 0 to query available resource types that support monitoring;
4. If CUID.(EAX=0FH, ECX=0):EDX.L3[bit 1] = 1, then execute CUID with EAX=0FH, ECX= 1 to query the specific capabilities of L3 Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.
5. If CUID.(EAX=0FH, ECX=0):EDX reports additional resource types supporting monitoring, then execute CUID with EAX=0FH, ECX set to a corresponding resource type ID (ResID) as enumerated by the bit position of CUID.(EAX=0FH, ECX=0):EDX.

18.18.4 Monitoring Resource Type and Capability Enumeration

CUID leaf function 0FH (Shared Resource Monitoring Enumeration leaf) provides one sub-leaf (sub-function 0) that reports shared enumeration infrastructure, and one or more sub-functions that report feature-specific enumeration data:

- Monitoring leaf sub-function 0 enumerates available resources that support monitoring, i.e., executing CUID with EAX=0FH and ECX=0H. In the initial implementation, L3 cache is the only resource type available. Each

supported resource type is represented by a bit in CPUID.(EAX=0FH, ECX=0):EDX[31:1]. The bit position corresponds to the sub-leaf index (ResID) that software must use to query details of the monitoring capability of that resource type (see Figure 18-21 and Figure 18-22). Reserved bits of CPUID.(EAX=0FH, ECX=0):EDX[31:2] correspond to unsupported sub-leaves of the CPUID.0FH leaf. Additionally, CPUID.(EAX=0FH, ECX=0H):EBX reports the highest RMID value of any resource type that supports monitoring in the processor.

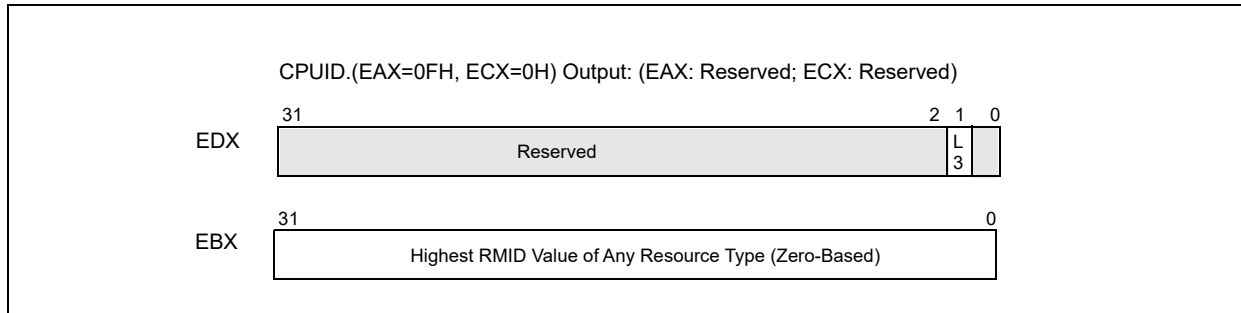


Figure 18-20. CPUID.(EAX=0FH, ECX=0H) Monitoring Resource Type Enumeration

18.18.5 Feature-Specific Enumeration

Each additional sub-leaf of CPUID.(EAX=0FH, ECX=ResID) enumerates the specific details for software to program Monitoring MSR using the resource type associated with the given ResID.

Note that in future Monitoring implementations the meanings of the returned registers may vary in other sub-leaves that are not yet defined. The registers will be specified and defined on a per-ResID basis.

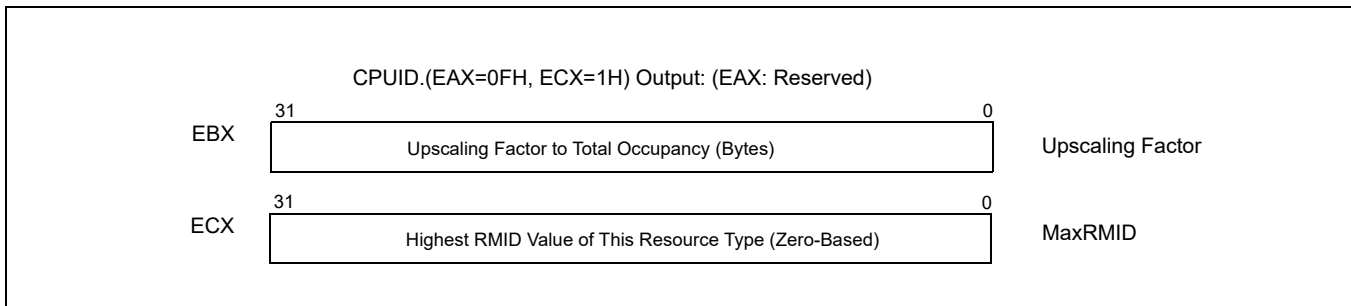


Figure 18-21. L3 Cache Monitoring Capability Enumeration Data (CPUID.(EAX=0FH, ECX=1H))

For each supported Cache Monitoring resource type, hardware supports only a finite number of RMIDs. CPUID.(EAX=0FH, ECX=1H).ECX enumerates the highest RMID value that can be monitored with this resource type, see Figure 18-21.

CPUID.(EAX=0FH, ECX=1H).EDX specifies a bit vector that is used to look up the EventID (See Figure 18-22 and Table 18-18) that software must program with IA32_QM_EVTSEL in order to retrieve event data. After software configures IA32_QMEVTSEL with the desired RMID and EventID, it can read the resulting data from IA32_QM_CTR. The raw numerical value reported from IA32_QM_CTR can be converted to the final value (occupancy in bytes or bandwidth in bytes per sampled time period) by multiplying the counter value by the value from CPUID.(EAX=0FH, ECX=1H).EBX, see Figure 18-21.

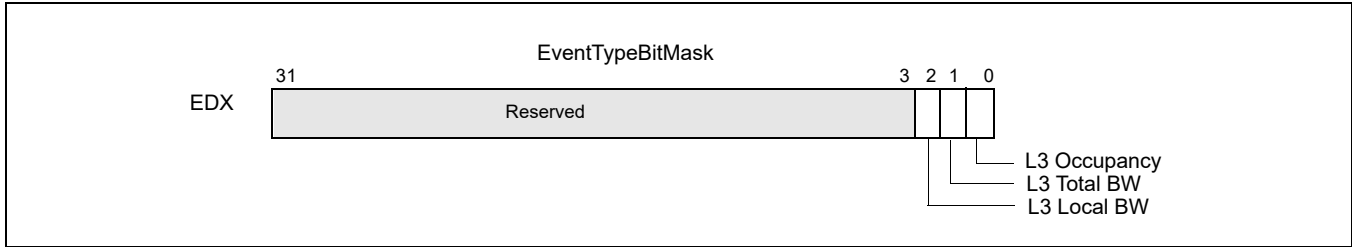


Figure 18-22. L3 Cache Monitoring Capability Enumeration Event Type Bit Vector (CPUID.(EAX=0FH, ECX=1H))

18.18.5.1 Cache Monitoring Technology

On processors for which Cache Monitoring Technology supports the L3 cache occupancy event, CPUID.(EAX=0FH, ECX=1H).EDX would return with only bit 0 set. The corresponding event ID can be looked up from Table 18-18. The L3 occupancy data accumulated in IA32_QM_CTR can be converted to total occupancy (in bytes) by multiplying with CPUID.(EAX=0FH, ECX=1H).EBX.

Event codes for Cache Monitoring Technology are discussed in the next section.

18.18.5.2 Memory Bandwidth Monitoring

On processors that monitoring supports Memory Bandwidth Monitoring using ResID=1 (L3), two additional bits will be set in the vector at CPUID.(EAX=0FH, ECX=1H).EDX:

- CPUID.(EAX=0FH, ECX=1H).EDX[bit 1]: indicates the L3 total external bandwidth monitoring event is supported if set. This event monitors the L3 total external bandwidth to the next level of the cache hierarchy, including all demand and prefetch misses from the L3 to the next hierarchy of the memory system. In most platforms, this represents memory bandwidth.
- CPUID.(EAX=0FH, ECX=1H).EDX[bit 2]: indicates L3 local memory bandwidth monitoring event is supported if set. This event monitors the L3 external bandwidth satisfied by the local memory. In most platforms that support this event, L3 requests are likely serviced by a memory system with non-uniform memory architecture. This allows bandwidth to off-package memory resources to be tracked by subtracting local from total bandwidth (for instance, bandwidth over QPI to a memory controller on another physical processor could be tracked by subtraction). Note that it is not possible to read the local and total bandwidth atomically; multiple operations are needed. Because of this, it is possible for the counters to change in between the two reads.

The corresponding Event ID can be looked up from Table 18-18. The L3 bandwidth data accumulated in IA32_QM_CTR can be converted to total bandwidth (in bytes) using CPUID.(EAX=0FH, ECX=1H).EBX.

Table 18-18. Monitoring Supported Event IDs

Event Type	Event ID	Context
L3 Cache Occupancy	01H	Cache Monitoring Technology
L3 Total External Bandwidth	02H	MBM
L3 Local External Bandwidth	03H	MBM
Reserved	All other event codes	N/A

18.18.6 Monitoring Resource RMID Association

After Monitoring and sub-features has been enumerated, software can begin using the monitoring features. The first step is to associate a given software thread (or multiple threads as part of an application, VM, group of applications or other abstraction) with an RMID.

Note that the process of associating an RMID with a given software thread is the same for all shared resource monitoring features (CMT, MBM), and a given RMID number has the same meaning from the viewpoint of any logical processors in a package. Stated another way, a thread may be associated in a 1:1 mapping with an RMID, and that

RMID may allow cache occupancy, memory bandwidth information or other monitoring data to be read back later with monitoring event codes (retrieving data is discussed in a previous section).

The association of an application thread with an RMID requires an OS to program the per-logical-processor MSR IA32_PQR_ASSOC at context swap time (updates may also be made at any other arbitrary points during program execution such as application phase changes). The IA32_PQR_ASSOC MSR specifies the active RMID that monitoring hardware will use to tag internal operations, such as L3 cache requests. The layout of the MSR is shown in Figure 18-23. Software specifies the active RMID to monitor in the IA32_PQR_ASSOC.RMID field. The width of the RMID field can vary from one implementation to another, and is derived from Ceil ($\log_2 (1 + \text{CPUID}.\text{EAX}=0\text{FH}, \text{ECX}=0):\text{EBX}[31:0])$). The value of IA32_PQR_ASSOC after power-on is 0.

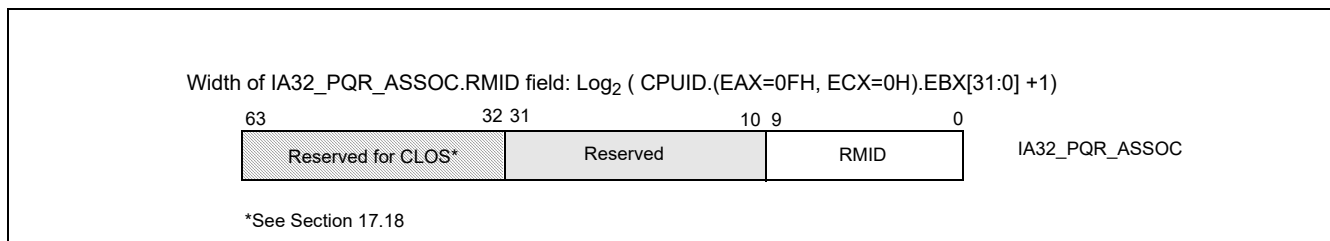


Figure 18-23. IA32_PQR_ASSOC MSR

In the initial implementation, the width of the RMID field is up to 10 bits wide, zero-referenced and fully encoded. However, software must use CPUID to query the maximum RMID supported by the processor. If a value larger than the maximum RMID is written to IA32_PQR_ASSOC.RMID, a #GP(0) fault will be generated.

RMIDs have a global scope within the physical package- if an RMID is assigned to one logical processor then the same RMID can be used to read multiple thread attributes later (for example, L3 cache occupancy or external bandwidth from the L3 to the next level of the cache hierarchy). In a multiple LLC platform the RMIDs are to be reassigned by the OS or VMM scheduler when an application is migrated across LLCs.

Note that in a situation where Monitoring supports multiple resource types, some upper range of RMIDs (e.g., RMID 31) may only be supported by one resource type but not by another resource type.

18.18.7 Monitoring Resource Selection and Reporting Infrastructure

The reporting mechanism for Cache Monitoring Technology and other related features is architecturally exposed as an MSR pair that can be programmed and read to measure various metrics such as the L3 cache occupancy (CMT) and bandwidths (MBM) depending on the level of Monitoring support provided by the platform. Data is reported back on a per-RMID basis. These events do not trigger based on event counts or trigger APIC interrupts (e.g., no Performance Monitoring Interrupt occurs based on counts). Rather, they are used to sample counts explicitly.

The MSR pair for the shared resource monitoring features (CMT, MBM) is separate from and not shared with architectural Perfmon counters, meaning software can use these monitoring features simultaneously with the Perfmon counters.

Access to the aggregated monitoring information is accomplished through the following programmable monitoring MSRs:

- IA32_QM_EVTSEL: This MSR provides a role similar to the event select MSRs for programmable performance monitoring described in Chapter 18. The simplified layout of the MSR is shown in Figure 18-24. Bits IA32_QM_EVTSEL.EvtID (bits 7:0) specify an event code of a supported resource type for hardware to report monitored data associated with IA32_QM_EVTSEL.RMID (bits 41:32). Software can configure IA32_QM_EVTSEL.RMID with any RMID that is active within the physical processor. The width of IA32_QM_EVTSEL.RMID matches that of IA32_PQR_ASSOC.RMID. Supported event codes for the IA32_QM_EVTSEL register are shown in Table 18-18. Note that valid event codes may not necessarily map directly to the bit position used to enumerate support for the resource via CPUID.

Software can program an RMID / Event ID pair into the IA32_QM_EVTSEL MSR bit field to select an RMID to read a particular counter for a given resource. The currently supported list of Monitoring Event IDs is discussed in Section 18.18.5, which covers feature-specific details.

Thread access to the IA32_QM_EVTSEL and IA32_QM_CTR MSR pair should be serialized (that is, treated as a critical section under lock) to avoid situations where one thread changes the RMID/EvtID just before another thread reads monitoring data from IA32_QM_CTR.

- IA32_QM_CTR: This MSR reports monitored data when available. It contains three bit fields. If software configures an unsupported RMID or event type in IA32_QM_EVTSEL, then IA32_QM_CTR.Error (bit 63) will be set, indicating there is no valid data to report. If IA32_QM_CTR.Unavailable (bit 62) is set, it indicates monitored data for the RMID is not available, and IA32_QM_CTR.data (bits 61:0) should be ignored. Therefore, IA32_QM_CTR.data (bits 61:0) is valid only if bit 63 and 62 are both clear. For Cache Monitoring Technology, software can convert IA32_QM_CTR.data into cache occupancy or bandwidth metrics expressed in bytes by multiplying with the conversion factor from CPUID.(EAX=0FH, ECX=1H).EBX.

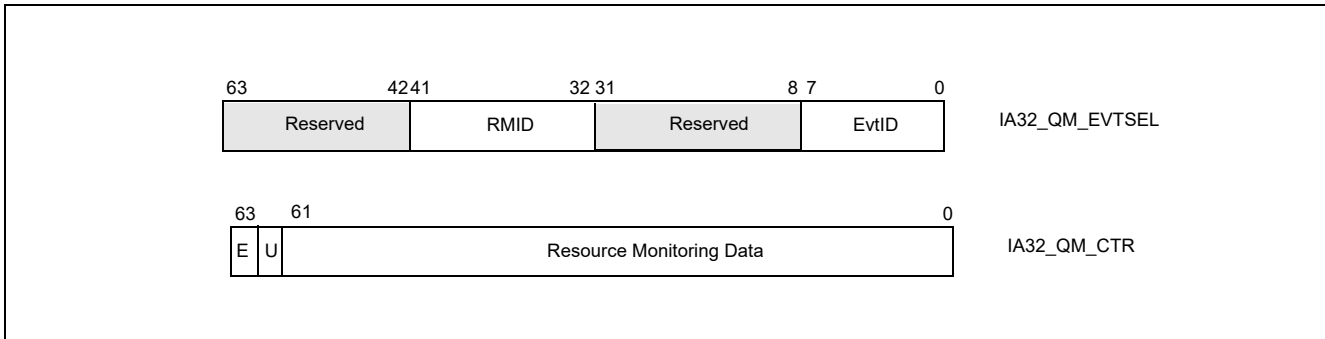


Figure 18-24. IA32_QM_EVTSEL and IA32_QM_CTR MSRs

18.18.8 Monitoring Programming Considerations

Figure 18-23 illustrates how system software can program IA32_QOSEVTSEL and IA32_QM_CTR to perform resource monitoring.

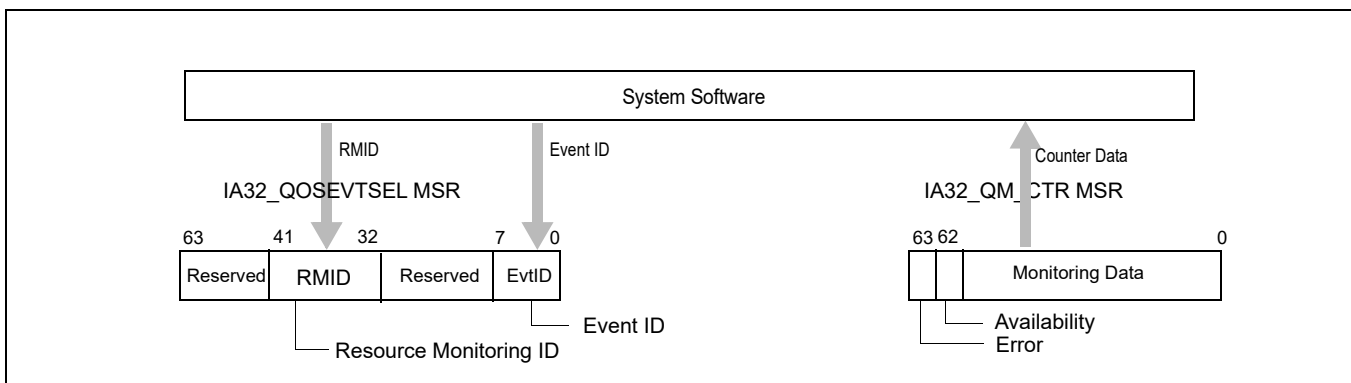


Figure 18-25. Software Usage of Cache Monitoring Resources

Though the field provided in IA32_QM_CTR allows for up to 62 bits of data to be returned, often a subset of bits are used. With Cache Monitoring Technology for instance, the number of bits used will be proportional to the base-two logarithm of the total cache size divided by the Upscaling Factor from CPUID.

In Memory Bandwidth Monitoring the initial counter size is 24 bits, and retrieving the value at 1Hz or faster is sufficient to ensure at most one rollover per sampling period. Any future changes to counter width will be enumerated to software.

18.18.8.1 Monitoring Dynamic Configuration

Both the IA32_QM_EVTSEL and IA32_PQR_ASSOC registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,
- An RMID exceeding the maxRMID is used.

18.18.8.2 Monitoring Operation With Power Saving Features

Note that some advanced power management features such as deep package C-states may shrink the L3 cache and cause CMT occupancy count to be reduced. MBM bandwidth counts may increase due to flushing cached data out of L3.

18.18.8.3 Monitoring Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and monitoring counter are unmodified across an SMI delivery. Thus, the execution of SMM handler code and SMM handler's data can manifest as spurious contribution in the monitored data.

It is possible for an SMM handler to minimize the impact on of spurious contribution in the QOS monitoring counters by reserving a dedicated RMID for monitoring the SMM handler. Such an SMM handler can save the previously configured QOS Monitoring state immediately upon entering SMM, and restoring the QOS monitoring state back to the prev-SMM RMID upon exit.

18.18.8.4 Monitoring Operation with RAS Features

In general, the Reliability, Availability, and Serviceability (RAS) features present in Intel Platforms are not expected to significantly affect shared resource monitoring counts. In cases where software RAS features cause memory copies or cache accesses, these may be tracked and may influence the shared resource monitoring counter values.

18.19 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) ALLOCATION FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of allocation (resource control) capabilities including Cache Allocation Technology (CAT) and Code and Data Prioritization (CDP). The Intel Xeon processor E5 v4 family (and a subset of communication-focused processors in the Intel Xeon E5 v3 family) introduce capabilities to configure and make use of the Cache Allocation Technology (CAT) mechanisms on the L3 cache. Certain Intel Atom processors also provide support for control over the L2 cache, with capabilities as described below. The programming interface for Cache Allocation Technology and for the more general allocation capabilities are described in the rest of this chapter. The CAT and CDP capabilities, where architecturally supported, may be detected and enumerated in software using the *CPUID* instruction, as described in this chapter.

The Intel Xeon Scalable Processor Family introduces the Memory Bandwidth Allocation (MBA) feature which provides indirect control over the memory bandwidth available to CPU cores, and is discussed later in this chapter.

18.19.1 Introduction to Cache Allocation Technology (CAT)

Cache Allocation Technology enables an Operating System (OS), Hypervisor /Virtual Machine Manager (VMM) or similar system service management agent to specify the amount of cache space into which an application can fill (as a hint to hardware - certain features such as power management may override CAT settings). Specialized user-level implementations with minimal OS support are also possible, though not necessarily recommended (see notes below for OS/Hypervisor with respect to ring 3 software and virtual guests). Depending on the processor family, L2 or L3 cache allocation capability may be provided, and the technology is designed to scale across multiple cache levels and technology generations.

Software can determine which levels are supported in a given platform programmatically using CPUID as described in the following sections.

The CAT mechanisms defined in this document provide the following key features:

- A mechanism to enumerate platform Cache Allocation Technology capabilities and available resource types that provides CAT control capabilities. For implementations that support Cache Allocation Technology, CPUID provides enumeration support to query which levels of the cache hierarchy are supported and specific CAT capabilities, such as the max allocation bitmask size,
- A mechanism for the OS or Hypervisor to configure the amount of a resource available to a particular Class of Service via a list of allocation bitmasks,
- Mechanisms for the OS or Hypervisor to signal the Class of Service to which an application belongs, and
- Hardware mechanisms to guide the LLC fill policy when an application has been designated to belong to a specific Class of Service.

Note that for many usages, an OS or Hypervisor may not want to expose Cache Allocation Technology mechanisms to Ring3 software or virtualized guests.

The Cache Allocation Technology feature enables more cache resources (i.e., cache space) to be made available for high priority applications based on guidance from the execution environment as shown in Figure 18-26. The architecture also allows dynamic resource reassignment during runtime to further optimize the performance of the high priority application with minimal degradation to the low priority app. Additionally, resources can be rebalanced for system throughput benefit across uses cases of OSES, VMMs, containers, and other scenarios by managing the CPUID and MSR interfaces. This section describes the hardware and software support required in the platform including what is required of the execution environment (i.e., OS/VMM) to support such resource control. Note that in Figure 18-26 the L3 Cache is shown as an example resource.

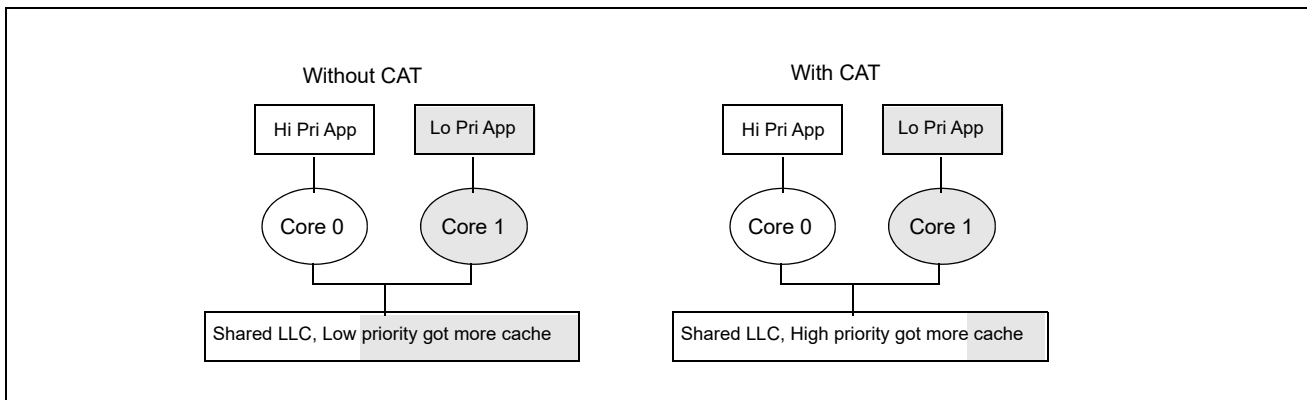


Figure 18-26. Cache Allocation Technology Enables Allocation of More Resources to High Priority Applications

18.19.2 Cache Allocation Technology Architecture

The fundamental goal of Cache Allocation Technology is to enable resource allocation based on application priority or Class of Service (COS or CLOS). The processor exposes a set of Classes of Service into which applications (or individual threads) can be assigned. Cache allocation for the respective applications or threads is then restricted based on the class with which they are associated. Each Class of Service can be configured using capacity bitmasks (CBMs) which represent capacity and indicate the degree of overlap and isolation between classes. For each logical processor there is a register exposed (referred to here as the IA32_PQR_ASSOC MSR or PQR) to allow the OS/VMM to specify a COS when an application, thread or VM is scheduled.

The usage of Classes of Service (COS) are consistent across resources and a COS may have multiple resource control attributes attached, which reduces software overhead at context swap time. Rather than adding new types of COS tags per resource for instance, the COS management overhead is constant. Cache allocation for the indicated application/thread/container/VM is then controlled automatically by the hardware based on the class and the bitmask associated with that class. Bitmasks are configured via the IA32_resourceType_MASK_n MSRs, where resourceType indicates a resource type (e.g., "L3" for the L3 cache) and "n" indicates a COS number.

The basic ingredients of Cache Allocation Technology are as follows:

- An architecturally exposed mechanism using CPUID to indicate whether CAT is supported, and what resource types are available which can be controlled,
- For each available resourceType, CPUID also enumerates the total number of Classes of Services and the length of the capacity bitmasks that can be used to enforce cache allocation to applications on the platform,
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to configure the behavior of different classes of service using the bitmasks available,
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to assign a COS to an executing software thread (i.e., associating the active CR3 of a logical processor with the COS in IA32_PQR_ASSOC),
- Implementation-dependent mechanisms to indicate which COS is associated with a memory access and to enforce the cache allocation on a per COS basis.

A capacity bitmask (CBM) provides a hint to the hardware indicating the cache space an application should be limited to as well as providing an indication of overlap and isolation in the CAT-capable cache from other applications contending for the cache. The bit length of the capacity mask available generally depends on the configuration of the cache and is specified in the enumeration process for CAT in CPUID (this may vary between models in a processor family as well). Similarly, other parameters such as the number of supported COS may vary for each resource type, and these details can be enumerated via CPUID.

	M7	M6	M5	M4	M3	M2	M1	M0	
COS0	A	A	A	A	A	A	A	A	Default Bitmask
COS1	A	A	A	A	A	A	A	A	
COS2	A	A	A	A	A	A	A	A	
COS3	A	A	A	A	A	A	A	A	
	M7	M6	M5	M4	M3	M2	M1	M0	
COS0	A	A	A	A	A	A	A	A	Overlapped Bitmask
COS1					A	A	A	A	
COS2							A	A	
COS3								A	
	M7	M6	M5	M4	M3	M2	M1	M0	
COS0	A	A	A	A					Isolated Bitmask
COS1					A	A			
COS2							A		
COS3								A	

Figure 18-27. Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bit length of 8 are shown in Figure 18-27. Please note that all (and only) contiguous '1' combinations are allowed (e.g., FFFFH, 0FF0H, 003CH, etc.). Attempts to program a value without contiguous '1's (including zero) will result in a general protection fault (#GP(0)). It is generally expected that in way-based implementations, one capacity mask bit corresponds to some number of ways in cache, but the specific mapping is implementation-dependent. In all cases, a mask bit set to '1' specifies that a particular Class of Service can allocate into the cache subset represented by that bit. A value of '0' in a mask bit specifies that a Class of

Service cannot allocate into the given cache subset. In general, allocating more cache to a given application is usually beneficial to its performance.

Figure 18-27 also shows three examples of sets of Cache Capacity Bitmasks. For simplicity these are represented as 8-bit vectors, though this may vary depending on the implementation and how the mask is mapped to the available cache capacity. The first example shows the default case where all 4 Classes of Service (the total number of COS are implementation-dependent) have full access to the cache. The second case shows an overlapped case, which would allow some lower-priority threads share cache space with the highest priority threads. The third case shows various non-overlapped partitioning schemes. As a matter of software policy for extensibility COS0 should typically be considered and configured as the highest priority COS, followed by COS1, and so on, though there is no hardware restriction enforcing this mapping. When the system boots all threads are initialized to COS0, which has full access to the cache by default.

Though the representation of the CBMs looks similar to a way-based mapping they are independent of any specific enforcement implementation (e.g., way partitioning.) Rather, this is a convenient manner to represent capacity, overlap, and isolation of cache space. For example, executing a POPCNT instruction (population count of set bits) on the capacity bitmask can provide the fraction of cache space that a class of service can allocate into. In addition to the fraction, the exact location of the bits also shows whether the class of service overlaps with other classes of service or is entirely isolated in terms of cache space used.

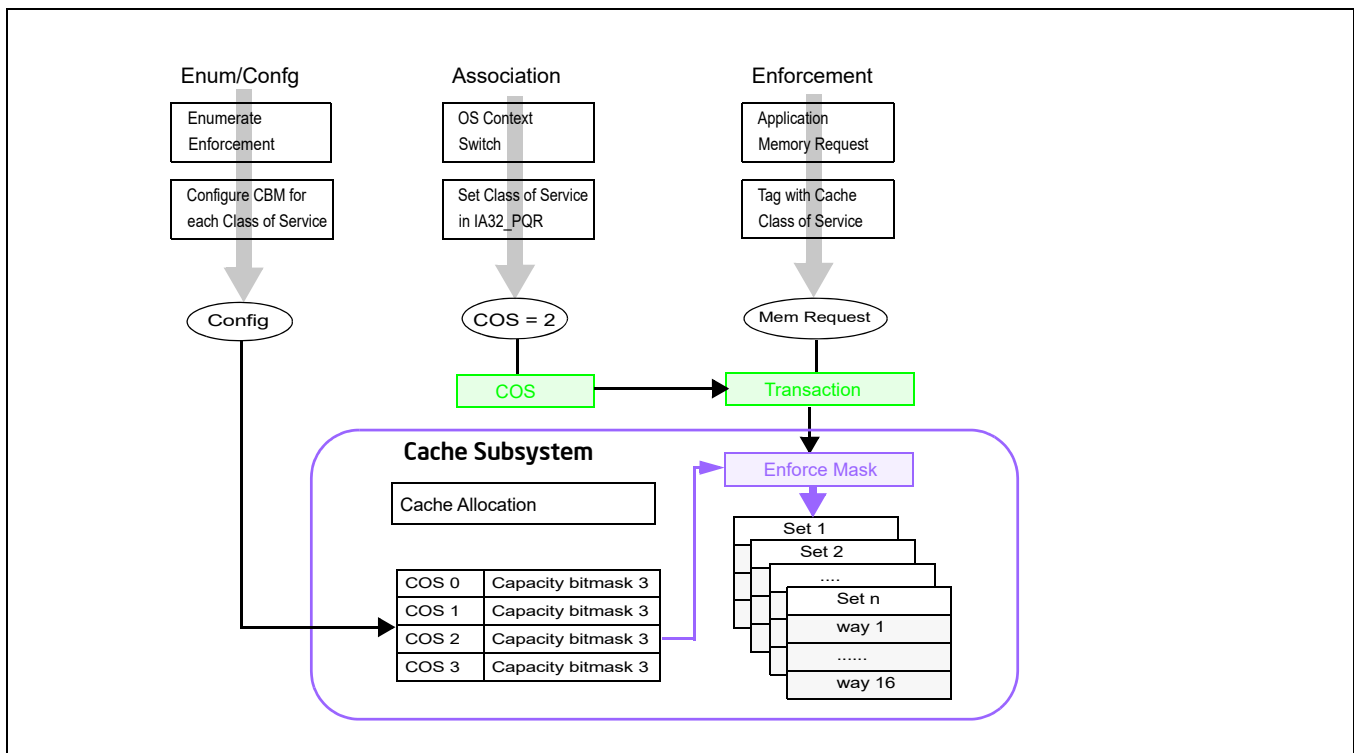


Figure 18-28. Class of Service and Cache Capacity Bitmasks

Figure 18-28 shows how the Cache Capacity Bitmasks and the per-logical-processor Class of Service are logically used to enable Cache Allocation Technology. All (and only) contiguous 1's in the CBM are permitted. The length of a CBM may vary from resource to resource or between processor generations and can be enumerated using CPUID. From the available mask set and based on the goals of the OS/VMM (shared or isolated cache, etc.) bitmasks are selected and associated with different classes of service. For the available Classes of Service the associated CBMs can be programmed via the global set of CAT configuration registers (in the case of L3 CAT, via the IA32_L3_MASK_n MSRs, where "n" is the Class of Service, starting from zero). In all architectural implementations supporting CPUID it is possible to change the CBMs dynamically, during program execution, unless stated otherwise by Intel.

The currently running application's Class of Service is communicated to the hardware through the per-logical-processor PQR MSR (IA32_PQR_ASSOC MSR). When the OS schedules an application thread on a logical processor,

the application thread is associated with a specific COS (i.e., the corresponding COS in the PQR) and all requests to the CAT-capable resource from that logical processor are tagged with that COS (in other words, the application thread is configured to belong to a specific COS). The cache subsystem uses this tagged request information to enforce QoS. The capacity bitmask may be mapped into a way bitmask (or a similar enforcement entity based on the implementation) at the cache before it is applied to the allocation policy. For example, the capacity bitmask can be an 8-bit mask and the enforcement may be accomplished using a 16-way bitmask for a cache enforcement implementation based on way partitioning.

The following sections describe extensions of CAT such as Code and Data Prioritization (CDP), followed by details on specific features such as L3 CAT, L3 CDP, L2 CAT, and L2 CDP. Depending on the specific processor a mix of features may be supported, and CPUID provides enumeration capabilities to enable software to dynamically detect the set of supported features.

18.19.3 Code and Data Prioritization (CDP) Technology

Code and Data Prioritization Technology is an extension of CAT. CDP enables isolation and separate prioritization of code and data fetches to the L2 or L3 cache in a software configurable manner, depending on hardware support, which can enable workload prioritization and tuning of cache capacity to the characteristics of the workload. CDP extends Cache Allocation Technology (CAT) by providing separate code and data masks per Class of Service (COS). Support for the L2 CDP feature and the L3 CDP features are separately enumerated (via CPUID) and separately controlled (via remapping the L2 CAT MSRs or L3 CAT MSRs respectively). Section 18.19.6.3 and Section 18.19.7 provide details on enumerating, controlling, and enabling L3 and L2 CDP respectively, while this section provides a general overview.

The L3 CDP feature was first introduced on the Intel Xeon E5 v4 family of server processors, as an extension to L3 CAT. The L2 CDP feature is first introduced on future Intel Atom family processors, as an extension to L2 CAT.

By default, CDP is disabled on the processor. If the CAT MSRs are used without enabling CDP, the processor operates in a traditional CAT-only mode. When CDP is enabled,

- the CAT mask MSRs are re-mapped into interleaved pairs of mask MSRs for data or code fetches (see Figure 18-29),
- the range of COS for CAT is re-indexed, with the lower-half of the COS range available for CDP.

Using the CDP feature, virtual isolation between code and data can be configured on the L2 or L3 cache if desired, similar to how some processor cache levels provide separate L1 data and L1 instruction caches.

Like the CAT feature, CDP may be dynamically configured by privileged software at any point during normal system operation, including dynamically enabling or disabling the feature provided that certain software configuration requirements are met (see Section 18.19.5).

An example of the operating mode of CDP is shown in Figure 18-29. Shown at the top are traditional CAT usage models where capacity masks map 1:1 with a COS number to enable control over the cache space which a given COS (and thus applications, threads or VMs) may occupy. Shown at the bottom are example mask configurations where CDP is enabled, and each COS number maps 1:2 to two masks, one for code and one for data. This enables code and data to be either overlapped or isolated to varying degrees either globally or on a per-COS basis, depending on application and system needs.

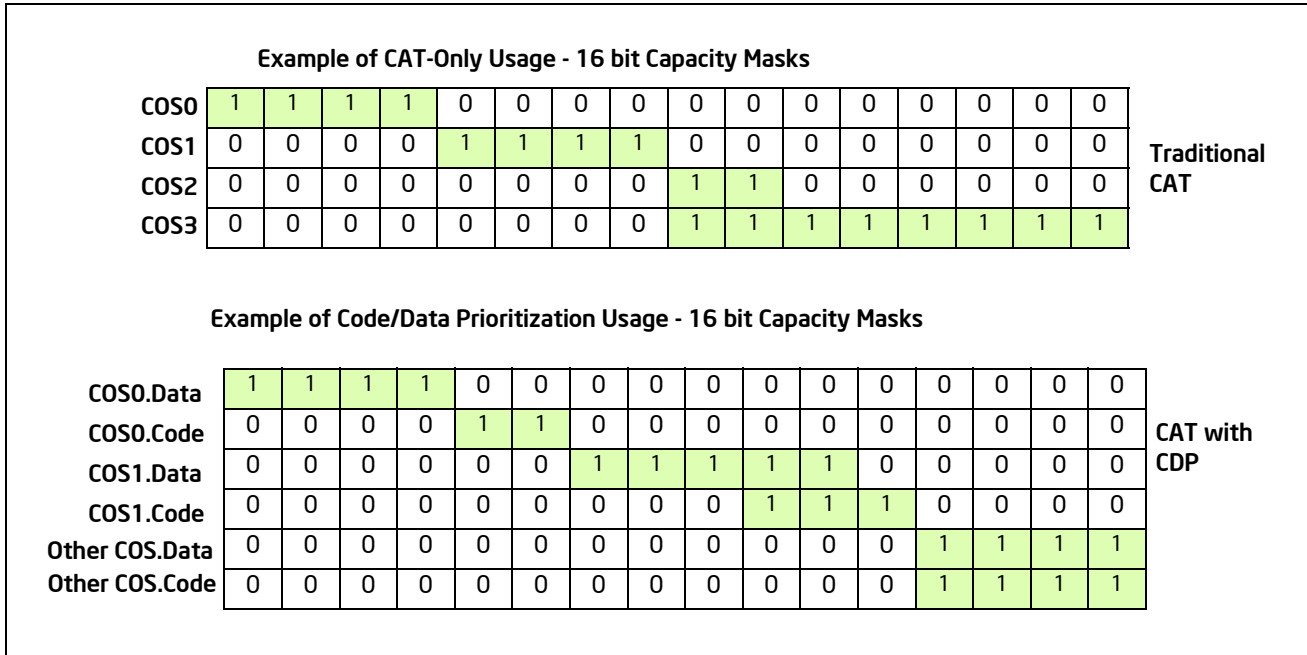


Figure 18-29. Code and Data Capacity Bitmasks of CDP

When CDP is enabled, the existing mask space for CAT-only operation is split. As an example if the system supports 16 CAT-only COS, when CDP is enabled the same MSR interfaces are used, however half of the masks correspond to code, half correspond to data, and the effective number of COS is reduced by half. Code/Data masks are defined per-COS and interleaved in the MSR space as described in subsequent sections.

In cases where CPUID exposes a non-even number of supported Classes of Service for the CAT or CDP features, software using CDP should use the lower matched pairs of code/data masks, and any upper unpaired masks should not be used. As an example, if CPUID exposes 5 CLOS, when CDP is enabled then two code/data pairs are available (masks 0/1 for CLOS[0] data/code and masks 2/3 for CLOS[1] data/code), however the upper un-paired mask should not be used (mask 4 in this case) or undefined behavior may result.

18.19.4 Enabling Cache Allocation Technology Usage Flow

Figure 18-30 illustrates the key steps for OS/VMM to detect support of Cache Allocation Technology and enable priority-based resource allocation for a CAT-capable resource.

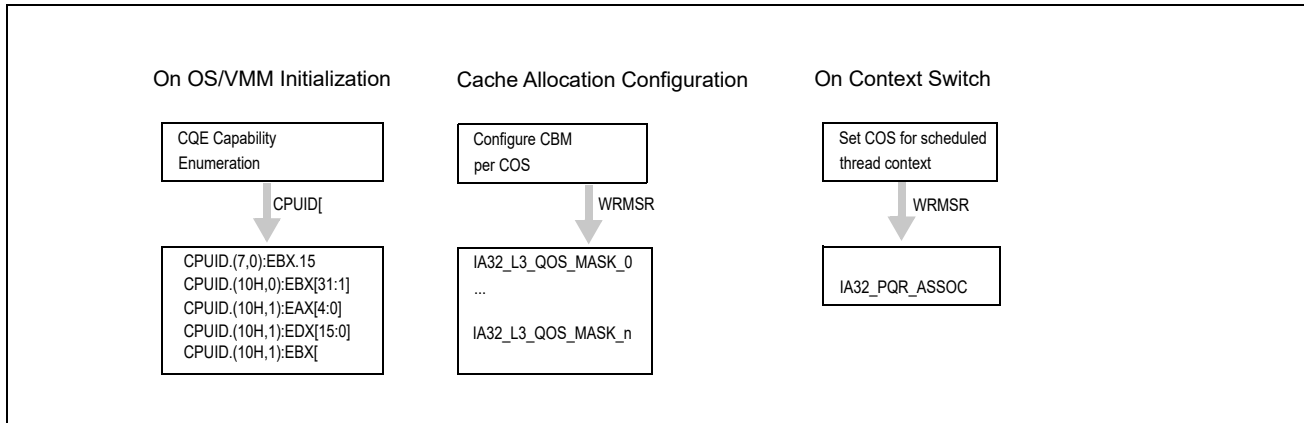


Figure 18-30. Cache Allocation Technology Usage Flow

Enumeration and configuration of L2 CAT is similar to L3 CAT, however CPUID details and MSR addresses differ. Common CLOS are used across the features.

18.19.4.1 Enumeration and Detection Support of Cache Allocation Technology

Software can query processor support of CAT capabilities by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.(EAX=07H, ECX=0):EBX.PQE[bit 15] reports 1, the processor supports software control over shared processor resources. Software must use CPUID leaf 10H to enumerate additional details of available resource types, classes of services and capability bitmasks. The programming interfaces provided by Cache Allocation Technology include:

- CPUID leaf function 10H (Cache Allocation Technology Enumeration leaf) and its sub-functions provide information on available resource types, and CAT capability for each resource type (see Section 18.19.4.2).
- IA32_L3_MASK_n: A range of MSRs is provided for each resource type, each MSR within that range specifying a software-configured capacity bitmask for each class of service. For L3 with Cache Allocation support, the CBM is specified using one of the IA32_L3_QOS_MASK_n MSR, where 'n' corresponds to a number within the supported range of COS, i.e., the range between 0 and CPUID.(EAX=10H, ECX=ResID):EDX[15:0], inclusive. See Section 18.19.4.3 for details.
- IA32_L2_MASK_n: A range of MSRs is provided for L2 Cache Allocation Technology, enabling software control over the amount of L2 cache available for each CLOS. Similar to L3 CAT, a CBM is specified for each CLOS using the set of registers, IA32_L2_QOS_MASK_n MSR, where 'n' ranges from zero to the maximum CLOS number reported for L2 CAT in CPUID. See Section 18.19.4.3 for details.

The L2 mask MSRs are scoped at the same level as the L2 cache (similarly, the L3 mask MSRs are scoped at the same level as the L3 cache). Software may determine which logical processors share an MSR (for instance local to a core, or shared across multiple cores) by performing a write to one of these MSRs and noting which logical threads observe the change. Example flows for a similar method to determine register scope are described in Section 16.5.2, "System Software Recommendation for Managing CMCI and Machine Check Resources". Software may also use CPUID leaf 4 to determine the maximum number of logical processor IDs that may share a given level of the cache.

- IA32_PQR_ASSOC.CLOS: The IA32_PQR_ASSOC MSR provides a COS field that OS/VMM can use to assign a logical processor to an available COS. The set of COS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software COS management overhead at context swap time. See Section 18.19.4.4 for details.

18.19.4.2 Cache Allocation Technology: Resource Type and Capability Enumeration

CPUID leaf function 10H (Cache Allocation Technology Enumeration leaf) provides two or more sub-functions:

- CAT Enumeration leaf sub-function 0 enumerates available resource types that support allocation control, i.e., by executing CPUID with EAX=10H and ECX=0H. Each supported resource type is represented by a bit field in

CPUID.(EAX=10H, ECX=0):EBX[31:1]. The bit position of each set bit corresponds to a Resource ID (ResID), for instance ResID=1 is used to indicate L3 CAT support, and ResID=2 indicates L2 CAT support. The ResID is also the sub-leaf index that software must use to query details of the CAT capability of that resource type (see Figure 18-31).

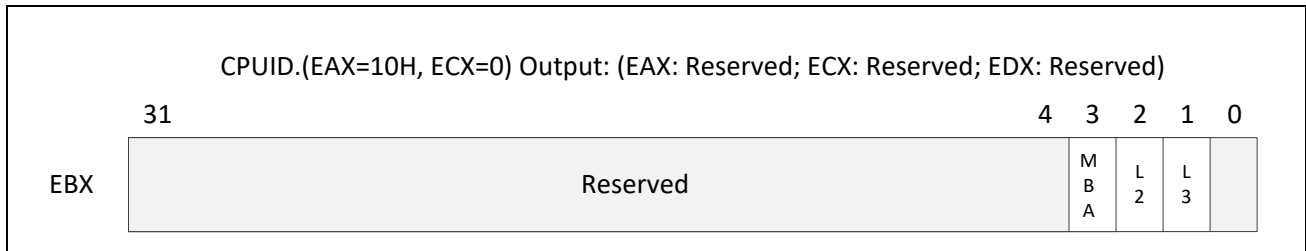


Figure 18-31. CPUID.(EAX=10H, ECX=0H) Available Resource Type Identification

- For ECX>0, EAX[4:0] reports the length of the capacity bitmask (ECX=1 or 2 for L3 CAT or L2 CAT respectively) using minus-one notation, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- Sub-functions of CPUID.EAX=10H with a non-zero ECX input matching a supported ResID enumerate the specific enforcement details of the corresponding ResID. The capabilities enumerated include the length of the capacity bitmasks and the number of Classes of Service for a given ResID. Software should query the capability of each available ResID that supports CAT from a sub-leaf of leaf 10H using the sub-leaf index reported by the corresponding non-zero bit in CPUID.(EAX=10H, ECX=0):EBX[31:1] in order to obtain additional feature details.
- CAT capability for L3 is enumerated by CPUID.(EAX=10H, ECX=1H), see Figure 18-32. The specific CAT capabilities reported by CPUID.(EAX=10H, ECX=1) are:

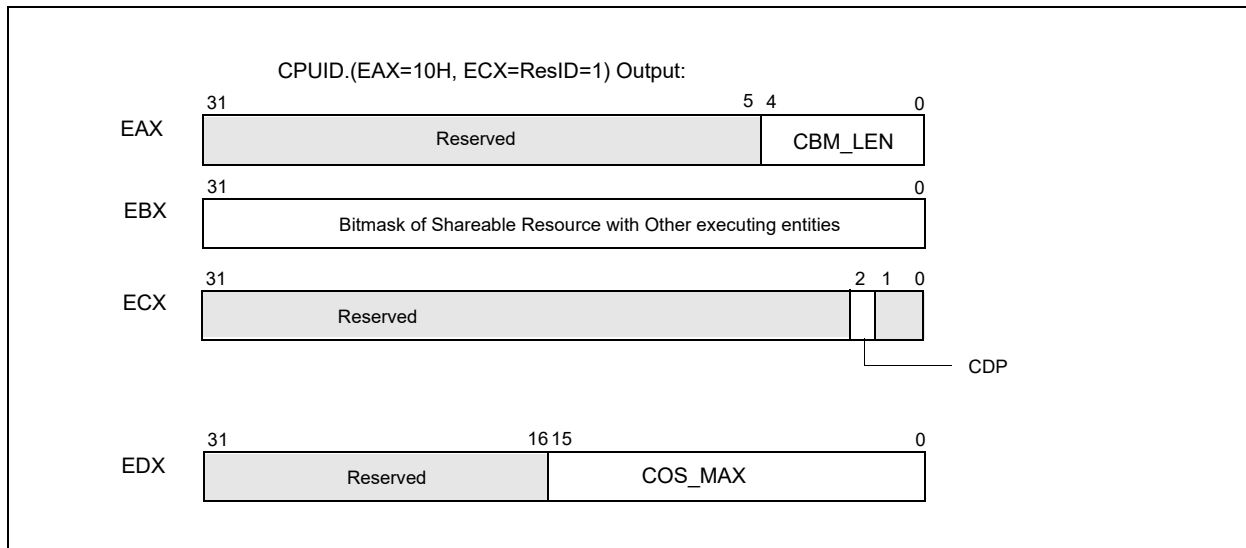


Figure 18-32. L3 Cache Allocation Technology and CDP Enumeration

- CPUID.(EAX=10H, ECX=ResID=1):EAX[4:0] reports the length of the capacity bitmask using minus-one notation, e.g., a value of 15 corresponds to the capability bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- CPUID.(EAX=10H, ECX=1):EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L3 allocation may be used by other entities in the platform (e.g., an

integrated graphics engine or hardware units outside the processor core and have direct access to L3). Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.

- CPUID.(EAX=10H, ECX=1):ECX.CDP[bit 2]: If 1, indicates L3 Code and Data Prioritization Technology is supported (see Section 18.19.5). Other bits of CPUID.(EAX=10H, ECX=1):ECX are reserved.
- CPUID.(EAX=10H, ECX=1):EDX[15:0] reports the maximum COS supported for the resource (COS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported COS). Bits 31:16 are reserved.
- CAT capability for L2 is enumerated by CPUID.(EAX=10H, ECX=2H), see Figure 18-33. The specific CAT capabilities reported by CPUID.(EAX=10H, ECX=2) are:

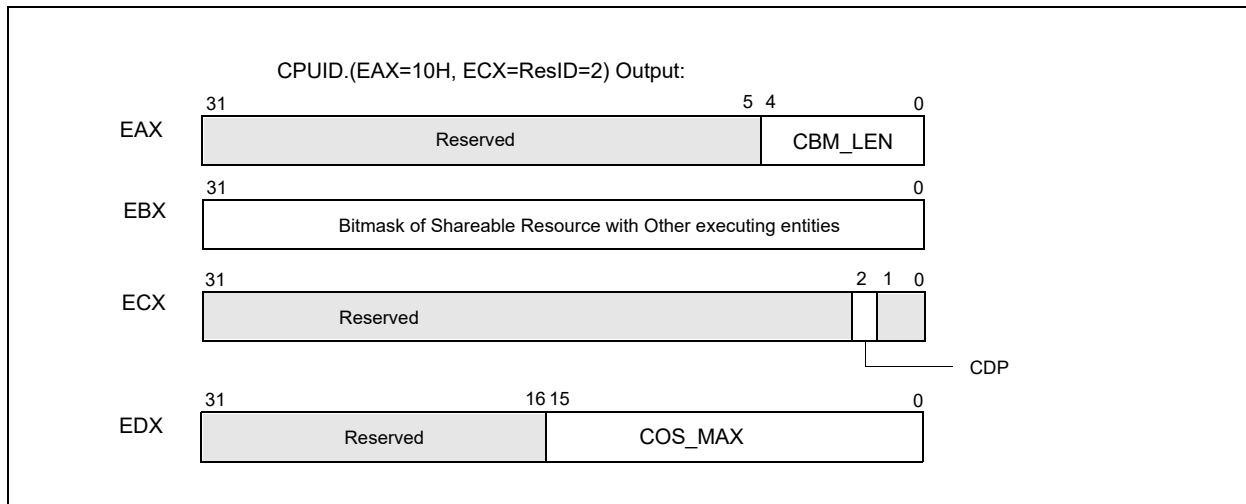


Figure 18-33. L2 Cache Allocation Technology

- CPUID.(EAX=10H, ECX=ResID=2):EAX[4:0] reports the length of the capacity bitmask using minus-one notation, e.g., a value of 15 corresponds to the capability bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- CPUID.(EAX=10H, ECX=2):EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L2 allocation may be used by other entities in the platform. Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
- CPUID.(EAX=10H, ECX=2):ECX.CDP[bit 2]: If 1, indicates L2 Code and Data Prioritization Technology is supported (see Section 17.19.6). Other bits of CPUID.(EAX=10H, ECX=2):ECX are reserved.
- CPUID.(EAX=10H, ECX=2):EDX[15:0] reports the maximum COS supported for the resource (COS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported COS). Bits 31:16 are reserved.

A note on migration of Classes of Service (COS): Software should minimize migrations of COS across logical processors (across threads or cores), as a reduction in the performance of the Cache Allocation Technology feature may result if COS are migrated frequently. This is aligned with the industry-standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and COS migration across processor logical threads and processor cores.

18.19.4.3 Cache Allocation Technology: Cache Mask Configuration

After determining the length of the capacity bitmasks (CBM) and number of COS supported using CPUID (see Section 18.19.4.2), each COS needs to be programmed with a CBM to dictate its available cache via a write to the corresponding IA32_resourceType_MASK_n register, where 'n' corresponds to a number within the supported range of COS, i.e., the range between 0 and CPUID.(EAX=10H, ECX=ResID):EDX[15:0], inclusive, and 'resourceType' corresponds to a specific resource as enumerated by the set bits of CPUID.(EAX=10H, ECX=0):EBX[31:1], for instance, 'L2' or 'L3' cache.

A hierarchy of MSRs is reserved for Cache Allocation Technology registers of the form IA32_resourceType_MASK_n:

- From 0C90H through 0D8FH (inclusive), providing support for multiple sub-ranges to support varying resource types. The first supported resourceType is 'L3', corresponding to the L3 cache in a platform. The MSRs range from 0C90H through 0D0FH (inclusive), enables support for up to 128 L3 CAT Classes of Service.

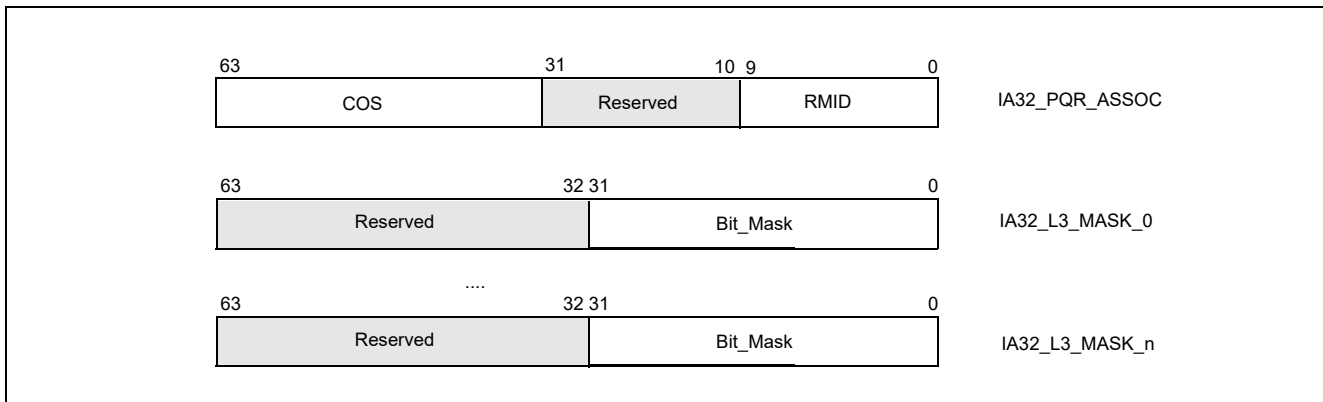


Figure 18-34. IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- Within the same CAT range hierarchy, another set of registers is defined for resourceType 'L2', corresponding to the L2 cache in a platform, and MSRs IA32_L2_MASK_n are defined for n=[0,63] at addresses 0D10H through 0D4FH (inclusive).

Figure 18-34 and Figure 18-35 provide an overview of the relevant registers.

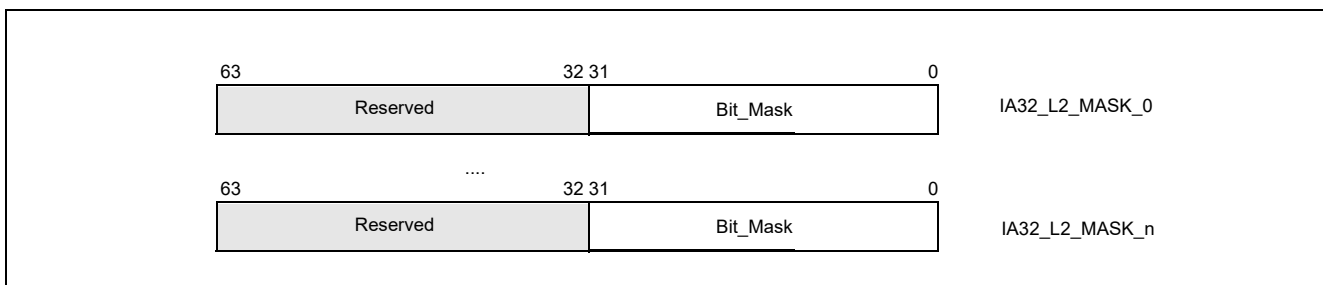


Figure 18-35. IA32_L2_MASK_n MSRs

All CAT configuration registers can be accessed using the standard RDMSR / WRMSR instructions.

Note that once L3 or L2 CAT masks are configured, threads can be grouped into Classes of Service (COS) using the IA32_PQR_ASSOC MSR as described in Chapter 18, "Class of Service to Cache Mask Association: Common Across Allocation Features".

18.19.4.4 Class of Service to Cache Mask Association: Common Across Allocation Features

After configuring the available classes of service with the preferred set of capacity bitmasks, the OS/VMM can set the IA32_PQR_ASSOC.COS of a logical processor to the class of service with the desired CBM when a thread context switch occurs. This allows the OS/VMM to indicate which class of service an executing thread/VM belongs

within. Each logical processor contains an instance of the IA32_PQR_ASSOC register at MSR location 0C8FH, and Figure 18-34 shows the bit field layout for this register. Bits[63:32] contain the COS field for each logical processor.

Note that placing the RMID field within the same PQR register enables both RMID and CLOS to be swapped at context swap time for simultaneous use of monitoring and allocation features with a single register write for efficiency.

When CDP is enabled, Specifying a COS value in IA32_PQR_ASSOC.COS greater than MAX_COS_CDP = (CPUID.(EAX=10H, ECX=1):EDX[15:0] >> 1) will cause undefined performance impact to code and data fetches. In all cases, code and data masks for L2 and L3 CDP should be programmed with at least one bit set.

Note that if the IA32_PQR_ASSOC.COS is never written then the CAT capability defaults to using COS 0, which in turn is set to the default mask in IA32_L3_MASK_0 - which is all “1”s (on reset). This essentially disables the enforcement feature by default or for legacy operating systems and software.

See Section 18.19.7, “Introduction to Memory Bandwidth Allocation” for important COS programming considerations including maximum values when using CAT and CDP.

18.19.5 Code and Data Prioritization (CDP): Enumerating and Enabling L3 CDP Technology

L3 CDP is an extension of L3 CAT. The presence of the L3 CDP feature is enumerated via CPUID.(EAX=10H, ECX=1):ECX.CDP[bit 2] (see Figure 18-32). Most of the CPUID.(EAX=10H, ECX=1) sub-leaf data that applies to CAT also apply to CDP. However, CPUID.(EAX=10H, ECX=1):EDX.COS_MAX_CAT specifies the maximum COS applicable to CAT-only operation. For CDP operations, COS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=1):EDX.COS_MAX_CAT >> 1).

If CPUID.(EAX=10H, ECX=1):ECX.CDP[bit 2] = 1, the processor supports CDP and provides a new MSR IA32_L3_QOS_CFG at address 0C81H. The layout of IA32_L3_QOS_CFG is shown in Figure 18-36. The bit field definition of IA32_L3_QOS_CFG are:

- Bit 0: L3 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.COS is COS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

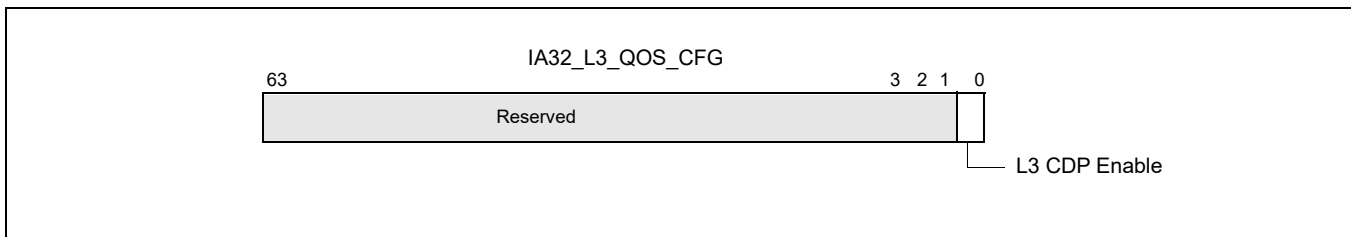


Figure 18-36. Layout of IA32_L3_QOS_CFG

IA32_L3_QOS_CFG default values are all 0s at RESET, the mask MSRs are all 1s. Hence, all logical processors are initialized in COS0 allocated with the entire L3 with CDP disabled, until software programs CAT and CDP. The scope of the IA32_L3_QOS_CFG MSR is defined to be the same scope as the L3 cache (e.g., typically per processor socket). Refer to Section 18.19.7 for software considerations while enabling or disabling L3 CDP.

18.19.5.1 Mapping Between L3 CDP Masks and CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per COS. The re-mapping is shown in Table 18-19.

Table 18-19. Re-indexing of COS Numbers and Mapping to CAT/CDP Mask MSRs

Mask MSR	CAT-only Operation	CDP Operation
IA32_L3_QOS_Mask_0	COS0	COS0.Data
IA32_L3_QOS_Mask_1	COS1	COS0.Code
IA32_L3_QOS_Mask_2	COS2	COS1.Data
IA32_L3_QOS_Mask_3	COS3	COS1.Code
IA32_L3_QOS_Mask_4	COS4	COS2.Data
IA32_L3_QOS_Mask_5	COS5	COS2.Code
....
IA32_L3_QOS_Mask_‘2n’	COS‘2n’	COS‘n’.Data
IA32_L3_QOS_Mask_‘2n+1’	COS‘2n+1’	COS‘n’.Code

One can derive the MSR address for the data mask or code mask for a given COS number ‘n’ by:

- data_mask_address (n) = base + (n <<1), where base is the address of IA32_L3_QOS_MASK_0.
- code_mask_address (n) = base + (n <<1) +1.

When CDP is enabled, each COS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 18-29).

Mask MSR field definitions remain the same. Capacity masks must be formed of contiguous set bits, with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

18.19.6 Code and Data Prioritization (CDP): Enumerating and Enabling L2 CDP Technology

L2 CDP is an extension of the L2 CAT feature. The presence of the L2 CDP feature is enumerated via CPUID.(EAX=10H, ECX=2):ECX.CDP[bit 2] (see Figure 17-33). Most of the CPUID.(EAX=10H, ECX=2) sub-leaf data that applies to CAT also apply to CDP. However, CPUID.(EAX=10H, ECX=2):EDX.COS_MAX_CAT specifies the maximum COS applicable to CAT-only operation. For CDP operations, COS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=2):EDX.COS_MAX_CAT >>1).

If CPUID.(EAX=10H, ECX=2):ECX.CDP[bit 2] =1, the processor supports L2 CDP and provides a new MSR IA32_L2_QOS_CFG at address 0C82H. The layout of IA32_L2_QOS_CFG is shown in Figure 18-37. The bit field definition of IA32_L2_QOS_CFG are:

- Bit 0: L2 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.COS is COS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

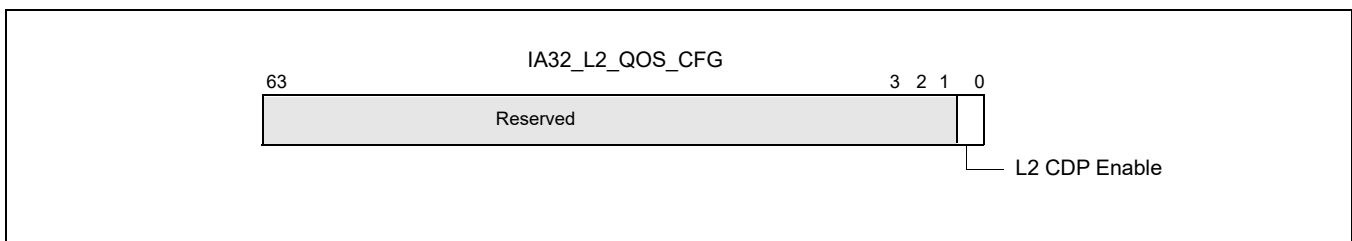


Figure 18-37. Layout of IA32_L2_QOS_CFG

IA32_L2_QOS_CFG default values are all 0s at RESET, and the mask MSRs are all 1s. Hence all logical processors are initialized in COS0 allocated with the entire L2 available and with CDP disabled, until software programs CAT and CDP. The IA32_L2_QOS_CFG MSR is defined at the same scope as the L2 cache, typically at the module level for Intel Atom processors for instance. In processors with multiple modules present it is recommended to program the IA32_L2_QOS_CFG MSR consistently across all modules for simplicity.

18.19.6.1 Mapping Between L2 CDP Masks and L2 CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per COS. This remapping is the same as the remapping shown in Table 17-19 for L3 CDP, but for the L2 MSR block (IA32_L2_QOS_MASK_n) instead of the L3 MSR block (IA32_L3_QOS_MASK_n). The same code / data mask mapping algorithm applies to remapping the MSR block between code and data masks.

As with L3 CDP, when L2 CDP is enabled, each COS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 18-29).

Mask MSR field definitions for L2 CDP remain the same as for L2 CAT. Capacity masks must be formed of contiguous set bits, with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

18.19.6.2 Common L2 and L3 CDP Programming Considerations

Before enabling or disabling L2 or L3 CDP, software should write all 1's to all of the corresponding CAT/CDP masks to ensure proper behavior (e.g., the IA32_L3_QOS_Mask_n set of MSRs for the L3 CAT feature). When enabling CDP, software should also ensure that only COS number which are valid in CDP operation is used, otherwise undefined behavior may result. For instance in a case with 16 CAT COS, since COS are reduced by half when CDP is enabled, software should ensure that only COS 0-7 are in use before enabling CDP (along with writing 1's to all mask bits before enabling or disabling CDP).

Software should also account for the fact that mask interpretations change when CDP is enabled or disabled, meaning for instance that a CAT mask for a given COS may become a code mask for a different Class of Service when CDP is enabled. In order to simplify this behavior and prevent unintended remapping software should consider resetting all threads to COS[0] before enabling or disabling CDP.

18.19.6.3 Cache Allocation Technology Dynamic Configuration

All Resource Director Technology (RDT) interfaces including the IA32_PQR_ASSOC MSR, CAT/CDP masks, MBA delay values and CQM/MBM registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,
- Accessing a QOS mask register outside the supported COS (the max COS number is specified in CPUID.(EAX=10H, ECX=ResID):EDX[15:0]), or
- Writing a COS greater than the supported maximum (specified as the maximum value of CPUID.(EAX=10H, ECX=ResID):EDX[15:0] for all valid ResID values) is written to the IA32_PQR_ASSOC.CLOS field.

When CDP is enabled, specifying a COS value in IA32_PQR_ASSOC.COS outside of the lower half of the COS space will cause undefined performance impact to code and data fetches due to MSR space re-indexing into code/data masks when CDP is enabled.

When reading the IA32_PQR_ASSOC register the currently programmed COS on the core will be returned.

When reading an IA32_resourceType_MASK_n register the current capacity bit mask for COS 'n' will be returned.

As noted previously, software should minimize migrations of COS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if COS are migrated frequently.

This is aligned with the industry standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and COS migration across processor logical threads and processor cores.

18.19.6.4 Cache Allocation Technology Operation With Power Saving Features

Note that the Cache Allocation Technology feature cannot be used to enforce cache coherency, and that some advanced power management features such as C-states which may shrink or power off various caches within the system may interfere with CAT hints - in such cases the CAT bitmasks are ignored and the other features take precedence. If the highest possible level of CAT differentiation or determinism is required, disable any power-saving features which shrink the caches or power off caches. The details of the power management interfaces are typically implementation-specific, but can be found at Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

If software requires differentiation between threads but not absolute determinism then in many cases it is possible to leave power-saving cache shrink features enabled, which can provide substantial power savings and increase battery life in mobile platforms. In such cases when the caches are powered off (e.g., package C-states) the entire cache of a portion thereof may be powered off. Upon resuming an active state any new incoming data to the cache will be filled subject to the cache capacity bitmasks. Any data in the cache prior to the cache shrink or power off may have been flushed to memory during the process of entering the idle state, however, and is not guaranteed to remain in the cache. If differentiation between threads is the goal of system software then this model allows substantial power savings while continuing to deliver performance differentiation. If system software needs optimal determinism then power saving modes which flush portions of the caches and power them off should be disabled.

NOTE

IA32_PQR_ASSOC is saved and restored across C6 entry/exit. Similarly, the mask register contents are saved across package C-state entry/exit and are not lost.

18.19.6.5 Cache Allocation Technology Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and mask registers are unmodified across an SMI delivery. Thus, the execution of SMM handler code can interact with the Cache Allocation Technology resource and manifest some degree of non-determinism to the non-SMM software stack. An SMM handler may also perform certain system-level or power management practices that affect CAT operation.

It is possible for an SMM handler to minimize the impact on data determinism in the cache by reserving a COS with a dedicated partition in the cache. Such an SMM handler can switch to the dedicated COS immediately upon entering SMM, and switching back to the previously running COS upon exit.

18.19.6.6 Associating Threads with CAT/CDP Classes of Service

Threads are associated with Classes of Service (CLOS) via the per-logical-processor IA32_PQR_ASSOC MSR. The same COS concept applies to both CAT and CDP (for instance, COS[5] means the same thing whether CAT or CDP is in use, and the COS has associated resource usage constraint attributes including cache capacity masks). The mapping of COS to mask MSRs does change when CDP is enabled, according to the following guidelines:

- In CAT-only Mode - one set of bitmasks in one mask MSR control both code and data.
 - Each COS number map 1:1 with a capacity mask on the applicable resource (e.g., L3 cache).
- When CDP is enabled,
 - Two mask sets exist for each COS number, one for code, one for data.
 - Masks for code/data are interleaved in the MSR address space (see Table 18-19).

18.19.7 Introduction to Memory Bandwidth Allocation

The Memory Bandwidth Allocation (MBA) feature provides indirect and approximate control over memory bandwidth available per-core, and was introduced on the Intel Xeon Scalable Processor Family. This feature provides a

method to control applications which may be over-utilizing bandwidth relative to their priority in environments such as the data-center.

The MBA feature uses existing constructs from the Resource Director Technology (RDT) feature set including Classes of Service (CLOS). A given CLOS used for L3 CAT for instance means the same thing as a CLOS used for MBA. Infrastructure such as the MSR used to associate a thread with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID leaf 10H) are shared.

The high-level implementation of Memory Bandwidth Allocation is shown in Figure 18-38.

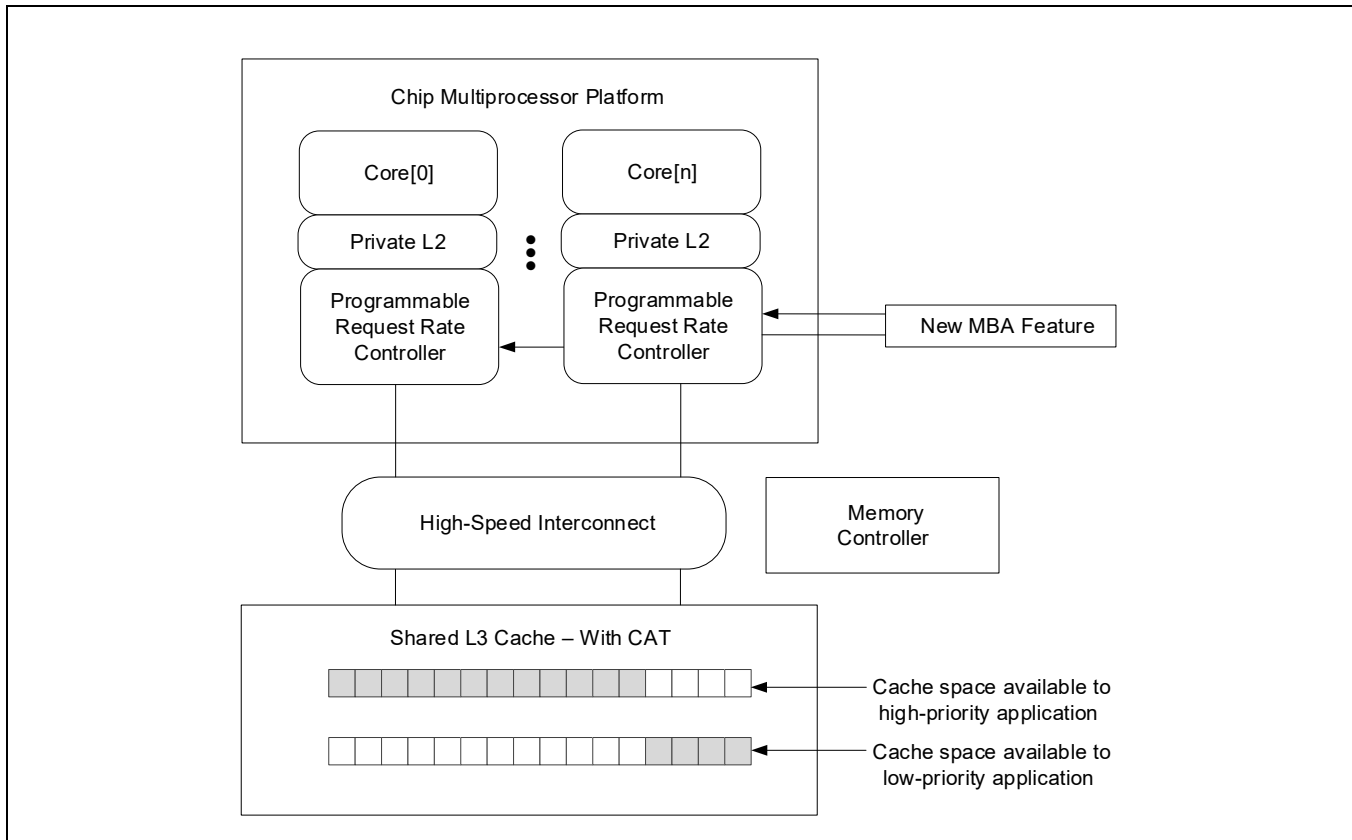


Figure 18-38. A High-Level Overview of the MBA Feature

As shown in Figure 18-38, the MBA feature introduces a programmable request rate controller between the cores and the high-speed interconnect, enabling indirect control over memory bandwidth for cores over-utilizing bandwidth relative to their priority. For instance, high-priority cores may be run un-throttled, but lower priority cores generating an excessive amount of traffic may be throttled to enable more bandwidth availability for the high-priority cores.

Since MBA uses a programmable rate controller between the cores and the interconnect, higher-level shared caches and memory controller, bandwidth to these caches may also be reduced, so care should be taken to throttle only bandwidth-intense applications which do not use the off-core caches effectively.

The throttling values exposed by MBA are approximate, and are calibrated to specific traffic patterns. As work-load characteristics vary, the throttling values provided may affect each workload differently. In cases where precise control is needed, the Memory Bandwidth Monitoring (MBM) feature can be used as input to a software controller which makes decisions about the MBA throttling level to apply.

Enumeration and configuration details are discussed below followed by usage model considerations.

18.19.7.1 Memory Bandwidth Allocation Enumeration

Similar to other RDT features, enumeration of the presence and details of the MBA feature is provided via a sub-leaf of the CPUID instruction.

Key components of the enumeration are as follows.

- Support for the MBA feature on the processor, and if MBA is supported, the following details:
 - Number of supported Classes of Service (CLOS) for the processor.
 - The maximum MBA delay value supported (which also implicitly provides a definition of the granularity).
 - An indication of whether the delay values which can be programmed are linearly spaced or not.

The presence of any of the RDT features which enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.(EAX=07H, ECX=0):EBX.PQE[bit 15] reports 1, the processor supports software control over shared processor resources. Software may then use CPUID leaf 10H to enumerate additional details on the specific controls provided.

Through CPUID leaf 10H software may determine whether MBA is supported on the platform. Specifically, as shown in Figure 17-31, bit 3 of the EBX register indicates whether MBA is supported on the processor, and the bit position (3) constitutes a Resource ID (ResID) which allows enumeration of MBA details. For instance, if bit 3 is supported this implies the presence of CPUID.10H.[ResID=3] as shown in Figure 18-39 which provides the following details.

- CPUID.(EAX=10H, ECX=ResID=3):EAX[11:0] reports the maximum MBA throttling value supported, minus one. For instance, a value of 89 indicates that a maximum throttling value of 90 is supported. Additionally, in cases where a linear interface (see below) is supported then one hundred minus the maximum throttling value indicates the granularity, 10% in this example.
- CPUID.(EAX=10H, ECX=ResID=3):EBX is reserved.
- CPUID.(EAX=10H, ECX=ResID=3):ECX[2] reports whether the response of the delay values is linear (see text).
- CPUID.(EAX=10H, ECX=ResID=3):EDX[15:0] reports the number of Classes of Service (CLOS) supported for the feature (minus one). For instance, a reported value of 15 implies a maximum of 16 supported MBA CLOS.

The number of CLOS supported for the MBA feature may or may not align with other resources such as L3 CAT. In cases where the RDT features support different numbers of CLOS the lowest numerical CLOS support the common set of features, while higher CLOS may support a subset. For instance, if L3 CAT supports 8 CLOS while MBA supports 4 CLOS, all 8 CLOS would have L3 CAT masks available for cache control, but the upper 4 CLOS would not offer MBA support. In this case the upper 4 CLOS would not be subject to any throttling control. Software can manage supported resources / CLOS in order to either have consistent capabilities across CLOS by using the common subset or enable more flexibility by selectively applying resource control where needed based on careful CLOS and thread mapping. In all cases, CLOS[0] supports all RDT resource control features present on the platform.

Discussion on the interpretation and usage of the MBA delay values is provided in Section 18.19.7.2 on MBA configuration.

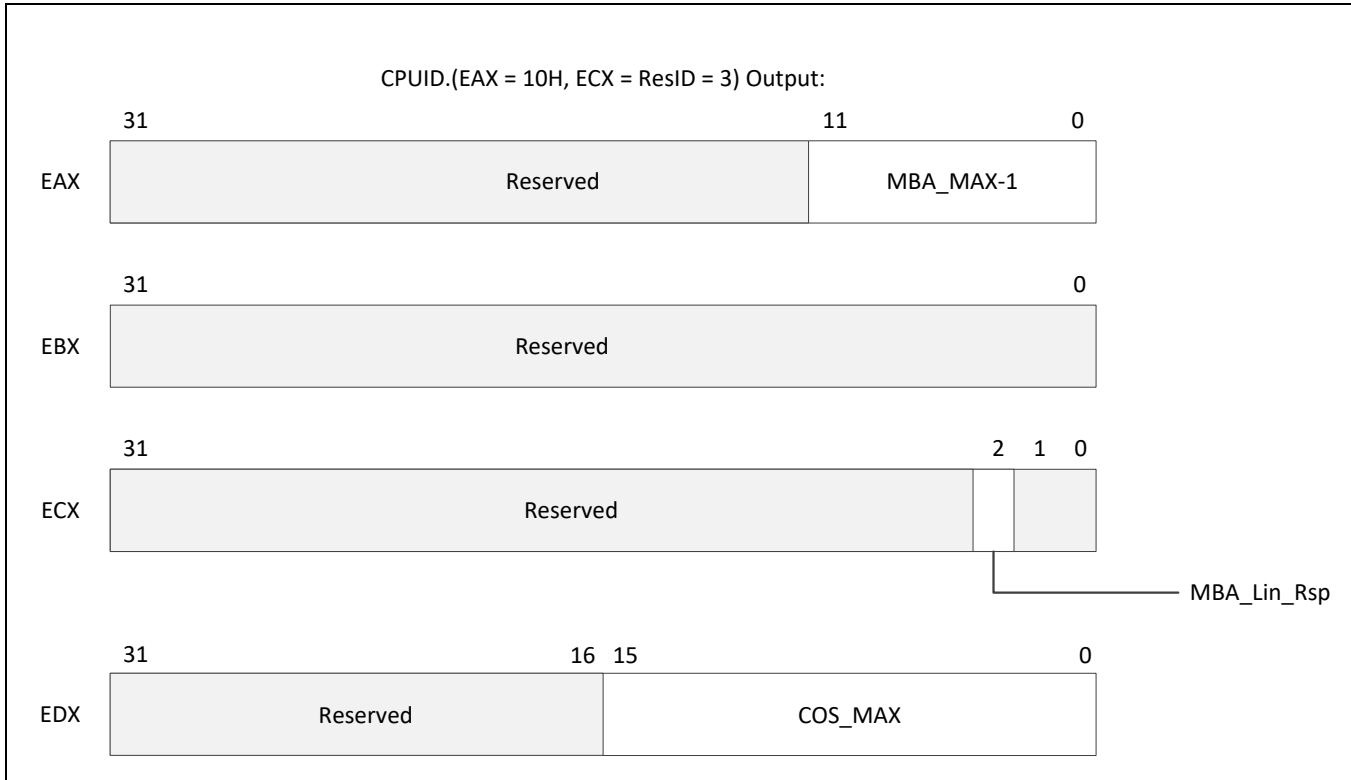


Figure 18-39. CPUID.(EAX=10H, ECX=3H) MBA Feature Details Identification

18.19.7.2 Memory Bandwidth Allocation Configuration

The configuration of MBA takes consists of two processes once enumeration is complete.

- Association of threads to Classes of Service (CLOS) - accomplished in a common fashion across RDT features as described in Section 18.19.7.1 via the IA32_PQR_ASSOC MSR. As with features such as L3 CAT, software may update the CLOS field of the PQR MSR at context swap time in order to maintain the proper association of software threads to Classes of Service on the hardware. While logical processors may each be associated with independent CLOS, see Section 18.19.7.3 for important usage model considerations (initial versions of the MBA feature select the maximum delay value across threads).
- Configuration of the per-CLOS delay values, accomplished via the IA32_L2_QoS_Ext_BW_Thrtl_n MSR set shown in Table 18-20.

The MBA delay values which may be programmed range from zero (implying zero delay, and full bandwidth available) to the maximum (MBA_MAX) specified in CPUID as discussed in Section 18.19.7.1. The throttling values are approximate and do not sum to 100% across CLOS, rather they should be viewed as a maximum bandwidth “cap” per-CLOS.

Software may select an MBA delay value then write the value into one or more of the IA32_L2_QoS_Ext_BW_Thrtl_n MSRs to update the delay values applied for a specific CLOS. As shown in Table 18-20 the base address of the MSRs is at D50H, and the range corresponds to the maximum supported CLOS from CPUID.(EAX=10H, ECX=ResID=1):EDX[15:0] as described in Section 18.19.7.1. For instance, if 16 CLOS are supported then the valid MSR range will extend from D50H through D5F inclusive.

Table 18-20. MBA Delay Value MSRs

Delay Value MSR	Address
IA32_L2_QoS_Ext_BW_Thrtl_0	D50H
IA32_L2_QoS_Ext_BW_Thrtl_1	D51H
IA32_L2_QoS_Ext_BW_Thrtl_2	D52H
....
IA32_L2_QoS_Ext_BW_Thrtl_'COS_MAX'	D50H + COS_MAX from CPUID.10H.3

The definition for the MBA delay value MSRs is provided in Figure 17.39. The lower 16 bits are used for MBA delay values, and values from zero to the maximum from the CPUID MBA_MAX-1 value are supported. Values outside this range will generate #GP(0).

If linear input throttling values are indicated by CPUID.(EAX=10H, ECX=ResID=3):ECX[bit 2] then values from zero through the MBA_MAX field from CPUID.(EAX=10H, ECX=ResID=3):EAX[11:0] are supported as inputs. In the linear mode the input precision is defined as 100-(MBA_MAX). For instance, if the MBA_MAX value is 90, the input precision is 10%. Values not an even multiple of the precision (e.g., 12%) will be rounded down (e.g., to 10% delay applied).

- If linear values are not supported (CPUID.(EAX=10H, ECX=ResID=3):ECX[bit 2] = 0) then input delay values are powers-of-two from zero to the MBA_MAX value from CPUID. In this case any values not a power of two will be rounded down the next nearest power of two.

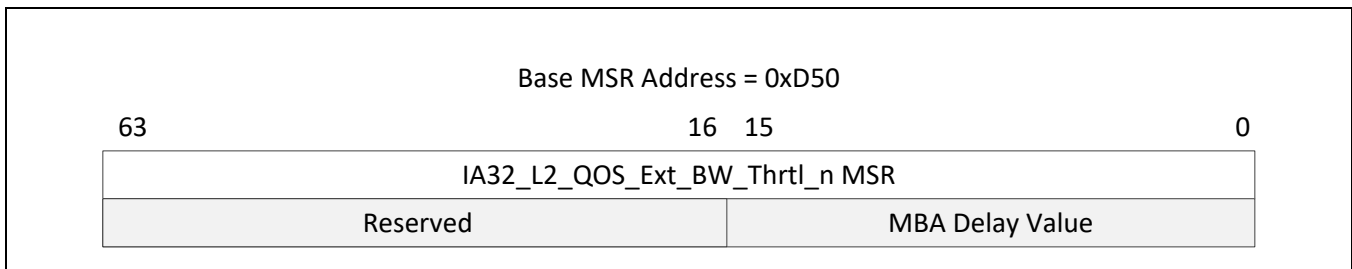


Figure 18-40. IA32_L2_QoS_Ext_BW_Thrtl_n MSR Definition

Note that the throttling values provided to software are calibrated through specific traffic patterns, however as workload characteristics may vary the response precision and linearity of the delay values will vary across products, and should be treated as approximate values only.

18.19.7.3 Memory Bandwidth Allocation Usage Considerations

As the memory bandwidth control that MBA provides is indirect and approximate, using the feature with a closed-loop controller to also monitor memory bandwidth and how effectively the applications use the cache (via the Cache Monitoring Technology feature) may provide additional value. This approach also allows administrators to provide a band-width target or set-point which a controller could use to guide MBA throttling values applied, and this allows bandwidth control independent of the execution characteristics of the application.

As control is provided per processor core (the max of the delay values of the per-thread CLOS applied to the core) care should be taking in scheduling threads so as to not inadvertently place a high-priority thread (with zero intended MBA throttling) next to a low-priority thread (with MBA throttling intended), which would lead to inadvertent throttling of the high-priority thread.

11. Updates to Chapter 20, Volume 3B

Change bars and green text show changes to Chapter 20 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- Updated Table 20-46, "Core PMU Summary of the Ice Lake Microarchitecture," to include the number of programmable counters per thread.

Intel 64 and IA-32 architectures provide facilities for monitoring performance via a PMU (Performance Monitoring Unit).

NOTE

Performance monitoring events can be found here: <https://perfmon-events.intel.com/>.

Additionally, performance monitoring event files for Intel processors are hosted by the Intel Open Source Technology Center. These files can be downloaded here:

<https://download.01.org/perfmon/>.

20.1 PERFORMANCE MONITORING OVERVIEW

Performance monitoring was introduced in the Pentium processor with a set of model-specific performance-monitoring counter MSR. These counters permit selection of processor performance parameters to be monitored and measured. The information obtained from these counters can be used for tuning system and compiler performance.

In Intel P6 family of processors, the performance monitoring mechanism was enhanced to permit a wider selection of events to be monitored and to allow greater control events to be monitored. Next, Intel processors based on Intel NetBurst microarchitecture introduced a distributed style of performance monitoring mechanism and performance events.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, and Intel processors based on Intel NetBurst microarchitecture are not architectural. They are all model specific (not compatible among processor families). Intel Core Solo and Intel Core Duo processors support a set of architectural performance events and a set of non-architectural performance events. Newer Intel processor generations support enhanced architectural performance events and non-architectural performance events.

Starting with Intel Core Solo and Intel Core Duo processors, there are two classes of performance monitoring capabilities. The first class supports events for monitoring performance using counting or interrupt-based event sampling usage. These events are non-architectural and vary from one processor model to another. They are similar to those available in Pentium M processors. These non-architectural performance monitoring events are specific to the microarchitecture and may change with enhancements. They are discussed in Section 20.6.3, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)." Non-architectural events for a given microarchitecture cannot be enumerated using CPUID; and they can be found at: <https://perfmon-events.intel.com/>.

The second class of performance monitoring capabilities is referred to as architectural performance monitoring. This class supports the same counting and interrupt-based event sampling usages, with a smaller set of available events. The visible behavior of architectural performance events is consistent across processor implementations. Availability of architectural performance monitoring capabilities is enumerated using the CPUID.0AH. These events are discussed in Section 20.2.

See also:

- Section 20.2, "Architectural Performance Monitoring."
- Section 20.3, "Performance Monitoring (Intel® Core™ Processors and Intel® Xeon® Processors)."
 - Section 20.3.1, "Performance Monitoring for Processors Based on Nehalem Microarchitecture."
 - Section 20.3.2, "Performance Monitoring for Processors Based on Westmere Microarchitecture."
 - Section 20.3.3, "Intel® Xeon® Processor E7 Family Performance Monitoring Facility."
 - Section 20.3.4, "Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture."
 - Section 20.3.5, "3rd Generation Intel® Core™ Processor Performance Monitoring Facility."

- Section 20.3.6, “4th Generation Intel® Core™ Processor Performance Monitoring Facility.”
- Section 20.3.7, “5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility.”
- Section 20.3.8, “6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility.”
- Section 20.3.9, “10th Generation Intel® Core™ Processor Performance Monitoring Facility.”
- Section 20.3.10, “12th and 13th Generation Intel® Core™ Processors, and 4th Generation Intel® Xeon® Scalable Processor Family Performance Monitoring Facility.”
- Section 20.4, “Performance monitoring (Intel® Xeon™ Phi Processors).”
 - Section 20.4.1, “Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring.”
- Section 20.5, “Performance Monitoring (Intel Atom® Processors).”
 - Section 20.5.1, “Performance Monitoring (45 nm and 32 nm Intel Atom® Processors).”
 - Section 20.5.2, “Performance Monitoring for Silvermont Microarchitecture.”
 - Section 20.5.3, “Performance Monitoring for Goldmont Microarchitecture.”
 - Section 20.5.4, “Performance Monitoring for Goldmont Plus Microarchitecture.”
 - Section 20.5.5, “Performance Monitoring for Tremont Microarchitecture.”
- Section 20.6, “Performance Monitoring (Legacy Intel Processors).”
 - Section 20.6.1, “Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors).”
 - Section 20.6.2, “Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture).”
 - Section 20.6.3, “Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture).”
 - Section 20.6.4, “Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture.”
 - Section 20.6.4.5, “Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture.”
 - Section 20.6.5, “Performance Monitoring and Dual-Core Technology.”
 - Section 20.6.6, “Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache.”
 - Section 20.6.7, “Performance Monitoring on L3 and Caching Bus Controller Sub-Systems.”
 - Section 20.6.8, “Performance Monitoring (P6 Family Processor).”
 - Section 20.6.9, “Performance Monitoring (Pentium Processors).”
- Section 20.7, “Counting Clocks.”
- Section 20.8, “IA32_PERF_CAPABILITIES MSR Enumeration.”
- Section 20.9, “PEBS Facility.”

20.2 ARCHITECTURAL PERFORMANCE MONITORING

Performance monitoring events are architectural when they behave consistently across microarchitectures. Intel Core Solo and Intel Core Duo processors introduced architectural performance monitoring. The feature provides a mechanism for software to enumerate performance events and provides configuration and counting facilities for events.

Architectural performance monitoring does allow for enhancement across processor implementations. The CPUID.0AH leaf provides version ID for each enhancement. Intel Core Solo and Intel Core Duo processors support base level functionality identified by version ID of 1. Processors based on Intel Core microarchitecture support, at a minimum, the base level functionality of architectural performance monitoring. Intel Core 2 Duo processor T

7700 and newer processors based on Intel Core microarchitecture support both the base level functionality and enhanced architectural performance monitoring identified by version ID of 2.

45 nm and 32 nm Intel Atom processors and Intel Atom processors based on the Silvermont microarchitecture support the functionality provided by versionID 1, 2, and 3; CPUID.0AH:EAX[7:0] reports versionID = 3 to indicate the aggregate of architectural performance monitoring capabilities. Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture. Intel Atom processors based on the Goldmont and Goldmont Plus microarchitectures support versionID 4. Intel Atom processors starting with processors based on the Tremont microarchitecture support versionID 5.

Intel Core processors and related Intel Xeon processor families based on the Nehalem through Broadwell microarchitectures support version ID 3. Intel processors based on the Skylake through Coffee Lake microarchitectures support versionID 4. Intel processors starting with processors based on the Ice Lake microarchitecture support versionID 5.

20.2.1 Architectural Performance Monitoring Version 1

Configuring an architectural performance monitoring event involves programming performance event select registers. There are a finite number of performance event select MSRs (IA32_PERFEVTSELx MSRs). The result of a performance monitoring event is reported in a performance monitoring counter (IA32_PMCx MSR). Performance monitoring counters are paired with performance monitoring select registers.

Performance monitoring select registers and counters are architectural in the following respects:

- Bit field layout of IA32_PERFEVTSELx is consistent across microarchitectures.
- Addresses of IA32_PERFEVTSELx MSRs remain the same across microarchitectures.
- Addresses of IA32_PMC MSRs remain the same across microarchitectures.
- Each logical processor has its own set of IA32_PERFEVTSELx and IA32_PMCx MSRs. Configuration facilities and counters are not shared between logical processors sharing a processor core.

Architectural performance monitoring provides a CPUID mechanism for enumerating the following information:

- Number of performance monitoring counters available to software in a logical processor (each IA32_PERFEVTSELx MSR is paired to the corresponding IA32_PMCx MSR).
- Number of bits supported in each IA32_PMCx.
- Number of architectural performance monitoring events supported in a logical processor.

Software can use CPUID to discover architectural performance monitoring availability (CPUID.0AH). The architectural performance monitoring leaf provides an identifier corresponding to the version number of architectural performance monitoring available in the processor.

The version identifier is retrieved by querying CPUID.0AH:EAX[bits 7:0] (see Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). If the version identifier is greater than zero, architectural performance monitoring capability is supported. Software queries the CPUID.0AH for the version identifier first; it then analyzes the value returned in CPUID.0AH.EAX, CPUID.0AH.EBX to determine the facilities available.

In the initial implementation of architectural performance monitoring; software can determine how many IA32_PERFEVTSELx/ IA32_PMCx MSR pairs are supported per core, the bit-width of PMC, and the number of architectural performance monitoring events available.

20.2.1.1 Architectural Performance Monitoring Version 1 Facilities

Architectural performance monitoring facilities include a set of performance monitoring counters and performance event select registers. These MSRs have the following properties:

- IA32_PMCx MSRs start at address 0C1H and occupy a contiguous block of MSR address space; the number of MSRs per logical processor is reported using CPUID.0AH:EAX[15:8]. Note that this may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.

- IA32_PERFEVTSELx MSRs start at address 186H and occupy a contiguous block of MSR address space. Each performance event select register is paired with a corresponding performance counter in the 0C1H address block. Note the number of IA32_PERFEVTSELx MSRs may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.
- The bit width of an IA32_PMCx MSR is reported using the CPUID.0AH:EAX[23:16]. This the number of valid bits for read operation. On write operations, the lower-order 32 bits of the MSR may be written with any value, and the high-order bits are sign-extended from the value of bit 31.
- Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally.

See Figure 20-1 for the bit field layout of IA32_PERFEVTSELx MSRs. The bit fields are:

- Event select field (bits 0 through 7)** — Selects the event logic unit used to detect microarchitectural conditions (see Table 20-1, for a list of architectural events and their 8-bit codes). The set of values for this field is defined architecturally; each value corresponds to an event logic unit for use with an architectural performance event. The number of architectural events is queried using CPUID.0AH:EAX. A processor may support only a subset of pre-defined values.

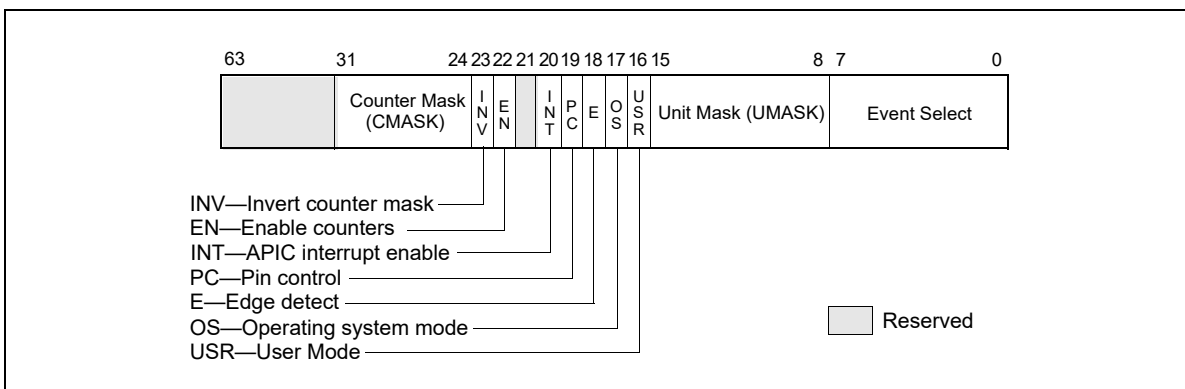


Figure 20-1. Layout of IA32_PERFEVTSELx MSRs

- Unit mask (UMASK) field (bits 8 through 15)** — These bits qualify the condition that the selected event logic unit detects. Valid UMASK values for each event logic unit are specific to the unit. For each architectural performance event, its corresponding UMASK value defines a specific microarchitectural condition. A pre-defined microarchitectural condition associated with an architectural event may not be applicable to a given processor. The processor then reports only a subset of pre-defined architectural events. Pre-defined architectural events are listed in Table 20-1; support for pre-defined architectural events is enumerated using CPUID.0AH:EBX.
- USR (user mode) flag (bit 16)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege levels 1, 2 or 3. This flag can be used with the OS flag.
- OS (operating system mode) flag (bit 17)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege level 0. This flag can be used with the USR flag.
- E (edge detect) flag (bit 18)** — Enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields. The mechanism does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).
- PC (pin control) flag (bit 19)** — Beginning with Sandy Bridge microarchitecture, this bit is reserved (not writeable). On processors based on previous microarchitectures, the logical processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.

- **INT (APIC interrupt enable) flag (bit 20)** — When set, the logical processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — When set, performance counting is enabled in the corresponding performance-monitoring counter; when clear, the corresponding counter is disabled. The event logic unit for a UMASK must be disabled by setting IA32_PERFEVTSELx[bit 22] = 0, before writing to IA32_PMCx.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When this field is not zero, a logical processor compares this mask to the events count of the detected microarchitectural condition during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented.

This mask is intended for software to characterize microarchitectural conditions that can count multiple occurrences per cycle (for example, two or more instructions retired per clock; or bus queue occupations). If the counter-mask field is 0, then the counter is incremented each cycle by the event count associated with multiple occurrences.

20.2.1.2 Pre-defined Architectural Performance Events

Table 20-1 lists architecturally defined events.

Table 20-1. UMask and Event Select Encodings for Pre-Defined Architectural Performance Events

Bit Position CPUID.AH.EBX	Event Name	UMask	Event Select
0	UnHalted Core Cycles	00H	3CH
1	Instruction Retired	00H	C0H
2	UnHalted Reference Cycles ¹	01H	3CH
3	LLC Reference	4FH	2EH
4	LLC Misses	41H	2EH
5	Branch Instruction Retired	00H	C4H
6	Branch Misses Retired	00H	C5H
7	Topdown Slots	01H	A4H

NOTES:

1. Implementations prior to the 12th generation Intel® Core™ processor P-cores count at core crystal clock, TSC, or bus clock frequency.

A processor that supports architectural performance monitoring may not support all the predefined architectural performance events (Table 20-1). The number of architectural events is reported through CPUID.0AH:EAX[31:24], while non-zero bits in CPUID.0AH:EBX indicate any architectural events that are not available.

The behavior of each architectural performance event is expected to be consistent on all processors that support that event. Minor variations between microarchitectures are noted below:

- **UnHalted Core Cycles** — Event select 3CH, Umask 00H
This event counts core clock cycles when the clock signal on a specific core is running (not halted). The counter does not advance in the following conditions:
 - An ACPI C-state other than C0 for normal operation.
 - HLT.
 - STPCLK# pin asserted.
 - Being throttled by TM1.
 - During the frequency switching phase of a performance state transition (see Chapter 15, “Power and Thermal Management”).

The performance counter for this event counts across performance state transitions using different core clock frequencies.

- **Instructions Retired** — Event select C0H, Umask 00H

This event counts the number of instructions at retirement. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. An instruction with a REP prefix counts as one instruction (not per iteration). Faults before the retirement of the last micro-op of a multi-ops instruction are not counted.

This event does not increment under VM-exit conditions. Counters continue counting during hardware interrupts, traps, and inside interrupt handlers.

- **UnHalted Reference Cycles** — Event select 3CH, Umask 01H

This event counts reference clock cycles at a fixed frequency while the clock signal on the core is running. The event counts at a fixed frequency, irrespective of core frequency changes due to performance state transitions. Processors may implement this behavior differently. Current implementations use the core crystal clock, TSC or the bus clock. Because the rate may differ between implementations, software should calibrate it to a time source with known frequency.

- **Last Level Cache References** — Event select 2EH, Umask 4FH

This event counts requests originating from the core that reference a cache line in the last level on-die cache. The event count includes speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Last Level Cache Misses** — Event select 2EH, Umask 41H

This event counts each cache miss condition for references to the last level on-die cache. The event count may include speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Branch Instructions Retired** — Event select C4H, Umask 00H

This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction.

- **All Branch Mispredict Retired** — Event select C5H, Umask 00H

This event counts mispredicted branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction in the architectural path of execution and experienced misprediction in the branch prediction hardware.

Branch prediction hardware is implementation-specific across microarchitectures; value comparison to estimate performance differences is not recommended.

- **Topdown Slots** — Event select A4H, Umask 01H

This event counts the total number of available slots for an unhalted logical processor.

The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core, in processors that support Intel Hyper-Threading Technology.

Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

NOTE

Programming decisions or software precisions on functionality should not be based on the event values or dependent on the existence of performance monitoring events.

20.2.2 Architectural Performance Monitoring Version 2

The enhanced features provided by architectural performance monitoring version 2 include the following:

- **Fixed-function performance counter register and associated control register** — Three of the architectural performance events are counted using three fixed-function MSRs (IA32_FIXED_CTR0 through IA32_FIXED_CTR2). Each of the fixed-function PMC can count only one architectural performance event.
Configuring the fixed-function PMCs is done by writing to bit fields in the MSR (IA32_FIXED_CTR_CTRL) located at address 38DH. Unlike configuring performance events for general-purpose PMCs (IA32_PMCx) via UMASK field in (IA32_PERFECTSELx), configuring, programming IA32_FIXED_CTR_CTRL for fixed-function PMCs do not require any UMASK.
- **Simplified event programming** — Most frequent operation in programming performance events are enabling/disabling event counting and checking the status of counter overflows. Architectural performance event version 2 provides three architectural MSRs:
 - IA32_PERF_GLOBAL_CTRL allows software to enable/disable event counting of all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or any general-purpose PMCs via a single WRMSR.
 - IA32_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single RDMSR.
 - IA32_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single WRMSR.
- **PMI Overhead Mitigation** — Architectural performance monitoring version 2 introduces two bit field interface in IA32_DEBUGCTL for PMI service routine to accumulate performance monitoring data and LBR records with reduced perturbation from servicing the PMI. The two bit fields are:
 - IA32_DEBUGCTL.Freeze_LBR_On_PMI(bit 11). In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 18.4.7 for details of the legacy Freeze LBRs on PMI control.
 - IA32_DEBUGCTL.Freeze_PerfMon_On_PMI(bit 12). In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 18.4.7 for details of the legacy Freeze LBRs on PMI control.

The facilities provided by architectural performance monitoring version 2 can be queried from CPUID leaf 0AH by examining the content of register EDX:

- Bits 0 through 4 of CPUID.0AH.EDX indicates the number of fixed-function performance counters available per core,
- Bits 5 through 12 of CPUID.0AH.EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.

NOTE

Early generation of processors based on Intel Core microarchitecture may report in CPUID.0AH:EDX of support for version 2 but indicating incorrect information of version 2 facilities.

The IA32_FIXED_CTR_CTRL MSR include multiple sets of 4-bit field, each 4 bit field controls the operation of a fixed-function performance counter. Figure 20-2 shows the layout of 4-bit controls for each fixed-function PMC. Two sub-fields are currently defined within each control. The definitions of the bit fields are:

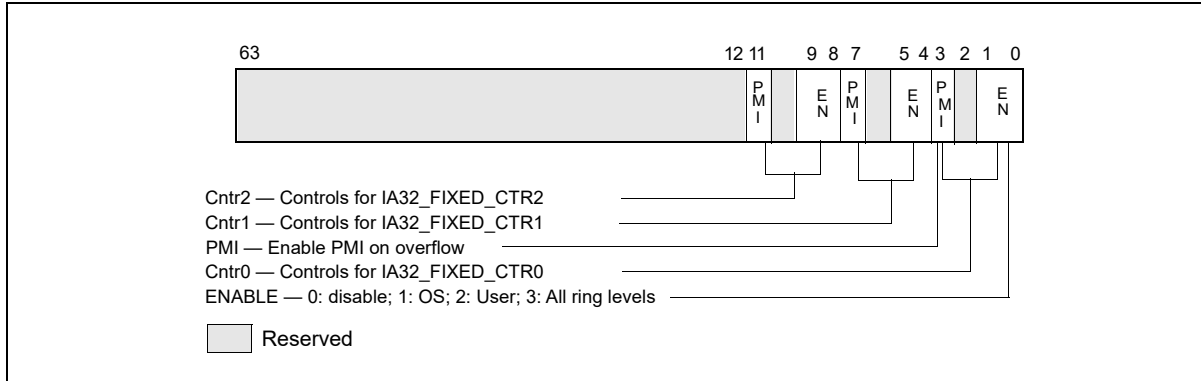


Figure 20-2. Layout of IA32_FIXED_CTR_CTRL MSR

- Enable field (lowest 2 bits within each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring 0. When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring greater than 0. Writing 0 to both bits stops the performance counter. Writing a value of 11B enables the counter to increment irrespective of privilege levels.
- PMI field (the fourth bit within each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

IA32_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting of each performance counter. Figure 20-3 shows the layout of IA32_PERF_GLOBAL_CTRL. Each enable bit in IA32_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

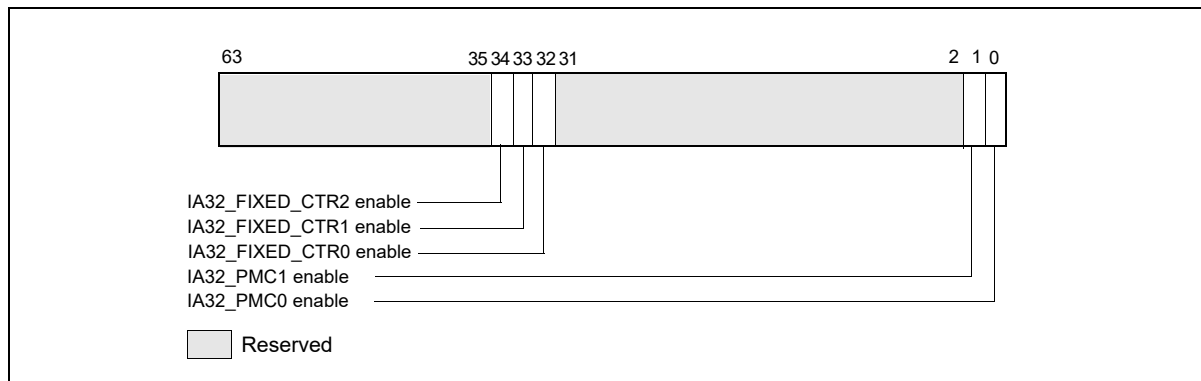


Figure 20-3. Layout of IA32_PERF_GLOBAL_CTRL MSR

The behavior of the fixed function performance counters supported by architectural performance version 2 is expected to be consistent on all processors that support those counters, and is defined as follows.

Table 20-2. Association of Fixed-Function Performance Counters with Architectural Performance Events

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_FIXED_CTR0	309H	INST_RETIRED.ANY	This event counts the number of instructions that retire execution. For instructions that consist of multiple uops, this event counts the retirement of the last uop of the instruction. The counter continues counting during hardware interrupts, traps, and in-side interrupt handlers.
IA32_FIXED_CTR1	30AH	CPU_CLK_UNHALTED.THREAD CPU_CLK_UNHALTED.CORE	The CPU_CLK_UNHALTED.THREAD event counts the number of core cycles while the logical processor is not in a halt state. If there is only one logical processor in a processor core, CPU_CLK_UNHALTED.CORE counts the unhalted cycles of the processor core. The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time.
IA32_FIXED_CTR2	30BH	CPU_CLK_UNHALTED.REF_TSC	This event counts the number of reference cycles at the TSC rate when the core is not in a halt state and not in a TM stop-clock state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction. This event is not affected by core frequency changes (e.g., P states) but counts at the same frequency as the time stamp counter. This event can approximate elapsed time while the core was not in a halt state and not in a TM stopclock state.
IA32_FIXED_CTR3	30CH	TOPDOWN.SLOTS	This event counts the number of available slots for an unhalted logical processor. The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core. Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

IA32_PERF_GLOBAL_STATUS MSR provides single-bit status for software to query the overflow condition of each performance counter. IA32_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. IA32_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware. Figure 20-4 shows the layout of IA32_PERF_GLOBAL_STATUS. A value of 1 in bits 0, 1, 32 through 34 indicates a counter overflow condition has occurred in the associated counter.

When a performance counter is configured for PEBS, overflow condition in the counter generates a performance-monitoring interrupt signaling a PEBS event. On a PEBS event, the processor stores data records into the buffer area (see Section 18.15.5), clears the counter overflow status, and sets the "OvfBuffer" bit in IA32_PERF_GLOBAL_STATUS.

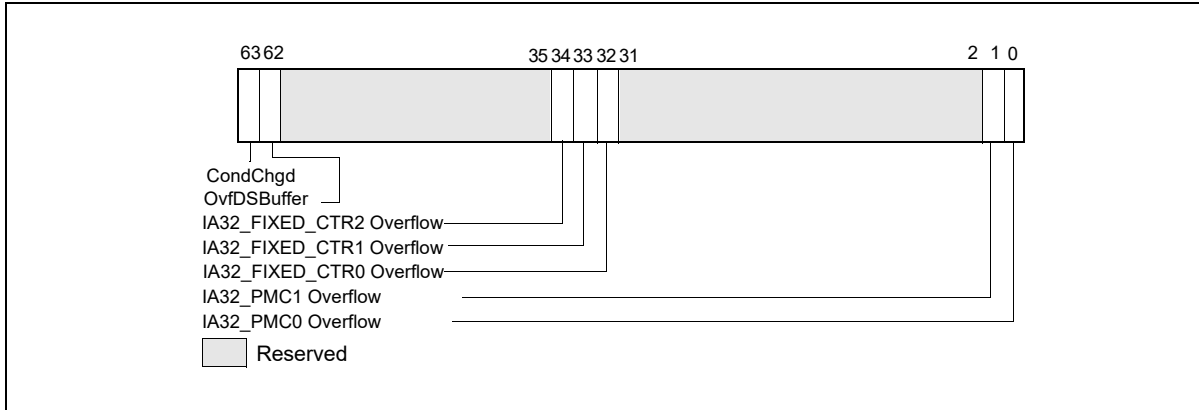


Figure 20-4. Layout of IA32_PERF_GLOBAL_STATUS MSR

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow indicator(s) of any general-purpose or fixed-function counters via a single WRMSR. Software should clear overflow indications when

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

The layout of IA32_PERF_GLOBAL_OVF_CTL is shown in Figure 20-5.

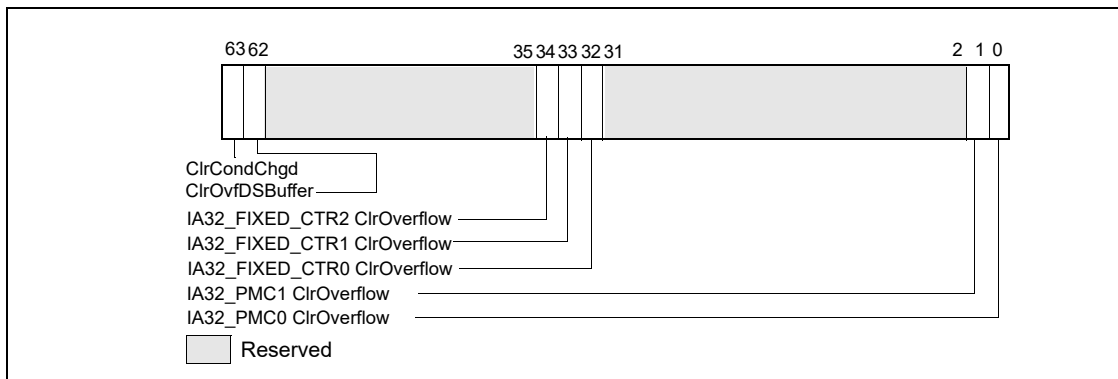


Figure 20-5. Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

20.2.3 Architectural Performance Monitoring Version 3

Processors supporting architectural performance monitoring version 3 also supports version 1 and 2, as well as capability enumerated by CPUID leaf 0AH. Specifically, version 3 provides the following enhancement in performance monitoring facilities if a processor core comprising of more than one logical processor, i.e., a processor core supporting Intel Hyper-Threading Technology or simultaneous multi-threading capability:

- AnyThread counting for processor core supporting two or more logical processors. The interface that supports AnyThread counting include:
 - Each IA32_PERFEVTSELx MSR (starting at MSR address 186H) support the bit field layout defined in Figure 20-6.

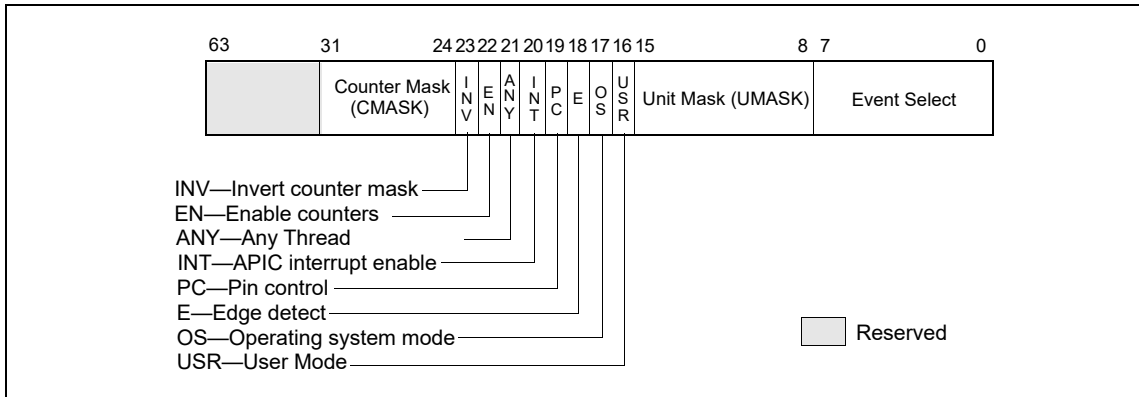


Figure 20-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3 for processor core comprising of two or more logical processors. When set to 1, it enables counting the associated event conditions (including matching the thread’s CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core. When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread’s CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.

- Each fixed-function performance counter IA32_FIXED_CTRx (starting at MSR address 309H) is configured by a 4-bit control block in the IA32_PERF_FIXED_CTR_CTRL MSR. The control block also allow thread-specificity configuration using an AnyThread bit. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown.

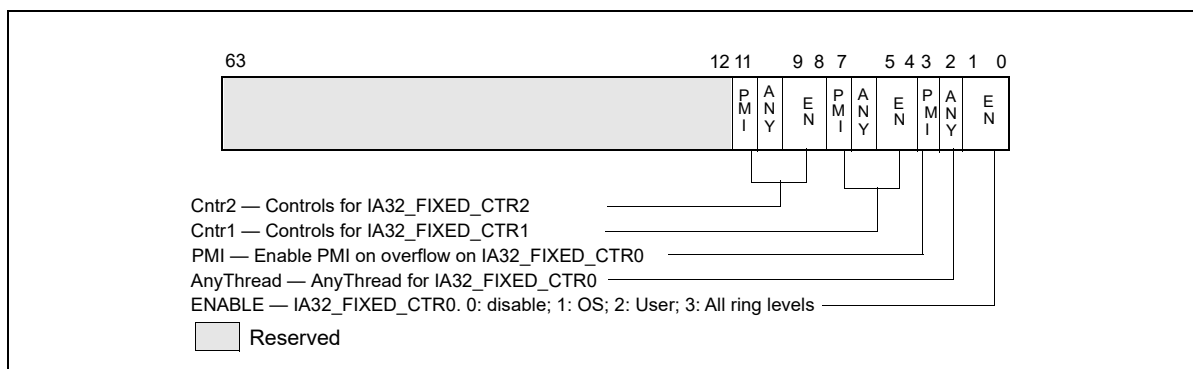


Figure 20-7. IA32_PERF_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

Each control block for a fixed-function performance counter provides an **AnyThread** (bit position 2 + 4*N, N= 0, 1, etc.) bit. When set to 1, it enables counting the associated event conditions (including matching the thread’s CPL with the ENABLE setting of the corresponding control block of IA32_PERF_FIXED_CTR_CTRL) occurring across all logical processors sharing a processor core. When an **AnyThread** bit is 0 in IA32_PERF_FIXED_CTR_CTRL, the corresponding fixed counter only increments the associated event conditions occurring in the logical processor which programmed the IA32_PERF_FIXED_CTR_CTRL MSR.

- The IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL MSRs provide single-bit controls/status for each general-purpose and fixed-function performance counter. Figure 20-8 and Figure 20-9 show the layout of these MSRs for N general-purpose performance counters (where N is reported by CPUID.0AH:EAX[15:8]) and three fixed-function counters.

20.2.3.1 AnyThread Counting and Software Evolution

The motivation for characterizing software workload over multiple software threads running on multiple logical processors of the same processor core originates from a time earlier than the introduction of the AnyThread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL. While AnyThread counting provides some benefits in simple software environments of an earlier era, the evolution contemporary software environments introduce certain concepts and pre-requisites that AnyThread counting does not comply with.

One example is the proliferation of software environments that support multiple virtual machines (VM) under VMX (see Chapter 24, “Introduction to Virtual-Machine Extensions”) where each VM represents a domain separated from one another.

A Virtual Machine Monitor (VMM) that manages the VMs may allow an individual VM to employ performance monitoring facilities to profiles the performance characteristics of a workload. The use of the Anythread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL is discouraged with software environments supporting virtualization or requiring domain separation.

Specifically, Intel recommends VMM:

- Configure the MSR bitmap to cause VM-exits for WRMSR to IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in VMX non-Root operation (see Chapter 25 for additional information),
- Clear the AnyThread bit of IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in the MSR-load lists for VM exits and VM entries (see Chapter 25, Chapter 27, and Chapter 28).

Even when operating in simpler legacy software environments which might not emphasize the pre-requisites of a virtualized software environment, the use of the AnyThread interface should be moderated and follow any event-specific guidance where explicitly noted.

20.2.4 Architectural Performance Monitoring Version 4

Processors supporting architectural performance monitoring version 4 also supports version 1, 2, and 3, as well as capability enumerated by CPUID leaf 0AH. Version 4 introduced a streamlined PMI overhead mitigation interface that replaces the legacy semantic behavior but retains the same control interface in IA32_DEBUGCTL.Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI. Specifically version 4 provides the following enhancements:

- New indicators (LBR_FRZ, CTR_FRZ) in IA32_PERF_GLOBAL_STATUS, see Section 20.2.4.1.
- Streamlined Freeze/PMI Overhead management interfaces to use IA32_DEBUGCTL.Freeze_LBRs_On_PMI and IA32_DEBUGCTL.Freeze_PerfMon_On_PMI: see Section 20.2.4.1. Legacy semantics of Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI (applicable to version 2 and 3) are not supported with version 4 or higher.
- Fine-grain separation of control interface to manage overflow/status of IA32_PERF_GLOBAL_STATUS and read-only performance counter enabling interface in IA32_PERF_GLOBAL_STATUS: see Section 20.2.4.2.
- Performance monitoring resource in-use MSR to facilitate cooperative sharing protocol between perfmon-managing privilege agents.

20.2.4.1 Enhancement in IA32_PERF_GLOBAL_STATUS

The IA32_PERF_GLOBAL_STATUS MSR provides the following indicators with architectural performance monitoring version 4:

- IA32_PERF_GLOBAL_STATUS.LBR_FRZ[bit 58]: This bit is set due to the following conditions:
 - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI has been set by the profiling agent, and
 - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently the LBR stack is frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.LBR_FRZ bit also serves as a control to enable capturing data in the LBR stack. To enable capturing LBR records, the following expression must hold with architectural perfmon version 4 or higher:

— $(\text{IA32_DEBUGCTL.LBR} \ \& \ (!\text{IA32_PERF_GLOBAL_STATUS.LBR_FRZ})) = 1$

- IA32_PERF_GLOBAL_STATUS.CTR_FRZ[bit 59]: This bit is set due to the following conditions:

- IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI has been set by the profiling agent, and
- A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently, all the performance counters are frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.CTR_FRZ bit also serve as an read-only control to enable programmable performance counters and fixed counters in the core PMU. To enable counting with the performance counters, the following expression must hold with architectural perfmon version 4 or higher:

- $(IA32_PERFEVTSELn.EN \& IA32_PERF_GLOBAL_CTRL.PMCn \& (!IA32_PERF_GLOBAL_STATUS.CTR_FRZ)) = 1$ for programmable counter 'n', or
- $(IA32_PERF_FIXED_CTRL.ENi \& IA32_PERF_GLOBAL_CTRL.FCi \& (!IA32_PERF_GLOBAL_STATUS.CTR_FRZ)) = 1$ for fixed counter 'i'

The read-only enable interface IA32_PERF_GLOBAL_STATUS.CTR_FRZ provides a more efficient flow for a PMI handler to use IA32_DEBUGCTL.Freeze_Perfmon_On_PMI to filter out data that may distort target workload analysis, see Table 18-3. It should be noted the IA32_PERF_GLOBAL_CTRL register continue to serve as the primary interface to control all performance counters of the logical processor.

For example, when the Freeze-On-PMI mode is not being used, a PMI handler would be setting IA32_PERF_GLOBAL_CTRL as the very last step to commence the overall operation after configuring the individual counter registers, controls, and PEBS facility. This does not only assure atomic monitoring but also avoids unnecessary complications (e.g., race conditions) when software attempts to change the core PMU configuration while some counters are kept enabled.

Additionally, IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55]: On processors that support Intel Processor Trace and configured to store trace output packets to physical memory using the ToPA scheme, bit 55 is set when a PMI occurred due to a ToPA entry memory buffer was completely filled.

IA32_PERF_GLOBAL_STATUS also provides an indicator to distinguish interaction of performance monitoring operations with other side-band activities, which apply Intel SGX on processors that support SGX (For additional information about Intel SGX, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D):

- IA32_PERF_GLOBAL_STATUS.ASCI[bit 60]: This bit is set when data accumulated in any of the configured performance counters (i.e., IA32_PMCx or IA32_FIXED_CTRx) may include contributions from direct or indirect operation of Intel SGX to protect an enclave (since the last time IA32_PERF_GLOBAL_STATUS.ASCI was cleared).

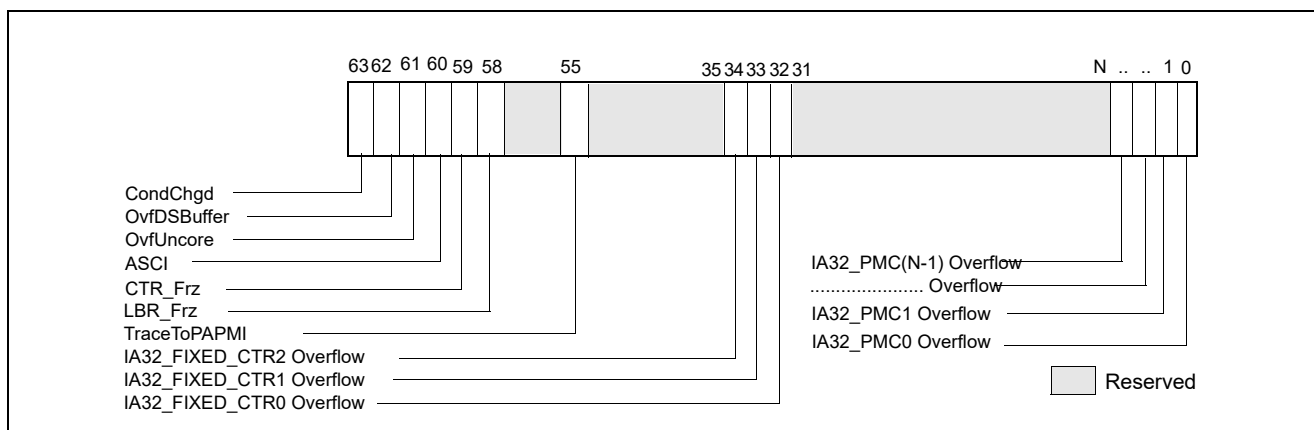


Figure 20-10. IA32_PERF_GLOBAL_STATUS MSR and Architectural Perfmon Version 4

Note, a processor's support for IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55] is enumerated as a result of CPUID enumerated capability of Intel Processor Trace and the use of the ToPA buffer scheme. Support of IA32_PERF_GLOBAL_STATUS.ASCI[bit 60] is enumerated by the CPUID enumeration of Intel SGX.

20.2.4.2 IA32_PERF_GLOBAL_STATUS_RESET and IA32_PERF_GLOBAL_STATUS_SET MSRS

With architectural performance monitoring version 3 and lower, clearing of the set bits in IA32_PERF_GLOBAL_STATUS MSR by software is done via IA32_PERF_GLOBAL_OVF_CTRL MSR. Starting with architectural performance monitoring version 4, software can manage the overflow and other indicators in IA32_PERF_GLOBAL_STATUS using separate interfaces to set or clear individual bits.

The address and the architecturally-defined bits of IA32_PERF_GLOBAL_OVF_CTRL is inherited by IA32_PERF_GLOBAL_STATUS_RESET (see Figure 20-11). Further, IA32_PERF_GLOBAL_STATUS_RESET provides additional bit fields to clear the new indicators in IA32_PERF_GLOBAL_STATUS described in Section 20.2.4.1.

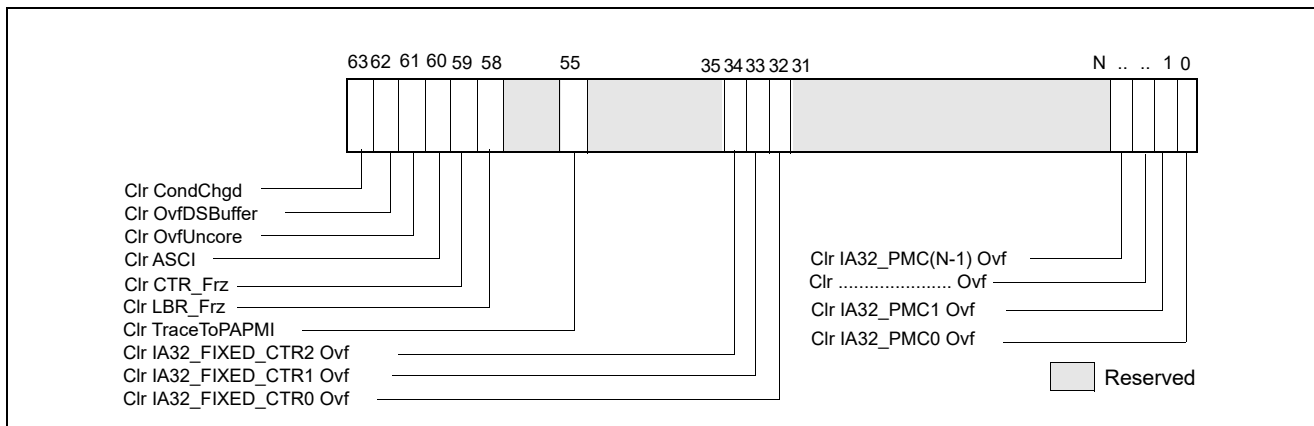


Figure 20-11. IA32_PERF_GLOBAL_STATUS_RESET MSR and Architectural Perfmon Version 4

The IA32_PERF_GLOBAL_STATUS_SET MSR is introduced with architectural performance monitoring version 4. It allows software to set individual bits in IA32_PERF_GLOBAL_STATUS. The IA32_PERF_GLOBAL_STATUS_SET interface can be used by a VMM to virtualize the state of IA32_PERF_GLOBAL_STATUS across VMs.

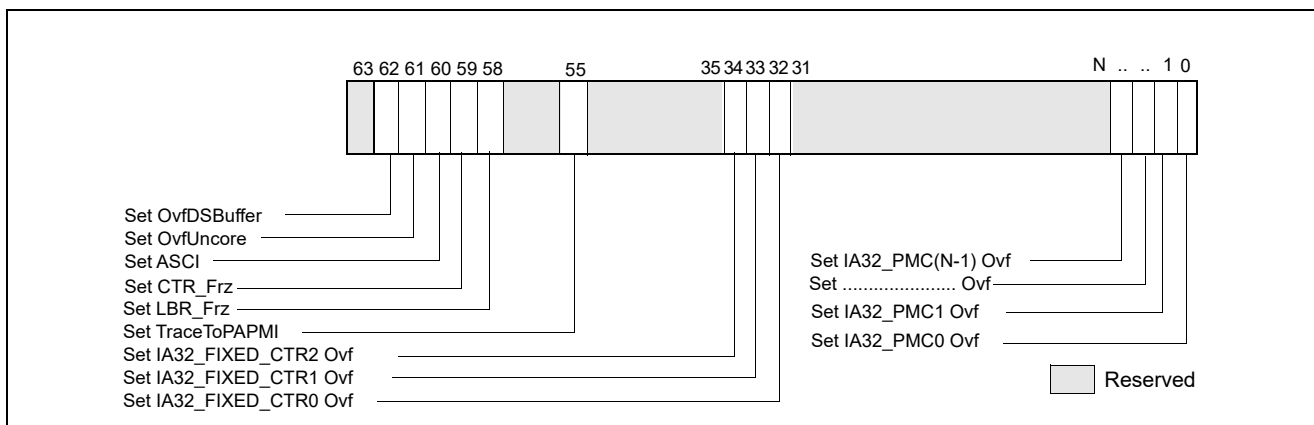


Figure 20-12. IA32_PERF_GLOBAL_STATUS_SET MSR and Architectural Perfmon Version 4

20.2.4.3 IA32_PERF_GLOBAL_INUSE MSR

In a contemporary software environment, multiple privileged service agents may wish to employ the processor’s performance monitoring facilities. The IA32_MISC_ENABLE.PERFMON_AVAILABLE[bit 7] interface could not serve

the need of multiple agent adequately. A white paper, “Performance Monitoring Unit Sharing Guideline”¹, proposed a cooperative sharing protocol that is voluntary for participating software agents.

Architectural performance monitoring version 4 introduces a new MSR, IA32_PERF_GLOBAL_INUSE, that simplifies the task of multiple cooperating agents to implement the sharing protocol.

The layout of IA32_PERF_GLOBAL_INUSE is shown in Figure 20-13.

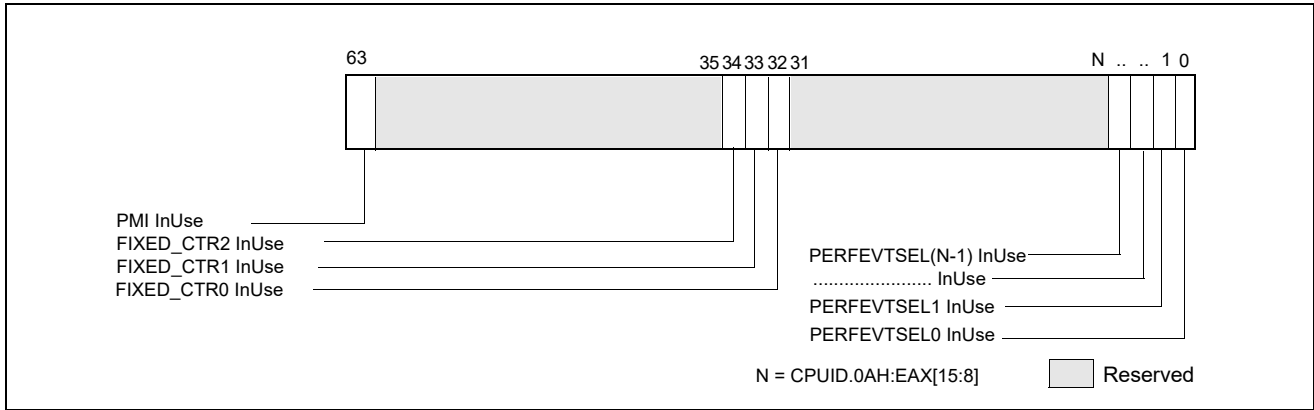


Figure 20-13. IA32_PERF_GLOBAL_INUSE MSR and Architectural Perfmon Version 4

The IA32_PERF_GLOBAL_INUSE MSR provides an “InUse” bit for each programmable performance counter and fixed counter in the processor. Additionally, it includes an indicator if the PMI mechanism has been configured by a profiling agent.

- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL0_InUse[bit 0]: This bit reflects the logical state of (IA32_PERFEVTSEL0[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL1_InUse[bit 1]: This bit reflects the logical state of (IA32_PERFEVTSEL1[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL2_InUse[bit 2]: This bit reflects the logical state of (IA32_PERFEVTSEL2[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSELn_InUse[bit n]: This bit reflects the logical state of (IA32_PERFEVTSELn[7:0] != 0), n < CPUID.0AH:EAX[15:8].
- IA32_PERF_GLOBAL_INUSE.FC0_InUse[bit 32]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[1:0] != 0).
- IA32_PERF_GLOBAL_INUSE.FC1_InUse[bit 33]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[5:4] != 0).
- IA32_PERF_GLOBAL_INUSE.FC2_InUse[bit 34]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[9:8] != 0).
- IA32_PERF_GLOBAL_INUSE.PMI_InUse[bit 63]: This bit is set if any one of the following bit is set:
 - IA32_PERFEVTSELn.INT[bit 20], n < CPUID.0AH:EAX[15:8].
 - IA32_FIXED_CTR_CTRL.ENi_PMI, i = 0, 1, 2.
 - Any IA32_PEBS_ENABLES bit which enables PEBS for a general-purpose or fixed-function performance counter.

1. Available at <http://www.intel.com/sdm>

20.2.5 Architectural Performance Monitoring Version 5

Processors supporting architectural performance monitoring version 5 also support versions 1, 2, 3, and 4, as well as capability enumerated by CPUID leaf 0AH. Specifically, version 5 provides the following enhancements:

- Deprecation of AnyThread mode, see Section 20.2.5.1.
- Individual enumeration of Fixed counters in CPUID.0AH, see Section 20.2.5.2.
- Domain separation, see Section 20.2.5.3.

20.2.5.1 AnyThread Mode Deprecation

With Architectural Performance Monitoring Version 5, a processor that supports AnyThread mode deprecation is enumerated by CPUID.0AH.EDX[15]. If set, software will not have to follow guidelines in Section 20.2.3.1.

20.2.5.2 Fixed Counter Enumeration

With Architectural Performance Monitoring Version 5, register CPUID.0AH.ECX indicates Fixed Counter enumeration. It is a bit mask which enumerates the supported Fixed Counters in a processor. If bit 'i' is set, it implies that Fixed Counter 'i' is supported. Software is recommended to use the following logic to check if a Fixed Counter is supported on a given processor:

```
FxCtr[i]_is_supported := ECX[i] || (EDX[4:0] > i);
```

20.2.5.3 Domain Separation

When the INV flag in IA32_PERFEVTSELx is used, a counter stops counting when the logical processor exits the C0 ACPI C-state.

20.2.6 Full-Width Writes to Performance Counter Registers

The general-purpose performance counter registers IA32_PMCx are writable via WRMSR instruction. However, the value written into IA32_PMCx by WRMSR is the signed extended 64-bit value of the EAX[31:0] input of WRMSR.

A processor that supports full-width writes to the general-purpose performance counters enumerated by CPUID.0AH:EAX[15:8] will set IA32_PERF_CAPABILITIES[13] to enumerate its full-width-write capability. See Figure 20-65.

If IA32_PERF_CAPABILITIES.FW_WRITE[bit 13] = 1, each IA32_PMCi is accompanied by a corresponding alias address starting at 4C1H for IA32_A_PMC0.

The bit width of the performance monitoring counters is specified in CPUID.0AH:EAX[23:16].

If IA32_A_PMCi is present, the 64-bit input value (EDX:EAX) of WRMSR to IA32_A_PMCi will cause IA32_PMCi to be updated by:

```
COUNTERWIDTH = CPUID.0AH:EAX[23:16] bit width of the performance monitoring counter
IA32_PMCi[COUNTERWIDTH-1:32] := EDX[COUNTERWIDTH-33:0];
IA32_PMCi[31:0] := EAX[31:0];
EDX[63:COUNTERWIDTH] are reserved
```

20.3 PERFORMANCE MONITORING (INTEL® CORE™ PROCESSORS AND INTEL® XEON® PROCESSORS)

20.3.1 Performance Monitoring for Processors Based on Nehalem Microarchitecture

Intel Core i7 processor family¹ supports architectural performance monitoring capability with version ID 3 (see Section 20.2.3) and a host of non-architectural monitoring capabilities. The Intel Core i7 processor family is based

on Nehalem microarchitecture, and provides four general-purpose performance counters (IA32_PMC0, IA32_PMC1, IA32_PMC2, IA32_PMC3) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2) in the processor core.

Non-architectural performance monitoring in Intel Core i7 processor family uses the IA32_PERFVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>. Non-architectural performance monitoring events fall into two broad categories:

- Performance monitoring events in the processor core: These include many events that are similar to performance monitoring events available to processor based on Intel Core microarchitecture. Additionally, there are several enhancements in the performance monitoring capability for detecting microarchitectural conditions in the processor core or in the interaction of the processor core to the off-core sub-systems in the physical processor package. The off-core sub-systems in the physical processor package is loosely referred to as “uncore”.
- Performance monitoring events in the uncore: The uncore sub-system is shared by more than one processor cores in the physical processor package. It provides additional performance monitoring facility outside of IA32_PMCx and performance monitoring events that are specific to the uncore sub-system.

Architectural and non-architectural performance monitoring events in Intel Core i7 processor family support thread qualification using bit 21 of IA32_PERFVTSELx MSR.

The bit fields within each IA32_PERFVTSELx MSR are defined in Figure 20-6 and described in Section 20.2.1.1 and Section 20.2.3.

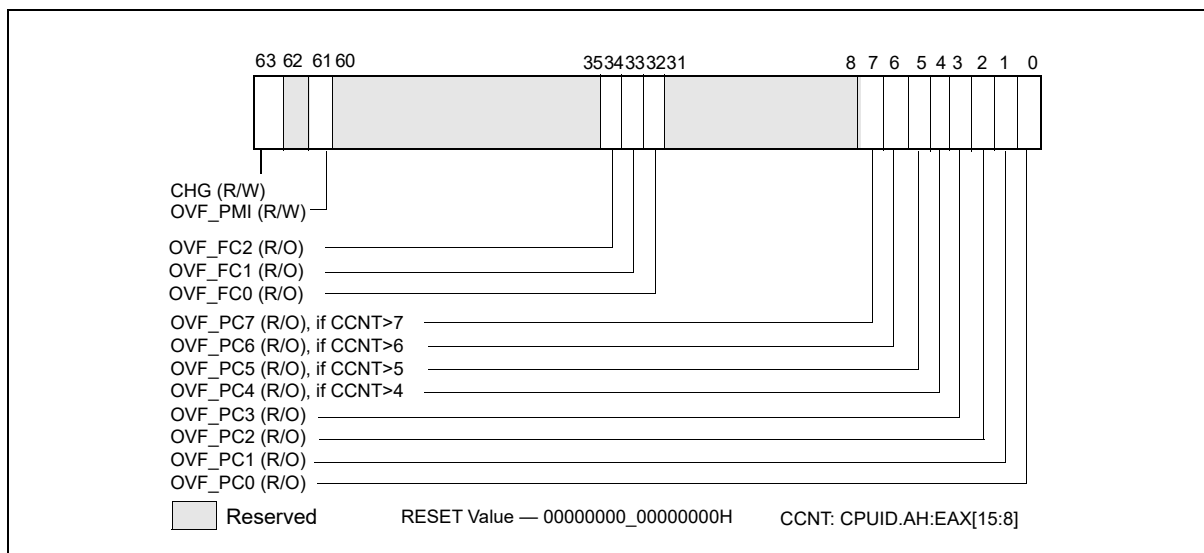


Figure 20-14. IA32_PERF_GLOBAL_STATUS MSR

20.3.1.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- Four general purpose performance counters, IA32_PMCx, associated counter configuration MSRs, IA32_PERFVTSELx, and global counter control MSR supporting simplified control of four counters. Each of the four performance counter can support processor event based sampling (PEBS) and thread-qualification of architectural and non-architectural performance events. Width of IA32_PMCx supported by hardware has been increased. The width of counter reported by CPUID.0AH:EAX[23:16] is 48 bits. The PEBS facility in Nehalem

1. Intel Xeon processor 5500 series and 3400 series are also based on Nehalem microarchitecture; the performance monitoring facilities described in this section generally also apply.

microarchitecture has been enhanced to include new data format to capture additional information, such as load latency.

- Load latency sampling facility. Average latency of memory load operation can be sampled using load-latency facility in processors based on Nehalem microarchitecture. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches). This facility is used in conjunction with the PEBS facility.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters. CPUID.0AH:EAX[15:8] reports the MSRs available to software; see Section 20.2.1.

20.3.1.1.1 Processor Event Based Sampling (PEBS)

All general-purpose performance counters, IA32_PMCx, can be used for PEBS if the performance event supports PEBS. Software uses IA32_MISC_ENABLE[7] and IA32_MISC_ENABLE[12] to detect whether the performance monitoring facility and PEBS functionality are supported in the processor. The MSR IA32_PEBS_ENABLE provides 4 bits that software must use to enable which IA32_PMCx overflow condition will cause the PEBS record to be captured.

Additionally, the PEBS record is expanded to allow latency information to be captured. The MSR IA32_PEBS_ENABLE provides 4 additional bits that software must use to enable latency data recording in the PEBS record upon the respective IA32_PMCx overflow condition. The layout of IA32_PEBS_ENABLE for processors based on Nehalem microarchitecture is shown in Figure 20-15.

When a counter is enabled to capture machine state (PEBS_EN_PMCx = 1), the processor will write machine state information to a memory buffer specified by software as detailed below. When the counter IA32_PMCx overflows from maximum count to zero, the PEBS hardware is armed.

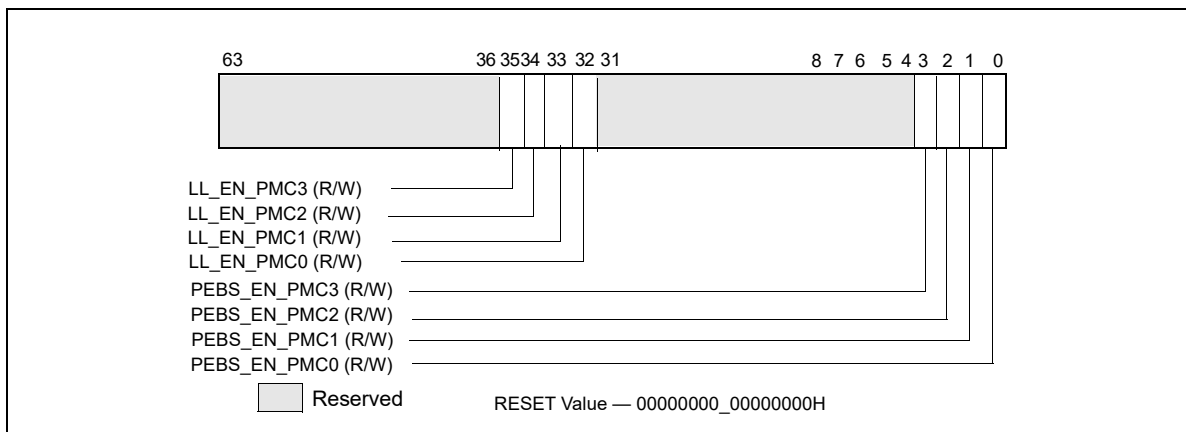


Figure 20-15. Layout of IA32_PEBS_ENABLE MSR

Upon occurrence of the next PEBS event, the PEBS hardware triggers an assist and causes a PEBS record to be written. The format of the PEBS record is indicated by the bit field IA32_PERF_CAPABILITIES[11:8] (see Figure 20-65).

The behavior of PEBS assists is reported by IA32_PERF_CAPABILITIES[6] (see Figure 20-65). The return instruction pointer (RIP) reported in the PEBS record will point to the instruction after (+1) the instruction that causes the PEBS assist. The machine state reported in the PEBS record is the machine state after the instruction that causes the PEBS assist is retired. For instance, if the instructions:

```
mov eax, [eax] ; causes PEBS assist
```

```
nop
```

are executed, the PEBS record will report the address of the nop, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation.

The PEBS record format is shown in Table 20-3, and each field in the PEBS record is 64 bits long. The PEBS record format, along with debug/store area storage format, does not change regardless of IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 20-3. PEBS Record Format for Intel Core i7 Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	58H	R9
08H	R/EIP	60H	R10
10H	R/EAX	68H	R11
18H	R/EBX	70H	R12
20H	R/ECX	78H	R13
28H	R/EDX	80H	R14
30H	R/ESI	88H	R15
38H	R/EDI	90H	IA32_PERF_GLOBAL_STATUS
40H	R/EBP	98H	Data Linear Address
48H	R/ESP	A0H	Data Source Encoding
50H	R8	A8H	Latency value (core cycles)

In IA-32e mode, the full 64-bit value is written to the register. If the processor is not operating in IA-32e mode, 32-bit value is written to registers with bits 63:32 zeroed. Registers not defined when the processor is not in IA-32e mode are written to zero.

Bytes AFH:90H are enhancement to the PEBS record format. Support for this enhanced PEBS record format is indicated by IA32_PERF_CAPABILITIES[11:8] encoding of 0001B.

The value written to bytes 97H:90H is the state of the IA32_PERF_GLOBAL_STATUS register before the PEBS assist occurred. This value is written so software can determine which counters overflowed when this PEBS record was written. Note that this field indicates the overflow status for all counters, regardless of whether they were programmed for PEBS or not.

Programming PEBS Facility

Only a subset of non-architectural performance events in the processor support PEBS. The subset of precise events are listed in Table 20-84. In addition to using IA32_PERFEVTSELx to specify event unit/mask settings and setting the EN_PMCx bit in the IA32_PEBS_ENABLE register for the respective counter, the software must also initialize the DS_BUFFER_MANAGEMENT_AREA data structure in memory to support capturing PEBS records for precise events.

The recording of PEBS records may not operate properly if accesses to the linear addresses in the DS buffer management area or in the PEBS buffer (see below) cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do "lazy" page-table entry propagation for these pages. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

The beginning linear address of the DS_BUFFER_MANAGEMENT_AREA data structure must be programmed into the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 20-16.

- **PEBS Buffer Base:** This field is programmed with the linear address of the first byte of the PEBS buffer allocated by software. The processor reads this field to determine the base address of the PEBS buffer.
- **PEBS Index:** This field is initially programmed with the same value as the PEBS Buffer Base field, or the beginning linear address of the PEBS buffer. The processor reads this field to determine the location of the next PEBS record to write to. After a PEBS record has been written, the processor also updates this field with the address of the next PEBS record to be written. The figure above illustrates the state of PEBS Index after the first PEBS record is written.
- **PEBS Absolute Maximum:** This field represents the absolute address of the maximum length of the allocated PEBS buffer plus the starting address of the PEBS buffer. The processor will not write any PEBS record beyond the end of PEBS buffer, when **PEBS Index** equals **PEBS Absolute Maximum**. No signaling is generated when PEBS buffer is full. Software must reset the **PEBS Index** field to the beginning of the PEBS buffer address to continue capturing PEBS records.

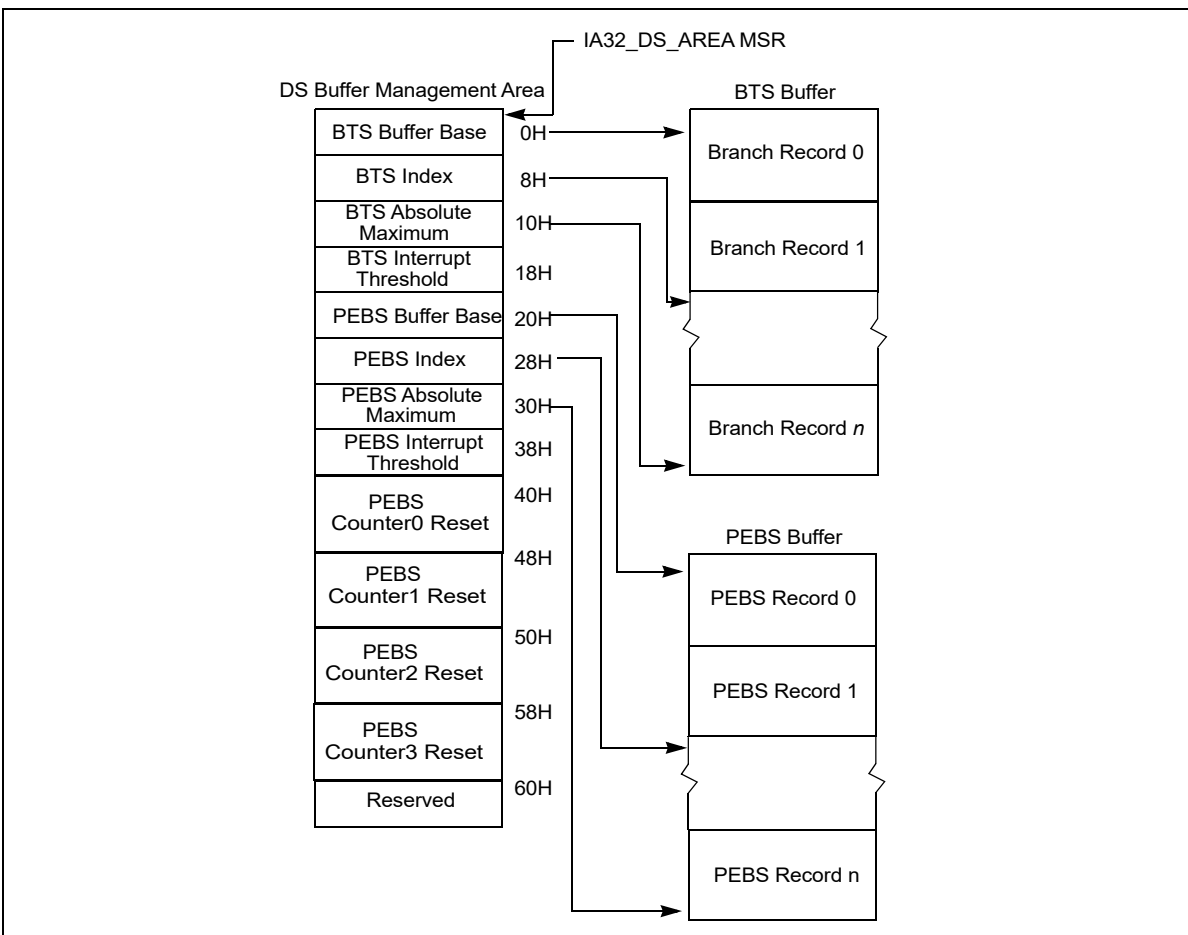


Figure 20-16. PEBS Programming Environment

- **PEBS Interrupt Threshold:** This field specifies the threshold value to trigger a performance interrupt and notify software that the PEBS buffer is nearly full. This field is programmed with the linear address of the first byte of the PEBS record within the PEBS buffer that represents the threshold record. After the processor writes a PEBS record and updates **PEBS Index**, if the **PEBS Index** reaches the threshold value of this field, the processor will generate a performance interrupt. This is the same interrupt that is generated by a performance counter overflow, as programmed in the Performance Monitoring Counters vector in the Local Vector Table of the Local APIC. When a performance interrupt due to PEBS buffer full is generated, the IA32_PERF_-GLOBAL_STATUS.PEBS_Ovf bit will be set.
- **PEBS CounterX Reset:** This field allows software to set up PEBS counter overflow condition to occur at a rate useful for profiling workload, thereby generating multiple PEBS records to facilitate characterizing the profile the execution of test code. After each PEBS record is written, the processor checks each counter to see if it overflowed and was enabled for PEBS (the corresponding bit in IA32_PEBS_ENABLED was set). If these conditions are met, then the reset value for each overflowed counter is loaded from the DS Buffer Management Area. For example, if counter IA32_PMC0 caused a PEBS record to be written, then the value of "PEBS Counter 0 Reset" would be written to counter IA32_PMC0. If a counter is not enabled for PEBS, its value will not be modified by the PEBS assist.

Performance Counter Prioritization

Performance monitoring interrupts are triggered by a counter transitioning from maximum count to zero (assuming IA32_PerfEvtSelX.INT is set). This same transition will cause PEBS hardware to arm, but not trigger. PEBS hardware triggers upon detection of the first PEBS event after the PEBS hardware has been armed (a 0 to 1 transition of the counter). At this point, a PEBS assist will be undertaken by the processor.

Performance counters (fixed and general-purpose) are prioritized in index order. That is, counter IA32_PMC0 takes precedence over all other counters. Counter IA32_PMC1 takes precedence over counters IA32_PMC2 and IA32_PMC3, and so on. This means that if simultaneous overflows or PEBS assists occur, the appropriate action will be taken for the highest priority performance counter. For example, if IA32_PMC1 cause an overflow interrupt and IA32_PMC2 causes a PEBS assist simultaneously, then the overflow interrupt will be serviced first.

The PEBS threshold interrupt is triggered by the PEBS assist, and is by definition prioritized lower than the PEBS assist. Hardware will not generate separate interrupts for each counter that simultaneously overflows. General-purpose performance counters are prioritized over fixed counters.

If a counter is programmed with a precise (PEBS-enabled) event and programmed to generate a counter overflow interrupt, the PEBS assist is serviced before the counter overflow interrupt is serviced. If in addition the PEBS interrupt threshold is met, the

threshold interrupt is generated after the PEBS assist completes, followed by the counter overflow interrupt (two separate interrupts are generated).

Uncore counters may be programmed to interrupt one or more processor cores (see Section 20.3.1.2). It is possible for interrupts posted from the uncore facility to occur coincident with counter overflow interrupts from the processor core. Software must check core and uncore status registers to determine the exact origin of counter overflow interrupts.

20.3.1.1.2 Load Latency Performance Monitoring Facility

The load latency facility provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table 20-3. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_INST_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 100H). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.

- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001_00000001H.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The load-latency information written into a PEBS record (see Table 20-3, bytes AFH:98H) consists of:

- **Data Linear Address:** This is the linear address of the target of the load operation.
- **Latency Value:** This is the elapsed cycles of the tagged load operation between dispatch to GO, measured in processor core clock domain.
- **Data Source:** The encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 20-4. In the descriptions, local memory refers to system memory physically attached to a processor package, and remote memory refers to system memory physically attached to another processor package.

Table 20-4. Data Source Encoding for Load Latency Record

Encoding	Description
00H	Unknown L3 cache miss.
01H	Minimal latency core cache hit. This request was satisfied by the L1 data cache.
02H	Pending core cache HIT. Outstanding core cache miss to same cache-line address was already underway.
03H	This data request was satisfied by the L2.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found. (clean).
06H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found.
07H ¹	Reserved/LLC Snoop HitM. Local or Remote home requests that hit the last level cache and were serviced by another core with a cross core snoop where modified copies were found.
08H	Reserved/L3 MISS. Local homed requests that missed the L3 cache and were serviced by forwarded data following a cross package snoop where no modified copies were found. (Remote home requests are not counted).
09H	Reserved
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to shared state).
0CH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to exclusive state).
0DH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to exclusive state).
0EH	I/O, Request of input/output operation.
0FH	The request was to un-cacheable memory.

NOTES:

1. Bit 7 is supported only for processors with a CPUID DisplayFamily_DisplayModel signature of 06_2A, and 06_2E; otherwise it is reserved.

The layout of MSR_PEBS_LD_LAT_THRESHOLD is shown in Figure 20-17.

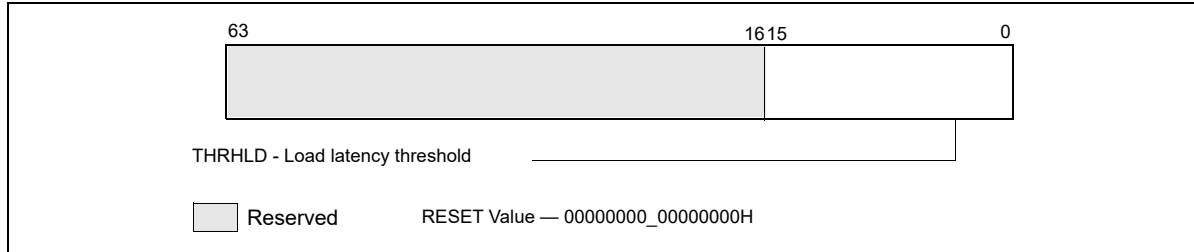


Figure 20-17. Layout of MSR_PEBS_LD_LAT MSR

Bits 15:0 specifies the threshold load latency in core clock cycles. Performance events with latencies greater than this value are counted in IA32_PMCx and their latency information is reported in the PEBS record. Otherwise, they are ignored. The minimum value that may be programmed in this field is 3.

20.3.1.1.3 Off-core Response Performance Monitoring in the Processor Core

Programming a performance event using the off-core response facility can choose any of the four IA32_PERFEVTSELx MSR with specific event codes and predefine mask bit value. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_0. There is only one off-core response configuration MSR. Table 20-5 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 20-5. Off-Core Response Event Encoding

Event code in IA32_PERFEVTSELx	Mask Value in IA32_PERFEVTSELx	Required Off-core Response MSR
B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)

The layout of MSR_OFFCORE_RSP_0 is shown in Figure 20-18. Bits 7:0 specifies the request type of a transaction request to the uncore. Bits 15:8 specifies the response of the uncore subsystem.

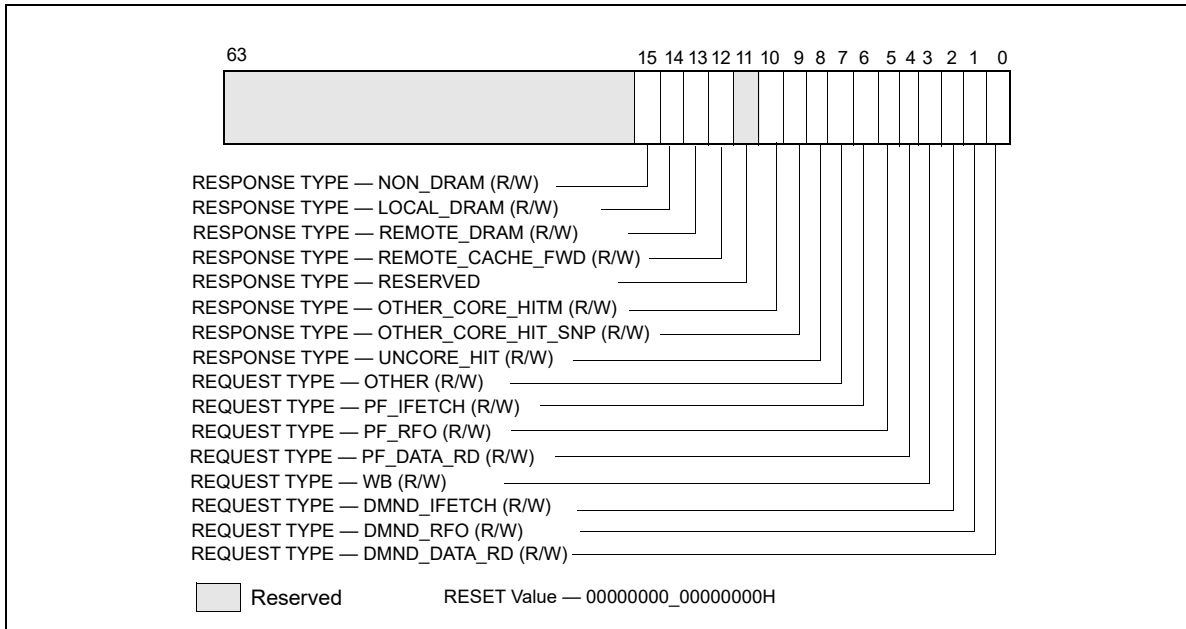


Figure 20-18. Layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 to Configure Off-core Response Events

Table 20-6. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
OTHER	7	Counts one of the following transaction types, including L3 invalidate, I/O, full or partial writes, WC or non-temporal stores, CLFLUSH, Fences, lock, unlock, split lock.
UNCORE_HIT	8	L3 Hit: local or remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
OTHER_CORE_HI T_SNP	9	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where no modified copies were found (clean).
OTHER_CORE_HI TM	10	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where modified copies were found (HITM).
Reserved	11	Reserved
REMOTE_CACHE_ FWD	12	L3 Miss: local homed requests that missed the L3 cache and was serviced by forwarded data following a cross package snoop where no modified copies found. (Remote home requests are not counted)
REMOTE_DRAM	13	L3 Miss: remote home requests that missed the L3 cache and were serviced by remote DRAM.
LOCAL_DRAM	14	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.

Table 20-6. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition (Contd.)

Bit Name	Offset	Description
NON_DRAM	15	Non-DRAM requests that were serviced by IOH.

20.3.1.2 Performance Monitoring Facility in the Uncore

The “uncore” in Nehalem microarchitecture refers to subsystems in the physical processor package that are shared by multiple processor cores. Some of the sub-systems in the uncore include the L3 cache, Intel QuickPath Interconnect link logic, and integrated memory controller. The performance monitoring facilities inside the uncore operates in the same clock domain as the uncore (U-clock domain), which is usually different from the processor core clock domain. The uncore performance monitoring facilities described in this section apply to Intel Xeon processor 5500 series and processors with the following CPUID signatures: 06_1AH, 06_1EH, 06_1FH (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4). An overview of the uncore performance monitoring facilities is described separately.

The performance monitoring facilities available in the U-clock domain consist of:

- Eight General-purpose counters (MSR_UNCORE_PerfCntr0 through MSR_UNCORE_PerfCntr7). The counters are 48 bits wide. Each counter is associated with a configuration MSR, MSR_UNCORE_PerfEvtSelx, to specify event code, event mask and other event qualification fields. A set of global uncore performance counter enabling/overflow/status control MSRs are also provided for software.
- Performance monitoring in the uncore provides an address/opcode match MSR that provides event qualification control based on address value or QPI command opcode.
- One fixed-function counter, MSR_UNCORE_FixedCntr0. The fixed-function uncore counter increments at the rate of the U-clock when enabled.

The frequency of the uncore clock domain can be determined from the uncore clock ratio which is available in the PCI configuration space register at offset C0H under device number 0 and Function 0.

20.3.1.2.1 Uncore Performance Monitoring Management Facility

MSR_UNCORE_PERF_GLOBAL_CTRL provides bit fields to enable/disable general-purpose and fixed-function counters in the uncore. Figure 20-19 shows the layout of MSR_UNCORE_PERF_GLOBAL_CTRL for an uncore that is shared by four processor cores in a physical package.

- EN_PCn (bit n, n = 0, 7): When set, enables counting for the general-purpose uncore counter MSR_UNCORE_PerfCntr n.
- EN_FC0 (bit 32): When set, enables counting for the fixed-function uncore counter MSR_UNCORE_FixedCntr0.
- EN_PMI_COREn (bit n, n = 0, 3 if four cores are present): When set, processor core n is programmed to receive an interrupt signal from any interrupt enabled uncore counter. PMI delivery due to an uncore counter overflow is enabled by setting IA32_DEBUGCTL.Offcore_PMI_EN to 1.
- PMI_FRZ (bit 63): When set, all U-clock uncore counters are disabled when any one of them signals a performance interrupt. Software must explicitly re-enable the counter by setting the enable bits in MSR_UNCORE_PERF_GLOBAL_CTRL upon exit from the ISR.

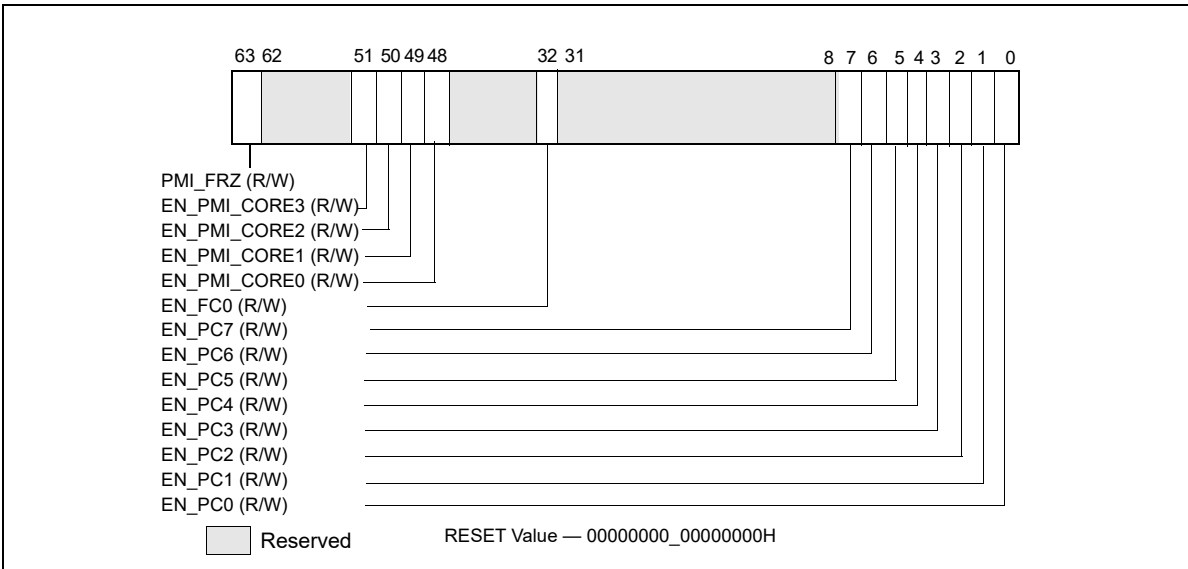


Figure 20-19. Layout of MSR_UNCORE_PERF_GLOBAL_CTRL MSR

MSR_UNCORE_PERF_GLOBAL_STATUS provides overflow status of the U-clock performance counters in the uncore. This is a read-only register. If an overflow status bit is set the corresponding counter has overflowed. The register provides a condition change bit (bit 63) which can be quickly checked by software to determine if a significant change has occurred since the last time the condition change status was cleared. Figure 20-20 shows the layout of MSR_UNCORE_PERF_GLOBAL_STATUS.

- OVF_PCn (bit n, n = 0, 7): When set, indicates general-purpose uncore counter MSR_UNCORE_PerfCntr n has overflowed.
- OVF_FC0 (bit 32): When set, indicates the fixed-function uncore counter MSR_UNCORE_FixedCntr0 has overflowed.
- OVF_PMI (bit 61): When set indicates that an uncore counter overflowed and generated an interrupt request.
- CHG (bit 63): When set indicates that at least one status bit in MSR_UNCORE_PERF_GLOBAL_STATUS register has changed state.

MSR_UNCORE_PERF_GLOBAL_OVF_CTRL allows software to clear the status bits in the UNCORE_PERF_GLOBAL_STATUS register. This is a write-only register, and individual status bits in the global status register are cleared by writing a binary one to the corresponding bit in this register. Writing zero to any bit position in this register has no effect on the uncore PMU hardware.

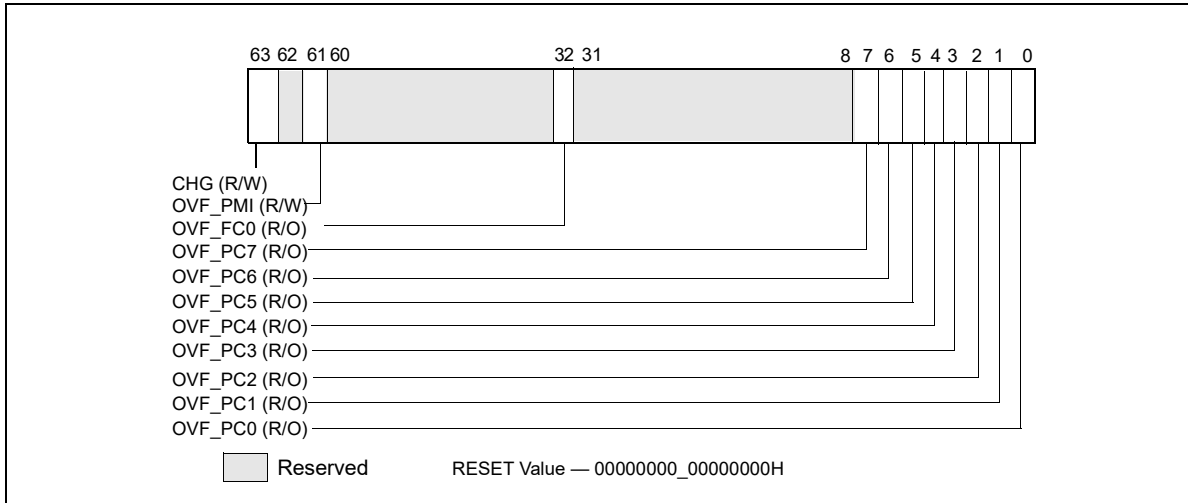


Figure 20-20. Layout of MSR_UNCORE_PERF_GLOBAL_STATUS MSR

Figure 20-21 shows the layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL.

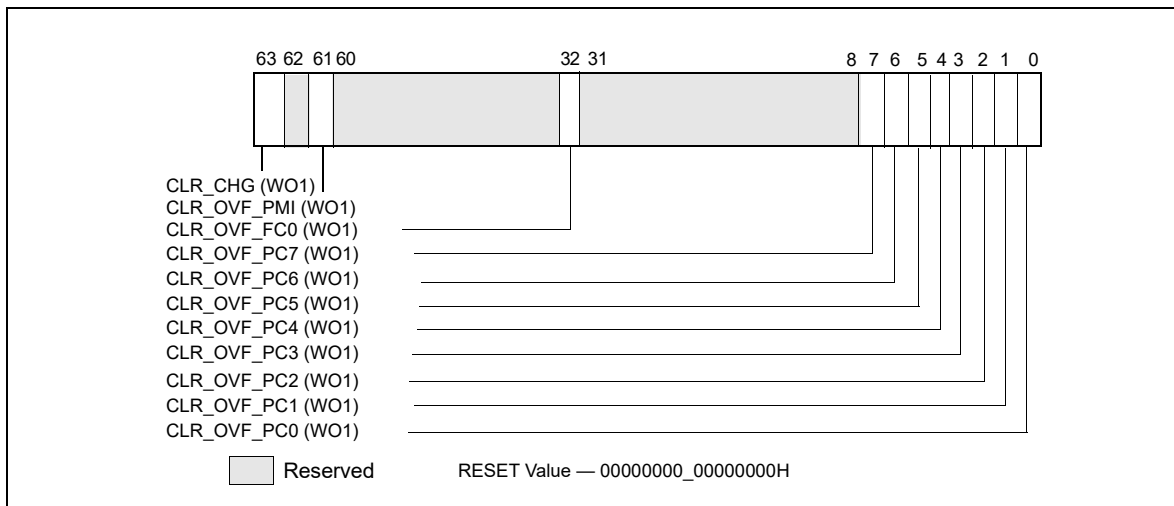


Figure 20-21. Layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL MSR

- CLR_OVF_PCn (bit n, n = 0, 7): Set this bit to clear the overflow status for general-purpose uncore counter MSR_UNCORE_PerfCntr n. Writing a value other than 1 is ignored.
- CLR_OVF_FC0 (bit 32): Set this bit to clear the overflow status for the fixed-function uncore counter MSR_UNCORE_FixedCntr0. Writing a value other than 1 is ignored.
- CLR_OVF_PMI (bit 61): Set this bit to clear the OVF_PMI flag in MSR_UNCORE_PERF_GLOBAL_STATUS. Writing a value other than 1 is ignored.
- CLR_CHG (bit 63): Set this bit to clear the CHG flag in MSR_UNCORE_PERF_GLOBAL_STATUS register. Writing a value other than 1 is ignored.

20.3.1.2.2 Uncore Performance Event Configuration Facility

MSR_UNCORE_PerfEvtSel0 through MSR_UNCORE_PerfEvtSel7 are used to select performance event and configure the counting behavior of the respective uncore performance counter. Each uncore PerfEvtSel MSR is paired with an uncore performance counter. Each uncore counter must be locally configured using the corre-

sponding MSR_UNCORE_PerEvtSelx and counting must be enabled using the respective EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL. Figure 20-22 shows the layout of MSR_UNCORE_PERFEVTSELx.

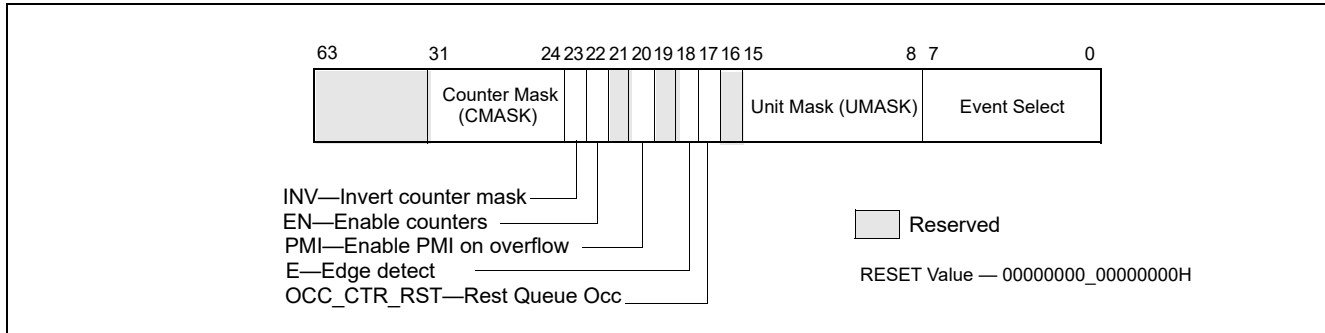


Figure 20-22. Layout of MSR_UNCORE_PERFEVTSELx MSRs

- Event Select (bits 7:0): Selects the event logic unit used to detect uncore events.
- Unit Mask (bits 15:8) : Condition qualifiers for the event selection logic specified in the Event Select field.
- OCC_CTR_RST (bit17): When set causes the queue occupancy counter associated with this event to be cleared (zeroed). Writing a zero to this bit will be ignored. It will always read as a zero.
- Edge Detect (bit 18): When set causes the counter to increment when a deasserted to asserted transition occurs for the conditions that can be expressed by any of the fields in this register.
- PMI (bit 20): When set, the uncore will generate an interrupt request when this counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.
- EN (bit 22): When clear, this counter is locally disabled. When set, this counter is locally enabled and counting starts when the corresponding EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- INV (bit 23): When clear, the Counter Mask field is interpreted as greater than or equal to. When set, the Counter Mask field is interpreted as less than.
- Counter Mask (bits 31:24): When this field is clear, it has no effect on counting. When set to a value other than zero, the logical processor compares this field to the event counts on each core clock cycle. If INV is clear and the event counts are greater than or equal to this field, the counter is incremented by one. If INV is set and the event counts are less than this field, the counter is incremented by one. Otherwise the counter is not incremented.

Figure 20-23 shows the layout of MSR_UNCORE_FIXED_CTR_CTRL.

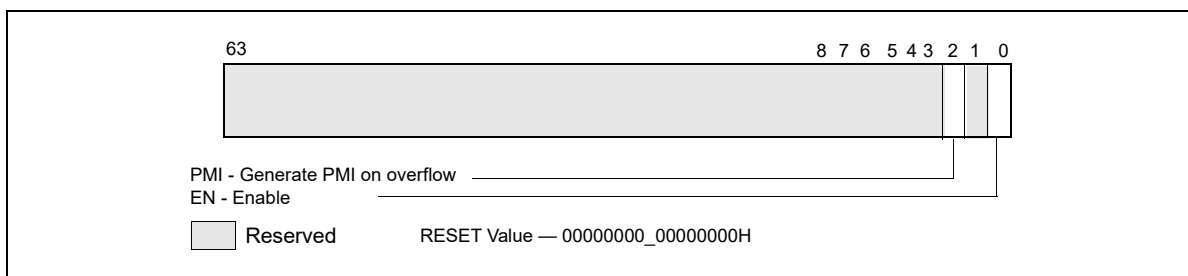


Figure 20-23. Layout of MSR_UNCORE_FIXED_CTR_CTRL MSR

- EN (bit 0): When clear, the uncore fixed-function counter is locally disabled. When set, it is locally enabled and counting starts when the EN_FC0 bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- PMI (bit 2): When set, the uncore will generate an interrupt request when the uncore fixed-function counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.

Both the general-purpose counters (MSR_UNCORE_PerfCnt) and the fixed-function counter (MSR_UNCORE_FixedCnt0) are 48 bits wide. They support both counting and interrupt based sampling usages. The event logic unit can filter event counts to specific regions of code or transaction types incoming to the home node logic.

20.3.1.2.3 Uncore Address/Opcode Match MSR

The Event Select field [7:0] of MSR_UNCORE_PERFEVTSELx is used to select different uncore event logic unit. When the event "ADDR_OPCODE_MATCH" is selected in the Event Select field, software can filter uncore performance events according to transaction address and certain transaction responses. The address filter and transaction response filtering requires the use of MSR_UNCORE_ADDR_OPCODE_MATCH register. The layout is shown in Figure 20-24.

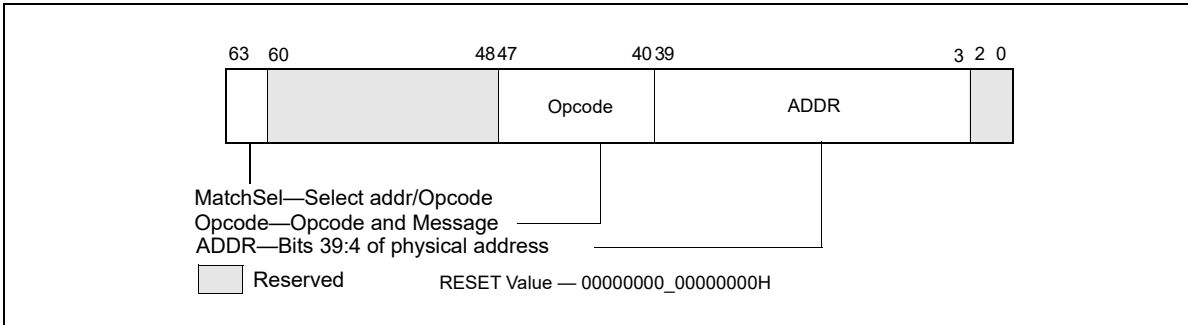


Figure 20-24. Layout of MSR_UNCORE_ADDR_OPCODE_MATCH MSR

- Addr (bits 39:3): The physical address to match if "MatchSel" field is set to select address match. The uncore performance counter will increment if the lowest 40-bit incoming physical address (excluding bits 2:0) for a transaction request matches bits 39:3.
- Opcode (bits 47:40) : Bits 47:40 allow software to filter uncore transactions based on QPI link message class/packed header opcode. These bits are consists two sub-fields:
 - Bits 43:40 specify the QPI packet header opcode.
 - Bits 47:44 specify the QPI message classes.

Table 20-7 lists the encodings supported in the opcode field.

Table 20-7. Opcode Field Encoding for MSR_UNCORE_ADDR_OPCODE_MATCH

Opcode [43:40]	QPI Message Class		
	Home Request [47:44] = 0000B	Snoop Response [47:44] = 0001B	Data Response [47:44] = 1110B
		1	
DMND_IFETCH	2	2	
WB	3	3	
PF_DATA_RD	4	4	
PF_RFO	5	5	
PF_IFETCH	6	6	
OTHER	7	7	
NON_DRAM	15	15	

- MatchSel (bits 63:61): Software specifies the match criteria according to the following encoding:
 - 000B: Disable addr_opcode match hardware.
 - 100B: Count if only the address field matches.
 - 010B: Count if only the opcode field matches.
 - 110B: Count if either opcode field matches or the address field matches.
 - 001B: Count only if both opcode and address field match.
 - Other encoding are reserved.

20.3.1.3 Intel® Xeon® Processor 7500 Series Performance Monitoring Facility

The performance monitoring facility in the processor core of Intel® Xeon® processor 7500 series are the same as those supported in Intel Xeon processor 5500 series. The uncore subsystem in Intel Xeon processor 7500 series are significantly different. The uncore performance monitoring facility consist of many distributed units associated with individual logic control units (referred to as boxes) within the uncore subsystem. A high level block diagram of the various box units of the uncore is shown in Figure 20-25.

Uncore PMUs are programmed via MSR interfaces. Each of the distributed uncore PMU units have several general-purpose counters. Each counter requires an associated event select MSR, and may require additional MSRs to configure sub-event conditions. The uncore PMU MSRs associated with each box can be categorized based on its functional scope: per-counter, per-box, or global across the uncore. The number counters available in each box type are different. Each box generally provides a set of MSRs to enable/disable, check status/overflow of multiple counters within each box.

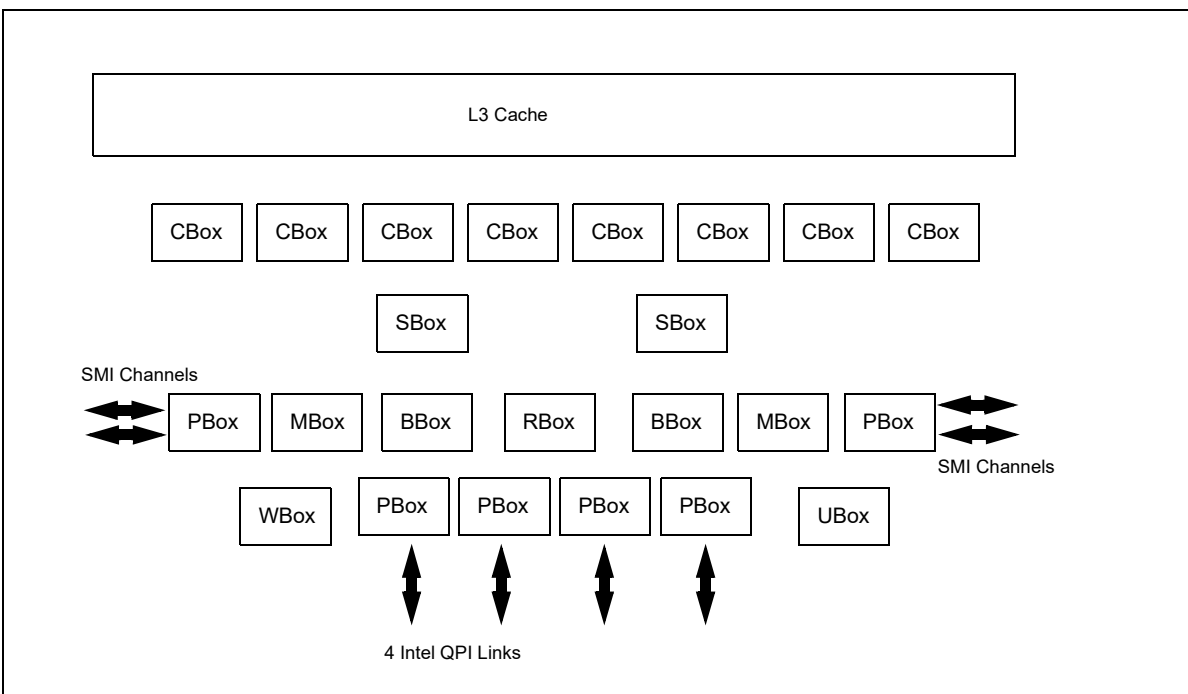


Figure 20-25. Distributed Units of the Uncore of Intel® Xeon® Processor 7500 Series

Table 20-8 summarizes the number MSRs for uncore PMU for each box.

Table 20-8. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

The W-Box provides 4 general-purpose counters, each requiring an event select configuration MSR, similar to the general-purpose counters in other boxes. There is also a fixed-function counter that increments clockticks in the uncore clock domain.

For C,S,B,M,R, and W boxes, each box provides an MSR to enable/disable counting, configuring PMI of multiple counters within the same box, this is somewhat similar the "global control" programming interface, IA32_PERF_GLOBAL_CTRL, offered in the core PMU. Similarly status information and counter overflow control for multiple counters within the same box are also provided in C,S,B,M,R, and W boxes.

In the U-Box, MSR_U_PMON_GLOBAL_CTL provides overall uncore PMU enable/disable and PMI configuration control. The scope of status information in the U-box is at per-box granularity, in contrast to the per-box status information MSR (in the C,S,B,M,R, and W boxes) providing status information of individual counter overflow. The difference in scope also apply to the overflow control MSR in the U-Box versus those in the other Boxes.

The individual MSRs that provide uncore PMU interfaces are listed in Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, Table 2-17 under the general naming style of MSR_%box#%_PMON_%scope_function%, where %box#% designates the type of box and zero-based index if there are more the one box of the same type, %scope_function% follows the examples below:

- Multi-counter enabling MSRs: MSR_U_PMON_GLOBAL_CTL, MSR_S0_PMON_BOX_CTL, MSR_C7_PMON_BOX_CTL, etc.
- Multi-counter status MSRs: MSR_U_PMON_GLOBAL_STATUS, MSR_S0_PMON_BOX_STATUS, MSR_C7_PMON_BOX_STATUS, etc.
- Multi-counter overflow control MSRs: MSR_U_PMON_GLOBAL_OVF_CTL, MSR_S0_PMON_BOX_OVF_CTL, MSR_C7_PMON_BOX_OVF_CTL, etc.
- Performance counters MSRs: the scope is implicitly per counter, e.g., MSR_U_PMON_CTR, MSR_S0_PMON_CTR0, MSR_C7_PMON_CTR5, etc.
- Event select MSRs: the scope is implicitly per counter, e.g., MSR_U_PMON_EVNT_SEL, MSR_S0_PMON_EVNT_SELO, MSR_C7_PMON_EVNT_SEL5, etc.
- Sub-control MSRs: the scope is implicitly per-box granularity, e.g., MSR_M0_PMON_TIMESTAMP, MSR_R0_PMON_IPERF0_P1, MSR_S1_PMON_MATCH.

Details of uncore PMU MSR bit field definitions can be found in a separate document "Intel Xeon Processor 7500 Series Uncore Performance Monitoring Guide".

20.3.2 Performance Monitoring for Processors Based on Westmere Microarchitecture

All of the performance monitoring programming interfaces (architectural and non-architectural core PMU facilities, and uncore PMU) described in Section 20.6.3 also apply to processors based on Westmere microarchitecture.

Table 20-5 describes a non-architectural performance monitoring event (event code 0B7H) and associated MSR_OFFCORE_RSP_0 (address 1A6H) in the core PMU. This event and a second functionally equivalent offcore

response event using event code 0BBH and MSR_OFFCORE_RSP_1 (address 1A7H) are supported in processors based on Westmere microarchitecture. The event code and event mask definitions of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The load latency facility is the same as described in Section 20.3.1.1.2, but added enhancement to provide more information in the data source encoding field of each load latency record. The additional information relates to STLB_MISS and LOCK, see Table 20-13.

20.3.3 Intel® Xeon® Processor E7 Family Performance Monitoring Facility

The performance monitoring facility in the processor core of the Intel® Xeon® processor E7 family is the same as those supported in the Intel Xeon processor 5600 series¹. The uncore subsystem in the Intel Xeon processor E7 family is similar to those of the Intel Xeon processor 7500 series. The high level construction of the uncore subsystem is similar to that shown in Figure 20-25, with the additional capability that up to 10 C-Box units are supported.

Table 20-9 summarizes the number MSRs for uncore PMU for each box.

Table 20-9. Uncore PMU MSR Summary for Intel® Xeon® Processor E7 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	10	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E7 family is available in the “Intel® Xeon® Processor E7 Uncore Performance Monitoring Programming Reference Manual”.

20.3.4 Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture

Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel® Xeon® processor E3-1200 family are based on Sandy Bridge microarchitecture; this section describes the performance monitoring facilities provided in the processor core. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 20.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 20.2.3.

The core PMU’s capability is similar to those described in Section 20.3.1.1 and Section 20.6.3, with some differences and enhancements relative to Westmere microarchitecture summarized in Table 20-10.

1. Exceptions are indicated for event code 0FH in the event list for this processor (<https://perfmon-events.intel.com/>); and valid bits of data source encoding field of each load latency record is limited to bits 5:4 of Table 20-13.

Table 20-10. Core PMU Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 20.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 20.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W:32	See Section 20.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to determine # of counters. See Section 20.2.1.
PMI Overhead Mitigation	<ul style="list-style-type: none"> Freeze_Perfmon_on_PMI with legacy semantics. Freeze_LBR_on_PMI with legacy semantics for branch profiling. Freeze_while_SMM. 	<ul style="list-style-type: none"> Freeze_Perfmon_on_PMI with legacy semantics. Freeze_LBR_on_PMI with legacy semantics for branch profiling. Freeze_while_SMM. 	See Section 18.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 20-12.	See Table 20-84.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 20.3.4.4.2; <ul style="list-style-type: none"> Data source encoding STLB miss encoding Lock transaction encoding 	Data source encoding	
PEBS-Precise Store	Section 20.3.4.4.3	No	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL).	No	
Off-core Response Event	MSR 1A6H and 1A7H, extended request and response types.	MSR 1A6H and 1A7H, limited response types.	Nehalem supports 1A6H only.

20.3.4.1 Global Counter Control Facilities in Sandy Bridge Microarchitecture

The number of general-purpose performance counters visible to a logical processor can vary across Processors based on Sandy Bridge microarchitecture. Software must use CPUID to determine the number performance counters/event select registers (See Section 20.2.1.1).

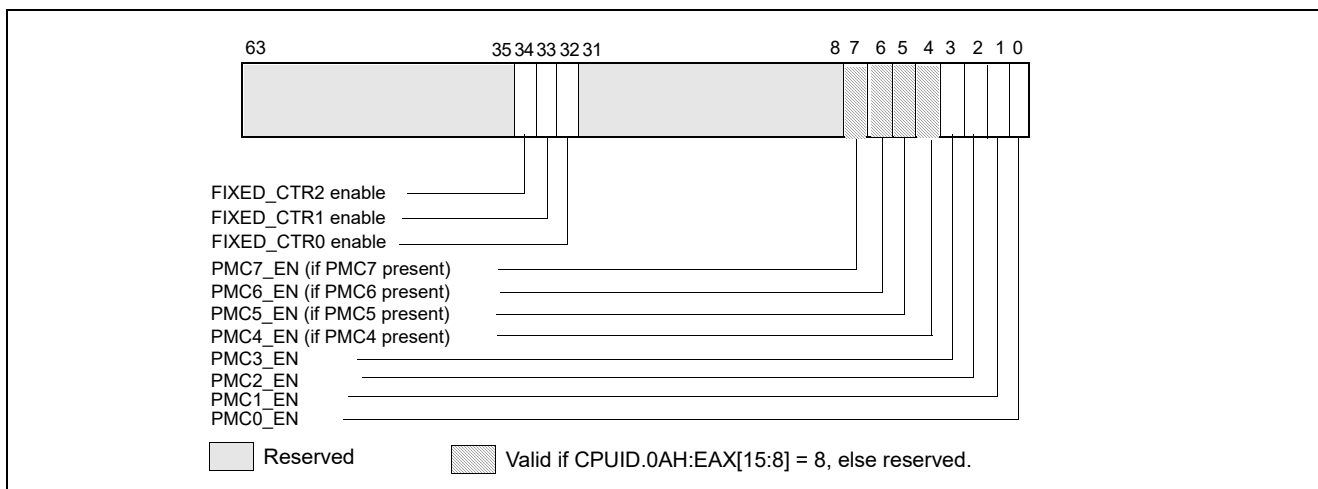


Figure 20-26. IA32_PERF_GLOBAL_CTRL MSR in Sandy Bridge Microarchitecture

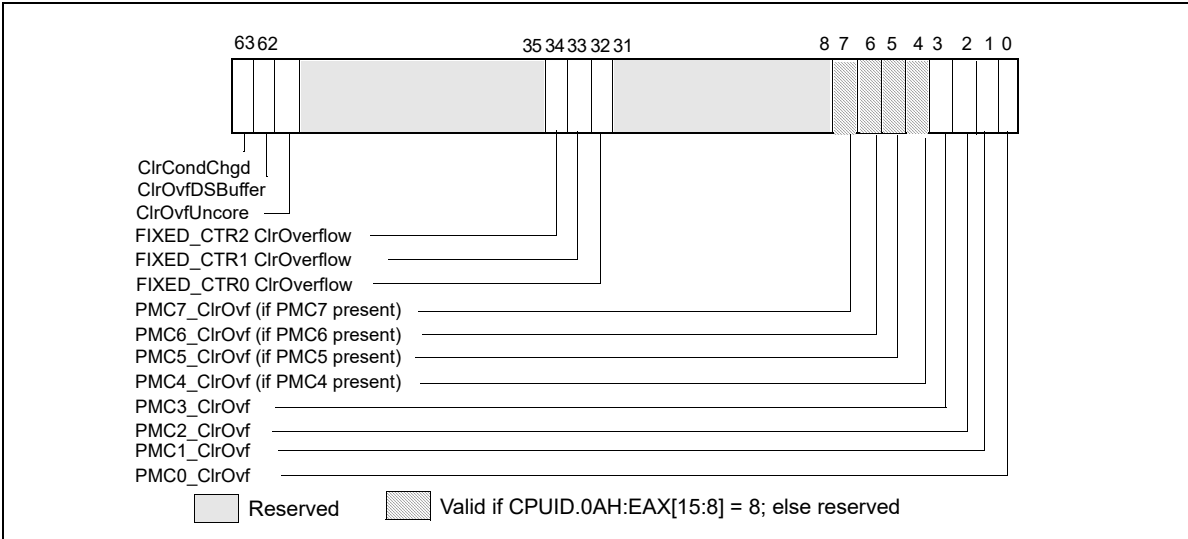


Figure 20-28. IA32_PERF_GLOBAL_OVF_CTRL MSR in Sandy Bridge Microarchitecture

20.3.4.2 Counter Coalescence

In processors based on Sandy Bridge microarchitecture, each processor core implements eight general-purpose counters. CPUID.0AH:EAX[15:8] will report the number of counters visible to software.

If a processor core is shared by two logical processors, each logical processors can access up to four counters (IA32_PMC0-IA32_PMC3). This is the same as in the prior generation for processors based on Nehalem microarchitecture.

If a processor core is not shared by two logical processors, up to eight general-purpose counters are visible. If CPUID.0AH:EAX[15:8] reports 8 counters, then IA32_PMC4-IA32_PMC7 would occupy MSR addresses 0C5H through 0C8H. Each counter is accompanied by an event select MSR (IA32_PERFEVTSEL4-IA32_PERFEVTSEL7).

If CPUID.0AH:EAX[15:8] report 4, access to IA32_PMC4-IA32_PMC7, IA32_PMC4-IA32_PMC7 will cause #GP. Writing 1's to bit position 7:4 of IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, or IA32_PERF_GLOBAL_OVF_CTL will also cause #GP.

20.3.4.3 Full Width Writes to Performance Counters

Processors based on Sandy Bridge microarchitecture support full-width writes to the general-purpose counters, IA32_PMCx. Support of full-width writes are enumerated by IA32_PERF_CAPABILITIES.FW_WRITES[13] (see Section 20.2.4).

The default behavior of IA32_PMCx is unchanged, i.e., WRMSR to IA32_PMCx results in a sign-extended 32-bit value of the input EAX written into IA32_PMCx. Full-width writes must issue WRMSR to a dedicated alias MSR address for each IA32_PMCx.

Software must check the presence of full-width write capability and the presence of the alias address IA32_A_PMCx by testing IA32_PERF_CAPABILITIES[13].

20.3.4.4 PEBS Support in Sandy Bridge Microarchitecture

Processors based on Sandy Bridge microarchitecture support PEBS, similar to those offered in prior generation, with several enhanced features. The key components and differences of PEBS facility relative to Westmere microarchitecture is summarized in Table 20-11.

Table 20-11. PEBS Facility Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 20.3.1.1.1	Section 20.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 20-29	Figure 20-15	
PEBS record layout	Physical Layout same as Table 20-3.	Table 20-3	Enhanced fields at offsets 98H, A0H, A8H.
PEBS Events	See Table 20-12.	See Table 20-84.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 20-13.	Table 20-4	
PEBS-Precise Store	Yes; see Section 20.3.4.4.3.	No	IA32_PMC3 only
PEBS-PDIR	Yes	No	IA32_PMC1 only
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

In IA32_PEBS_ENABLE MSR, bit 63 is defined as PS_ENABLE: When set, this enables IA32_PMC3 to capture precise store information. Only IA32_PMC3 supports the precise store facility. In typical usage of PEBS, the bit fields in IA32_PEBS_ENABLE are written to when the agent software starts PEBS operation; the enabled bit fields should be modified only when re-programming another PEBS event or cleared when the agent uses the performance counters for non-PEBS operations.

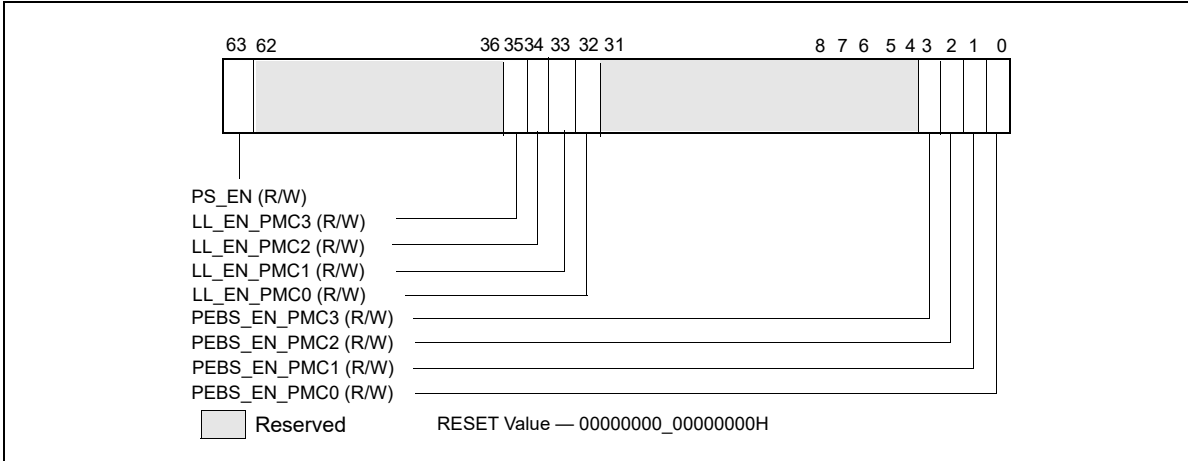


Figure 20-29. Layout of IA32_PEBS_ENABLE MSR

20.3.4.4.1 PEBS Record Format

The layout of PEBS records physically identical to those shown in Table 20-3, but the fields at offsets 98H, A0H, and A8H have been enhanced to support additional PEBS capabilities.

- Load/Store Data Linear Address (Offset 98H): This field will contain the linear address of the source of the load, or linear address of the destination of the store.
- Data Source /Store Status (Offset A0H): When load latency is enabled, this field will contain three piece of information (including an encoded value indicating the source which satisfied the load operation). The source field encodings are detailed in Table 20-4. When precise store is enabled, this field will contain information indicating the status of the store, as detailed in Table 19.
- Latency Value/0 (Offset A8H): When load latency is enabled, this field contains the latency in cycles to service the load. This field is not meaningful when precise store is enabled and will be written to zero in that case. Upon writing the PEBS record, microcode clears the overflow status bits in the IA32_PERF_GLOBAL_STATUS corresponding to those counters that both overflowed and were enabled in the IA32_PEBS_ENABLE register. The status bits of other counters remain unaffected.

The number PEBS events has expanded. The list of PEBS events supported in Sandy Bridge microarchitecture is shown in Table 20-12.

Table 20-12. PEBS Performance Events for Sandy Bridge Microarchitecture

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST	01H ¹
UOPS_RETIRED	C2H	All	01H
		Retire_Slots	02H
BR_INST_RETIRED	C4H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Near_Return	08H
		Near_Taken	20H
BR_MISP_RETIRED	C5H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Not_Taken	10H
		Taken	20H

Table 20-12. PEBS Performance Events for Sandy Bridge Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_UOPS_RETIRED	DOH	STLB_MISS_LOADS	11H
		STLB_MISS_STORE	12H
		LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H
MEM_LOAD_UOPS_RETIRED	D1H	L1_Hit	01H
		L2_Hit	02H
		L3_Hit	04H
		Hit_LFB	40H
MEM_LOAD_UOPS_LLC_HIT_RETIRED	D2H	XSNP_Miss	01H
		XSNP_Hit	02H
		XSNP_Hitm	04H
		XSNP_None	08H

NOTES:

1. Only available on IA32_PMC1.

20.3.4.4.2 Load Latency Performance Monitoring Facility

The load latency facility in Sandy Bridge microarchitecture is similar to that in prior microarchitectures. It provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table 20-3 and Section 20.3.4.4.1. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_TRANS_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001.00000001H.
- When Load latency event is enabled, no other PEBS event can be configured with other counters.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally. The MEM_TRANS_RETIRED event for load latency counts only tagged retired loads. If a load is cancelled it will not be counted and the internal state of the load latency facility will not be updated. In this case the hardware will tag the next available load.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The physical layout of the PEBS records is the same as shown in Table 20-3. The specificity of Data Source entry at offset A0H has been enhanced to report three pieces of information.

Table 20-13. Layout of Data Source Field of Load Latency Record

Field	Position	Description
Source	3:0	See Table 20-4
STLB_MISS	4	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
Lock	5	0: The load was not part of a locked transaction. 1: The load was part of a locked transaction.
Reserved	63:6	Reserved

The layout of MSR_PEBS_LD_LAT_THRESHOLD is the same as shown in Figure 20-17.

20.3.4.4.3 Precise Store Facility

Processors based on Sandy Bridge microarchitecture offer a precise store capability that complements the load latency facility. It provides a means to profile store memory references in the system.

Precise stores leverage the PEBS facility and provide additional information about sampled stores. Having precise memory reference events with linear address information for both loads and stores can help programmers improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

Only IA32_PMC3 can be used to capture precise store information. After enabling this facility, counter overflows will initiate the generation of PEBS records as previously described in PEBS. Upon counter overflow hardware captures the linear address and other status information of the next store that retires. This information is then written to the PEBS record.

To enable the precise store facility, software must complete the following steps. Please note that the precise store facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture precise store information.

- Complete the PEBS configuration steps.
- Program the MEM_TRANS_RETIRED.PRECISE_STORE event in IA32_PERFVTSEL3. Only counter 3 (IA32_PMC3) supports collection of precise store information.
- Set IA32_PEBS_ENABLE[3] and IA32_PEBS_ENABLE[63]. This enables IA32_PMC3 as a PEBS counter and enables the precise store facility, respectively.

The precise store information written into a PEBS record affects entries at offsets 98H, A0H, and A8H of Table 20-3. The specificity of Data Source entry at offset A0H has been enhanced to report three piece of information.

Table 20-14. Layout of Precise Store Information In PEBS Record

Field	Offset	Description
Store Data Linear Address	98H	The linear address of the destination of the store.
Store Status	A0H	L1D Hit (Bit 0): The store hit the data cache closest to the core (lowest latency cache) if this bit is set, otherwise the store missed the data cache. STLB Miss (bit 4): The store missed the STLB if set, otherwise the store hit the STLB Locked Access (bit 5): The store was part of a locked access if set, otherwise the store was not part of a locked access.
Reserved	A8H	Reserved

20.3.4.4.4 Precise Distribution of Instructions Retired (PDIR)

Upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. INST_RETIREED is a very common event that is used to sample where performance bottleneck happened and to help identify its location in instruction address space. Even if the delay is constant in core clock space, it invariably manifest as variable “skids” in instruction address space. This creates a challenge for programmers to profile a workload and pinpoint the location of bottlenecks.

The core PMU in processors based on Sandy Bridge microarchitecture include a facility referred to as precise distribution of Instruction Retired (PDIR).

The PDIR facility mitigates the “skid” problem by providing an early indication of when the INST_RETIREED counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow. On processors based on Sandy Bridge microarchitecture, skid is significantly reduced and can be as little as one instruction. On future implementations, PDIR may eliminate skid.

PDIR applies only to the INST_RETIREED.ALL precise event, and processors based on Sandy Bridge microarchitecture must use IA32_PMC1 with PerfEvtSel1 property configured and bit 1 in the IA32_PEBS_ENABLE set to 1. INST_RETIREED.ALL is a non-architectural performance event, it is not supported in prior generation microarchitectures. Additionally, on processors with CPUID DisplayFamily_DisplayModel signatures of 06_2A and 06_2D, the tool that programs PDIR should quiesce the rest of the programmable counters in the core when PDIR is active.

20.3.4.5 Off-core Response Performance Monitoring

The core PMU in processors based on Sandy Bridge microarchitecture provides off-core response facility similar to prior generation. Off-core response can be programmed only with a specific pair of event select and counter MSR, and with specific event codes and predefine mask bit value in a dedicated MSR to specify attributes of the off-core transaction. Two event codes are dedicated for off-core response event programming. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Table 20-15 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 20-15. Off-Core Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMCO-3	B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)
PMCO-3	BBH	01H	MSR_OFFCORE_RSP_1 (address 1A7H)

The layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 are shown in Figure 20-30 and Figure 20-31. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

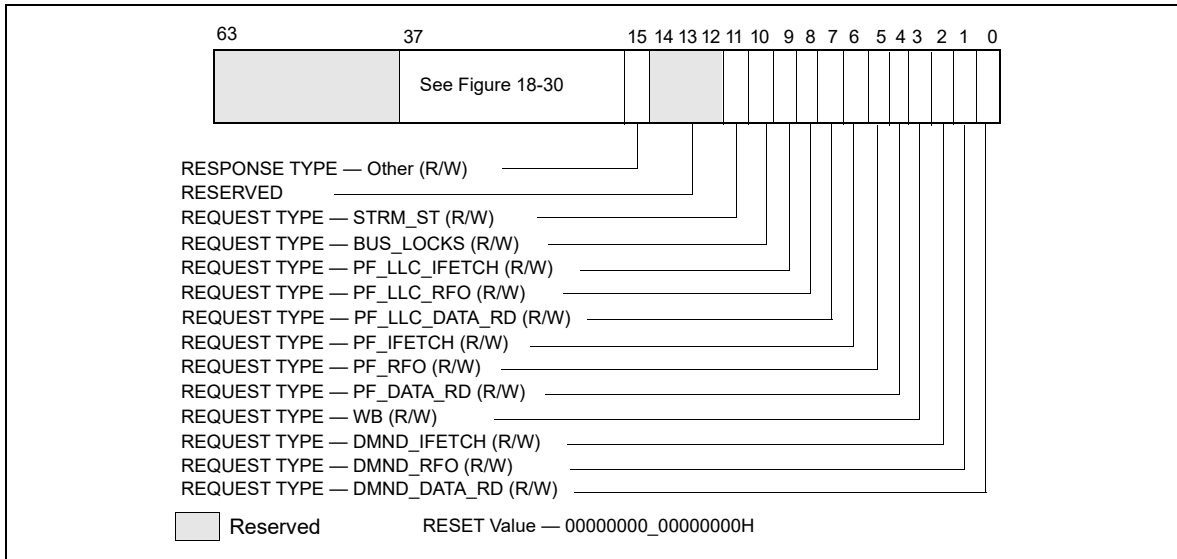


Figure 20-30. Request_Type Fields for MSR_OFFCORE_RSP_x

Table 20-16. MSR_OFFCORE_RSP_x Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_LLC_DATA_RD	7	L2 prefetcher to L3 for loads.
PF_LLC_RFO	8	RFO requests generated by L2 prefetcher
PF_LLC_IFETCH	9	L2 prefetcher to L3 for instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests
STRM_ST	11	Streaming store requests
OTHER	15	Any other request that crosses IDI, including I/O.

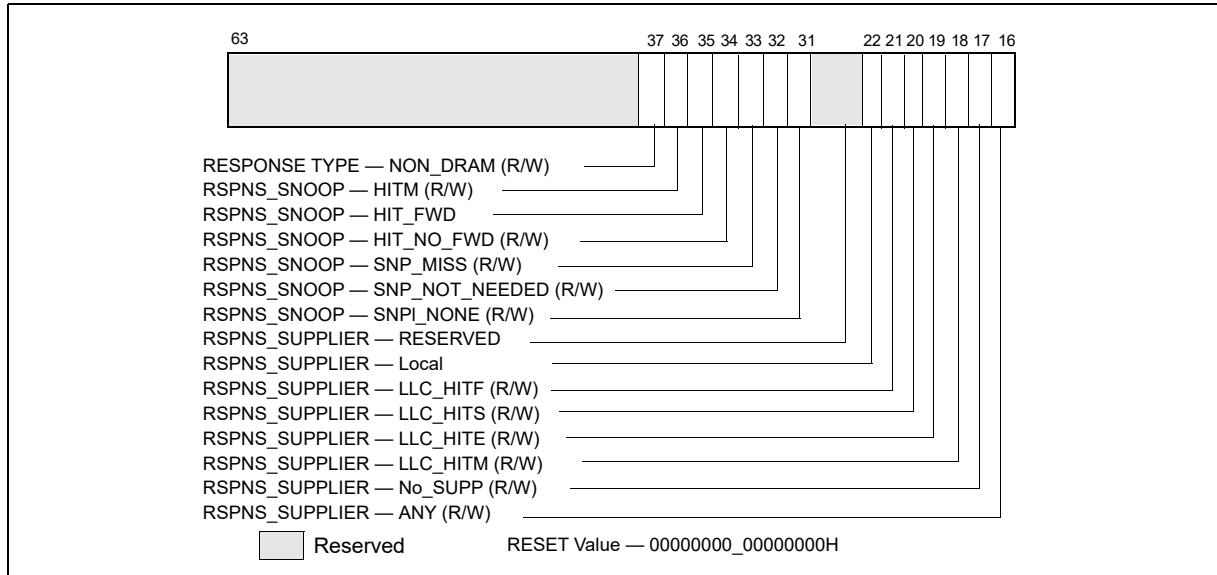


Figure 20-31. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSP_x

To properly program this extra register, software must set at least one request type bit and a valid response type pattern. Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSP_x allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 20-17. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 20-18. MSR_OFFCORE_RSP_x Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	SNP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNP_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO) -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD) -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S) In the LLC Miss case, data is returned from DRAM.
	SNP_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD) -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO) -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

20.3.4.6 Uncore Performance Monitoring Facilities in the Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, and Intel® Core™ i3-2xxx Processor Series

The uncore sub-system in Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series provides a unified L3 that can support up to four processor cores. The L3 cache consists multiple slices, each slice interface with a processor via a coherence engine, referred to as a C-Box. Each C-Box provides dedicated facility of MSRs to select uncore performance monitoring events and each C-Box event select MSR is paired with a counter register, similar in style as those described in Section 20.3.1.2.2. The ARB unit in the uncore also provides its local performance counters and event select MSRs. The layout of the event select MSRs in the C-Boxes and the ARB unit are shown in Figure 20-32.

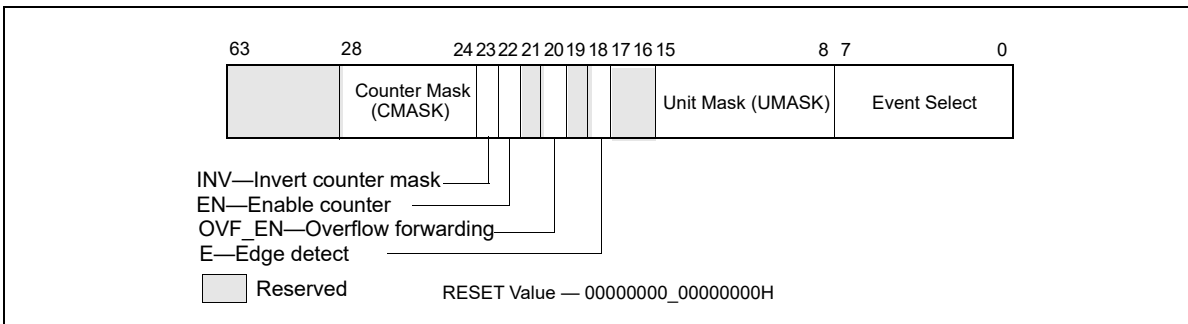


Figure 20-32. Layout of Uncore PERFVTSSEL MSR for a C-Box Unit or the ARB Unit

The bit fields of the uncore event select MSRs for a C-box unit or the ARB unit are summarized below:

- Event_Select (bits 7:0) and UMASK (bits 15:8): Specifies the microarchitectural condition to count in a local uncore PMU counter, see the event list at: <https://perfmon-events.intel.com/>.
- E (bit 18): Enables edge detection filtering, if 1.
- OVF_EN (bit 20): Enables the overflow indicator from the uncore counter forwarded to MSR_UNC_PERF_-GLOBAL_CTRL, if 1.
- EN (bit 22): Enables the local counter associated with this event select MSR.
- INV (bit 23): Event count increments with non-negative value if 0, with negated value if 1.
- CMASK (bits 28:24): Specifies a positive threshold value to filter raw event count input.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 20-33 shows the layout of the uncore domain global control.

When an uncore counter overflows, a PMI can be routed to a processor core. Bits 3:0 of MSR_UNC_PERF_-GLOBAL_CTRL can be used to select which processor core to handle the uncore PMI. Software must then write to bit 13 of IA32_DEBUGCTL (at address 1D9H) to enable this capability.

- PMI_SEL_Core#: Enables the forwarding of an uncore PMI request to a processor core, if 1. If bit 30 (WakePMI) is '1', a wake request is sent to the respective processor core prior to sending the PMI.
- EN: Enables the fixed uncore counter, the ARB counters, and the CBO counters in the uncore PMU, if 1. This bit is cleared if bit 31 (FREEZE) is set and any enabled uncore counters overflow.
- WakePMI: Controls sending a wake request to any halted processor core before issuing the uncore PMI request. If a processor core was halted and not sent a wake request, the uncore PMI will not be serviced by the processor core.
- FREEZE: Provides the capability to freeze all uncore counters when an overflow condition occurs in a unit counter. When this bit is set, and a counter overflow occurs, the uncore PMU logic will clear the global enable bit (bit 29).

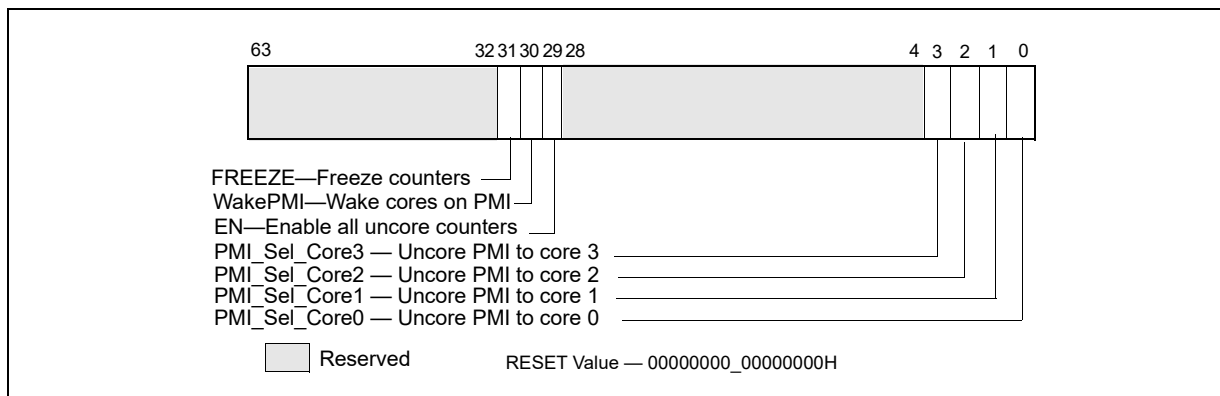


Figure 20-33. Layout of MSR_UNC_PERF_GLOBAL_CTRL MSR for Uncore

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 20-19 summarizes the number MSRs for uncore PMU for each box.

Table 20-19. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, see Table 2-21 MSR_UNC_CBO_CONFIG
ARB	1	2	44	Yes	Uncore	

Table 20-19. Uncore PMU MSR Summary (Contd.)

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
Fixed Counter	N.A.	N.A.	48	No	Uncore	

20.3.4.6.1 Uncore Performance Monitoring Events

There are certain restrictions on the uncore performance counters in each C-Box. Specifically,

- Occupancy events are supported only with counter 0 but not counter 1.
- Other uncore C-Box events can be programmed with either counter 0 or 1.

The C-Box uncore performance events can collect performance characteristics of transactions initiated by processor core. In that respect, they are similar to various sub-events in the OFFCORE_RESPONSE family of performance events in the core PMU. Information such as data supplier locality (LLC HIT/MISS) and snoop responses can be collected via OFFCORE_RESPONSE and qualified on a per-thread basis.

On the other hand, uncore performance event logic cannot associate its counts with the same level of per-thread qualification attributes as the core PMU events can. Therefore, whenever similar event programming capabilities are available from both core PMU and uncore PMU, the recommendation is that utilizing the core PMU events may be less affected by artifacts, complex interactions and other factors.

20.3.4.7 Intel® Xeon® Processor E5 Family Performance Monitoring Facility

The Intel® Xeon® Processor E5 Family (and Intel® Core™ i7-3930K Processor) are based on Sandy Bridge-E microarchitecture. While the processor cores share the same microarchitecture as those of the Intel® Xeon® Processor E3 Family and 2nd generation Intel Core i7-2xxx, Intel Core i5-2xxx, Intel Core i3-2xxx processor series, the uncore subsystems are different. An overview of the uncore performance monitoring facilities of the Intel Xeon processor E5 family (and Intel Core i7-3930K processor) is described in Section 20.3.4.8.

Thus, the performance monitoring facilities in the processor core generally are the same as those described in Section 20.6.3 through Section 20.3.4.5. However, the MSR_OFFCORE_RSP_0/MSR_OFFCORE_RSP_1 Response Supplier Info field shown in Table 20-17 applies to Intel Core Processors with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2AH; Intel Xeon processor with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2DH supports an additional field for remote DRAM controller shown in Table 20-20. Additionally, there are some small differences in the non-architectural performance monitoring events (see event list available at: <https://perfmon-events.intel.com/>).

Table 20-20. MSR_OFFCORE_RSP_x Supplier Info Field Definitions

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Remote	30:23	Remote DRAM Controller (either all 0s or all 1s).

20.3.4.8 Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5-2600 product family has some similarities with those of the Intel Xeon processor E7 family. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope. For example, each Cbox caching agent has a set of local performance counters, and the power controller unit (PCU) has its own local performance counters. Up to 8 C-Box units are supported in the uncore sub-system.

Table 20-21 summarizes the uncore PMU facilities providing MSR interfaces.

Table 20-21. Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	4	44	Yes	per-box	None
PCU	1	4	48	Yes	per-box	Match/Mask
U-Box	1	2	44	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 family is available in “Intel® Xeon® Processor E5 Uncore Performance Monitoring Programming Reference Manual”. The MSR-based uncore PMU interfaces are listed in Table 2-24.

20.3.5 3rd Generation Intel® Core™ Processor Performance Monitoring Facility

The 3rd generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v2 product family are based on the Ivy Bridge microarchitecture. The performance monitoring facilities in the processor core generally are the same as those described in Section 20.6.3 through Section 20.3.4.5. The non-architectural performance monitoring events supported by the processor core can be found at: <https://perfmon-events.intel.com/>.

20.3.5.1 Intel® Xeon® Processor E5 v2 and E7 v2 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5 v2 and Intel Xeon Processor E7 v2 product families are based on the Ivy Bridge-E microarchitecture. There are some similarities with those of the Intel Xeon processor E5 family based on the Sandy Bridge microarchitecture. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope.

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v2 and Intel Xeon Processor E7 v2 families are available in the “Intel® Xeon® Processor E5 v2 and E7 v2 Uncore Performance Monitoring Programming Reference Manual”. The MSR-based uncore PMU interfaces are listed in Table 2-28.

20.3.6 4th Generation Intel® Core™ Processor Performance Monitoring Facility

The 4th generation Intel® Core™ processor and Intel® Xeon® processor E3-1200 v3 product family are based on the Haswell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 20.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 20.2.3.

The core PMU’s capability is similar to those described in Section 20.6.3 through Section 20.3.4.5, with some differences and enhancements summarized in Table 20-22. Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 20.3.6.5. For details of Intel TSX, see Chapter 16, “Programming with Intel® Transactional Synchronization Extensions” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

Table 20-22. Core PMU Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 20.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 20.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 20.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 20.2.1.
PMI Overhead Mitigation	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_LBR_on_PMI with legacy semantics for branch profiling. ▪ Freeze_while_SMM. 	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_LBR_on_PMI with legacy semantics for branch profiling. ▪ Freeze_while_SMM. 	See Section 18.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 20-12 and Section 20.3.6.5.1.	See Table 20-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 20.3.4.4.2.	See Section 20.3.4.4.2.	
PEBS-Precise Store	No, replaced by Data Address profiling.	Section 20.3.4.4.3	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL)	Yes (using precise INST_RETIRED.ALL)	
PEBS-EventingIP	Yes	No	
Data Address Profiling	Yes	No	
LBR Profiling	Yes	Yes	
Call Stack Profiling	Yes, see Section 18.11.	No	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; extended request and response types.	MSR 1A6H and 1A7H; extended request and response types.	
Intel TSX support for Perfmon	See Section 20.3.6.5.	No	

20.3.6.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 4th Generation Intel Core processor is similar to those in processors based on Sandy Bridge microarchitecture, with several enhanced features. The key components and differences of PEBS facility relative to Sandy Bridge microarchitecture is summarized in Table 20-23.

Table 20-23. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
Valid IA32_PMCx	PMCO-PMC3	PMCO-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 20.3.1.1.1	Section 20.3.1.1.1	Unchanged
IA32_PEBES_ENABLE Layout	Figure 20-15	Figure 20-29	
PEBS record layout	Table 20-24; enhanced fields at offsets 98H, A0H, A8H, B0H.	Table 20-3; enhanced fields at offsets 98H, A0H, A8H.	

Table 20-23. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
Precise Events	See Table 20-12.	See Table 20-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 20-13.	Table 20-13	
PEBS-Precise Store	No, replaced by data address profiling.	Yes; see Section 20.3.4.4.3.	
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

20.3.6.2 PEBS Data Format

The PEBS record format for the 4th Generation Intel Core processor is shown in Table 20-24. The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 20-24. PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	Data Linear Address
40H	R/EBP	A0H	Data Source Encoding
48H	R/ESP	A8H	Latency value (core cycles)
50H	R8	B0H	EventingIP
58H	R9	B8H	TX Abort Information (Section 20.3.6.5.1)

The layout of PEBS records are almost identical to those shown in Table 20-3. Offset B0H is a new field that records the eventing IP address of the retired instruction that triggered the PEBS assist.

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 20.3.4.4.2), PDIR (Section 20.3.4.4.4), and the equivalent capability of precise store in prior generation (see Section 20.3.6.3).

In the core PMU of the 4th generation Intel Core processor, load latency facility and PDIR capabilities are unchanged. However, precise store is replaced by an enhanced capability, data address profiling, that is not restricted to store address. Data address profiling also records information in PEBS records at offsets 98H, A0H, and ABH.

20.3.6.3 PEBS Data Address Profiling

The Data Linear Address facility is also abbreviated as DataLA. The facility is a replacement or extension of the precise store facility in previous processor generations. The DataLA facility complements the load latency facility by providing a means to profile load and store memory references in the system, leverages the PEBS facility, and provides additional information about sampled loads and stores. Having precise memory reference events with linear address information for both loads and stores provides information to improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

The DataLA facility in the 4th generation processor supports the following events configured to use PEBS:

Table 20-25. Precise Events That Supports Data Linear Address Profiling

Event Name	Event Name
MEM_UOPS_RETIRED.STLB_MISS_LOADS	MEM_UOPS_RETIRED.STLB_MISS_STORES
MEM_UOPS_RETIRED.LOCK_LOADS	MEM_UOPS_RETIRED.SPLIT_STORES
MEM_UOPS_RETIRED.SPLIT_LOADS	MEM_UOPS_RETIRED.ALL_STORES
MEM_UOPS_RETIRED.ALL_LOADS	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM
MEM_LOAD_UOPS_RETIRED.L1_HIT	MEM_LOAD_UOPS_RETIRED.L2_HIT
MEM_LOAD_UOPS_RETIRED.L3_HIT	MEM_LOAD_UOPS_RETIRED.L1_MISS
MEM_LOAD_UOPS_RETIRED.L2_MISS	MEM_LOAD_UOPS_RETIRED.L3_MISS
MEM_LOAD_UOPS_RETIRED.HIT_LFB	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM
UOPS_RETIRED.ALL (if load or store is tagged)	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE

DataLA can use any one of the IA32_PMC0-IA32_PMC3 counters. Counter overflows will initiate the generation of PEBS records. Upon counter overflow, hardware captures the linear address and possible other status information of the retiring memory uop. This information is then written to the PEBS record that is subsequently generated.

To enable the DataLA facility, software must complete the following steps. Please note that the DataLA facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture DataLA information.

- Complete the PEBS configuration steps.
- Program an event listed in Table 20-25 using any one of IA32_PERFVTSEL0-IA32_PERFVTSEL3.
- Set the corresponding IA32_PEBS_ENABLE.PEBS_EN_CTRx bit. This enables the corresponding IA32_PMCx as a PEBS counter and enables the DataLA facility.

When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H, and A8H, as shown in Table 20-26.

Table 20-26. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	<ul style="list-style-type: none"> ▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_UOPS_RETIRED.STLB_MISS_STORES, MEM_UOPS_RETIRED.SPLIT_STORES, MEM_UOPS_RETIRED.ALL_STORES ▪ Other bits are zero, The STLB_MISS, LOCK bit information can be obtained by programming the corresponding store event in Table 20-25.
Reserved	A8H	Always zero.

20.3.6.3.1 EventingIP Record

The PEBS record layout for processors based on Haswell microarchitecture adds a new field at offset 0B0H. This is the eventingIP field that records the IP address of the retired instruction that triggered the PEBS assist. The EIP/RIP field at offset 08H records the IP address of the next instruction to be executed following the PEBS assist.

20.3.6.4 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 20.3.4.5. The event codes are listed in Table 20-15. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 20-27.
- Supplier information (bits 30:16): see Table 20-28.
- Snoop response information (bits 37:31): see Table 20-18.

Table 20-27. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture)

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
COREWB	3	Counts the number of modified cachelines written back.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_L3_DATA_RD	7	Counts the number of data cacheline reads generated by L3 prefetchers.
PF_L3_RFO	8	Counts the number of RFO requests generated by L3 prefetchers.
PF_L3_CODE_RD	9	Counts the number of code reads generated by L3 prefetchers.
SPLIT_LOCK_UC_LOCK	10	Counts the number of lock requests that split across two cachelines or are to UC memory.
STRM_ST	11	Counts the number of streaming store requests electronically.
Reserved	14:12	Reserved

Table 20-27. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture) (Contd.)

Bit Name	Offset	Description
OTHER	15	Any other request that crosses IDI, including I/O.

The supplier information field listed in Table 20-28. The fields vary across products (according to CPUID signatures) and is noted in the description.

Table 20-28. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures: 06_3CH, 06_46H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

Table 20-29. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_45H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	L4_HIT_LOCAL_L4	22	L4 Cache
	L4_HIT_REMOTE_HOPO_L4	23	L4 Cache
	L4_HIT_REMOTE_HOP1_L4	24	L4 Cache
	L4_HIT_REMOTE_HOP2P_L4	25	L4 Cache
Reserved	30:26	Reserved	

20.3.6.4.1 Off-core Response Performance Monitoring in Intel Xeon Processors E5 v3 Series

Table 20-28 lists the supplier information field that apply to Intel Xeon processor E5 v3 series (CPUID signature 06_3FH).

Table 20-30. MSR_OFFCORE_RSP_x Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	L3_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	26:23	Reserved
	L3_MISS_REMOTE_HOP0	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

20.3.6.5 Performance Monitoring and Intel® TSX

Chapter 16 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1 describes the details of Intel® Transactional Synchronization Extensions (Intel® TSX). This section describes performance monitoring support for Intel TSX.

If a processor supports Intel TSX, the core PMU enhances its IA32_PERFEVTSELx MSR with two additional bit fields for event filtering. Support for Intel TSX is indicated by either (a) CPUID.(EAX=7, ECX=0):RTM[bit 11]=1, or (b) if CPUID.07H.EBX.HLE [bit 4] = 1. The TSX-enhanced layout of IA32_PERFEVTSELx is shown in Figure 20-34. The two additional bit fields are:

- **IN_TX** (bit 32): When set, the counter will only include counts that occurred inside a transactional region, regardless of whether that region was aborted or committed. This bit may only be set if the processor supports HLE or RTM.
- **IN_TXCP** (bit 33): When set, the counter will not include counts that occurred inside of an aborted transactional region. This bit may only be set if the processor supports HLE or RTM. This bit may only be set for IA32_PERFEVTSEL2.

When the IA32_PERFEVTSELx MSR is programmed with both IN_TX=0 and IN_TXCP=0 on a processor that supports Intel TSX, the result in a counter may include detectable conditions associated with a transaction code region for its aborted execution (if any) and completed execution.

In the initial implementation, software may need to take pre-caution when using the IN_TXCP bit. See Table 2-29.

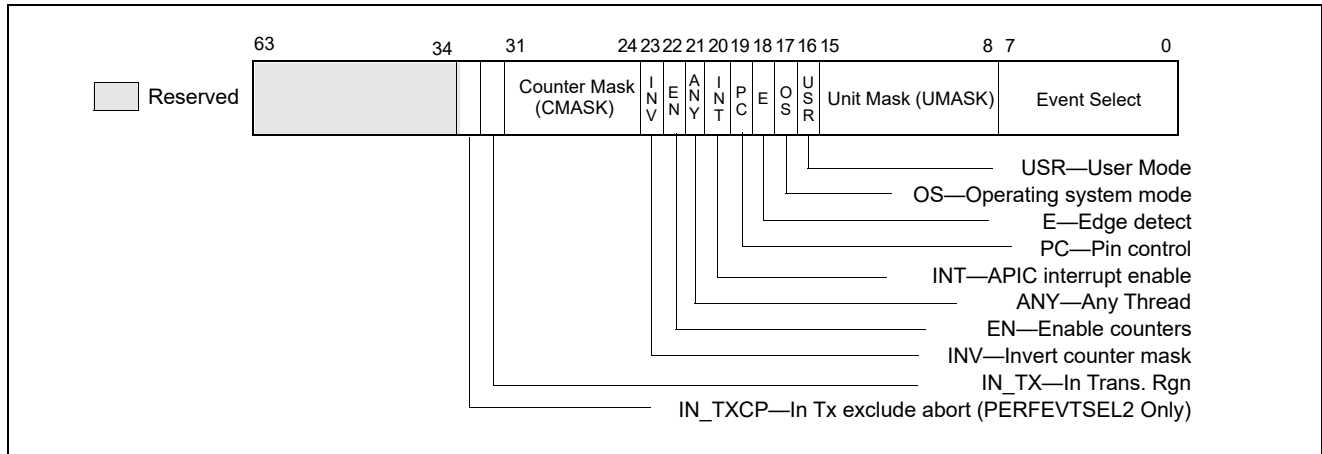


Figure 20-34. Layout of IA32_PERFEVTSELx MSRs Supporting Intel TSX

A common usage of setting IN_TXCP=1 is to capture the number of events that were discarded due to a transactional abort. With IA32_PMC2 configured to count in such a manner, then when a transactional region aborts, the value for that counter is restored to the value it had prior to the aborted transactional region. As a result, any updates performed to the counter during the aborted transactional region are discarded.

On the other hand, setting IN_TX=1 can be used to drill down on the performance characteristics of transactional code regions. When a PMCx is configured with the corresponding IA32_PERFEVTSELx.IN_TX=1, only eventing conditions that occur inside transactional code regions are propagated to the event logic and reflected in the counter result. Eventing conditions specified by IA32_PERFEVTSELx but occurring outside a transactional region are discarded.

Additionally, a number of performance events are solely focused on characterizing the execution of Intel TSX transactional code, they can be found at: <https://perfmon-events.intel.com/>.

20.3.6.5.1 Intel® TSX and PEBS Support

If a PEBS event would have occurred inside a transactional region, then the transactional region first aborts, and then the PEBS event is processed.

Two of the TSX performance monitoring events also support using the PEBS facility to capture additional information. They are:

- HLE_RETIREDA.BORTED (encoding C8H mask 04H),
- RTM_RETIREDA.BORTED (encoding C9H mask 04H).

A transactional abort (HLE_RETIREDA.BORTED,RTM_RETIREDA.BORTED) can also be programmed to cause PEBS events. In this scenario, a PEBS event is processed following the abort.

Pending a PEBS record inside of a transactional region will cause a transactional abort. If a PEBS record was pended at the time of the abort or on an overflow of the TSX PEBS events listed above, only the following PEBS entries will be valid (enumerated by PEBS entry offset B8H bits[33:32] to indicate an HLE abort or an RTM abort):

- Offset B0H: EventingIP,
- Offset B8H: TX Abort Information

These fields are set for all PEBS events.

- Offset 08H (RIP/EIP) corresponds to the instruction following the outermost XACQUIRE in HLE or the first instruction of the fallback handler of the outermost XBEGIN instruction in RTM. This is useful to identify the aborted transactional region.

In the case of HLE, an aborted transaction will restart execution deterministically at the start of the HLE region. In the case of RTM, an aborted transaction will transfer execution to the RTM fallback handler.

The layout of the TX Abort Information field is given in Table 20-31.

Table 20-31. TX Abort Information Field Definition

Bit Name	Offset	Description
Cycles_Last_TX	31:0	The number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
HLE_Abort	32	If set, the abort information corresponds to an aborted HLE execution
RTM_Abort	33	If set, the abort information corresponds to an aborted RTM execution
Instruction_Abort	34	If set, the abort was associated with the instruction corresponding to the eventing IP (offset OBOH) within the transactional region.
Non_Instruction_Abort	35	If set, the instruction corresponding to the eventing IP may not necessarily be related to the transactional abort.
Retry	36	If set, retrying the transactional execution may have succeeded.
Data_Conflict	37	If set, another logical processor conflicted with a memory address that was part of the transactional region that aborted.
Capacity Writes	38	If set, the transactional region aborted due to exceeding resources for transactional writes.
Capacity Reads	39	If set, the transactional region aborted due to exceeding resources for transactional reads.
In_Suspend	40	Transaction was aborted while in a suspend region. This is an Intel Xeon processor only feature, available beginning with 4th generation Intel Xeon Scalable Processor Family; otherwise reserved.
Reserved	63:41	Reserved

20.3.6.6 Uncore Performance Monitoring Facilities in the 4th Generation Intel® Core™ Processors

The uncore sub-system in the 4th Generation Intel® Core™ processors provides its own performance monitoring facility. The uncore PMU facility provides dedicated MSR to select uncore performance monitoring events in a similar manner as those described in Section 20.3.4.6.

The ARB unit and each C-Box provide local pairs of event select MSR and counter register. The layout of the event select MSRs in the C-Boxes are identical as shown in Figure 20-32.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 20-33 shows the layout of the uncore domain global control.

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 20-19 summarizes the number MSRs for uncore PMU for each box.

Table 20-32. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, see Table 2-21 MSR_UNC_CBO_CONFIG
ARB	1	2	44	Yes	Uncore	
Fixed Counter	N.A.	N.A.	48	No	Uncore	

The uncore performance events for the C-Box and ARB units can be found at: <https://perfmon-events.intel.com/>.

20.3.6.7 Intel® Xeon® Processor E5 v3 Family Uncore Performance Monitoring Facility

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v3 families are available in “Intel® Xeon® Processor E5 v3 Uncore Performance Monitoring Programming Reference Manual”. The MSR-based uncore PMU interfaces are listed in Table 2-33.

20.3.7 5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility

The 5th Generation Intel® Core™ processor and the Intel® Core™ M processor families are based on the Broadwell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 20.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 20.2.3.

The core PMU has the same capability as those described in Section 20.3.6. IA32_PERF_GLOBAL_STATUS provide a bit indicator (bit 55) for PMI handler to distinguish PMI due to output buffer overflow condition due to accumulating packet data from Intel Processor Trace.

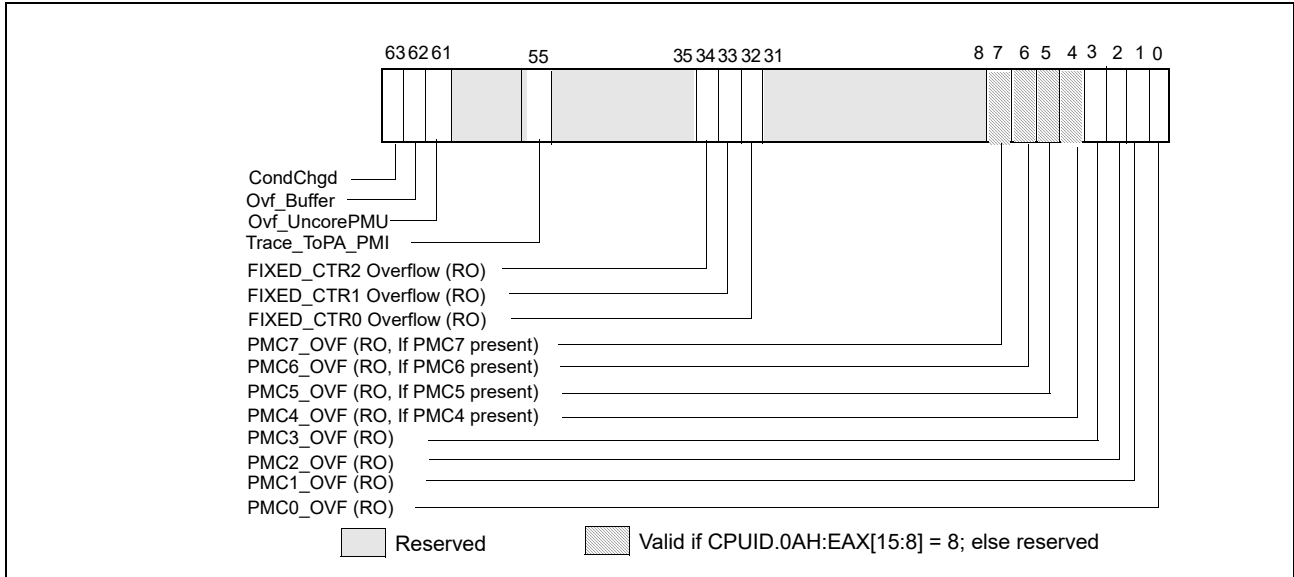


Figure 20-35. IA32_PERF_GLOBAL_STATUS MSR in Broadwell Microarchitecture

Details of Intel Processor Trace is described in Chapter 33, “Intel® Processor Trace”. The IA32_PERF_GLOBAL_OVF_CTRL MSR provides a corresponding reset control bit.

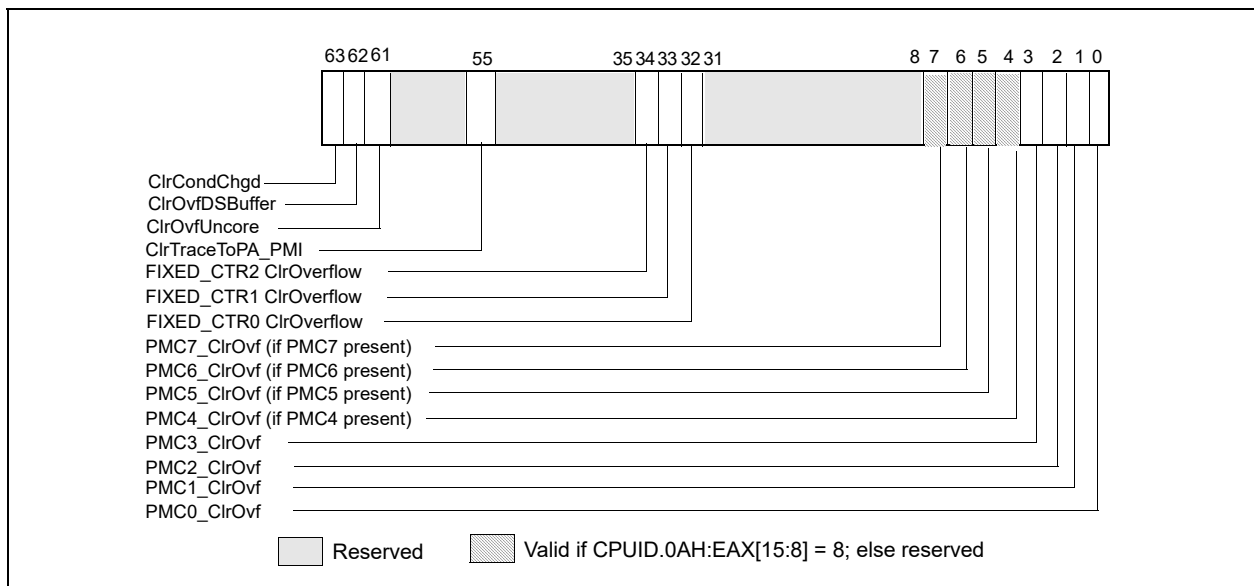


Figure 20-36. IA32_PERF_GLOBAL_OVF_CTRL MSR in Broadwell microarchitecture

The specifics of non-architectural performance events can be found at: <https://perfmon-events.intel.com/>.

20.3.8 6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility

The 6th generation Intel® Core™ processor is based on the Skylake microarchitecture. The 7th generation Intel® Core™ processor is based on the Kaby Lake microarchitecture. The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture. For these microarchitectures, the core PMU supports architectural performance monitoring capability with version ID 4 (see Section 20.2.4) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 4 capabilities are described in Section 20.2.4.

The core PMU's capability is similar to those described in Section 20.6.3 through Section 20.3.4.5, with some differences and enhancements summarized in Table 20-33. Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 20.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® Transactional Synchronization Extensions" of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

Performance monitoring result may be affected by side-band activity on processors that support Intel SGX, details are described in Chapter 40, "Enclave Code Debug and Profiling".

Table 20-33. Core PMU Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 20.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 20.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 20.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 20.2.1.
Architectural Perfmon version	4	3	See Section 20.2.4
PMI Overhead Mitigation	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with streamlined semantics. ▪ Freeze_LBR_on_PMI with streamlined semantics. ▪ Freeze_while_SMM. 	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_LBR_on_PMI with legacy semantics for branch profiling. ▪ Freeze_while_SMM. 	See Section 18.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	<ul style="list-style-type: none"> ▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_STATUS_RESET ▪ Set via IA32_PERF_GLOBAL_STATUS_SET 	<ul style="list-style-type: none"> ▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_OVF_CTRL 	See Section 20.2.4.

Table 20-33. Core PMU Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	<ul style="list-style-type: none"> Individual counter overflow PEBS buffer overflow ToPA buffer overflow CTR_Frz, LBR_Frz, ASCI 	<ul style="list-style-type: none"> Individual counter overflow PEBS buffer overflow ToPA buffer overflow (applicable to Broadwell microarchitecture) 	See Section 20.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	<ul style="list-style-type: none"> CTR_Frz LBR_Frz 	NA	See Section 20.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	NA	See Section 20.2.4.3.
Precise Events	See Table 20-36.	See Table 20-12.	IA32_PMC4-PMC7 do not support PEBS.
PEBS for front end events	See Section 20.3.8.2.	No	
LBR Record Format Encoding	000101b	000100b	Section 18.4.8.1
LBR Size	32 entries	16 entries	
LBR Entry	From_IP/To_IP/LBR_Info triplet	From_IP/To_IP pair	Section 18.12
LBR Timing	Yes	No	Section 18.12.1
Call Stack Profiling	Yes, see Section 18.11	Yes, see Section 18.11	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types.	MSR 1A6H and 1A7H; Extended request and response types.	
Intel TSX support for Perfmon	See Section 20.3.6.5.	See Section 20.3.6.5.	

20.3.8.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 6th generation, 7th generation and 8th generation Intel Core processors provides a number enhancement relative to PEBS in processors based on Haswell/Broadwell microarchitectures. The key components and differences of PEBS facility relative to Haswell/Broadwell microarchitecture is summarized in Table 20-34.

Table 20-34. PEBS Facility Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 20.3.1.1.1	Section 20.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 20-15	Figure 20-15	
PEBS-EventingIP	Yes	Yes	
PEBS record format encoding	0011b	0010b	
PEBS record layout	Table 20-35; enhanced fields at offsets 98H- B8H; and TSC record field at C0H.	Table 20-24; enhanced fields at offsets 98H, A0H, A8H, B0H.	
Multi-counter PEBS resolution	PEBS record 90H resolves the eventing counter overflow.	PEBS record 90H reflects IA32_PERF_GLOBAL_STATUS.	
Precise Events	See Table 20-36.	See Table 20-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.

Table 20-34. PEBS Facility Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS-Load Latency	See Section 20.3.4.4.2.	See Section 20.3.4.4.2.	
Data Address Profiling	Yes	Yes	
FrontEnd event support	FrontEnd_Retried event and MSR_PEBS_FRONTEND.	No	IA32_PMC0-PMC3 only.

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTES

Precise events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

20.3.8.1.1 PEBS Data Format

The PEBS record format for the 6th generation, 7th generation and 8th generation Intel Core processors is reporting with encoding 0011b in IA32_PERF_CAPABILITIES[11:8]. The lay out is shown in Table 20-35. The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 20-35. PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counter
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Data Source Encoding
40H	R/EBP	A8H	Latency value (core cycles)
48H	R/ESP	B0H	EventingIP
50H	R8	B8H	TX Abort Information (Section 20.3.6.5.1)
58H	R9	C0H	TSC
60H	R10		

The layout of PEBS records are largely identical to those shown in Table 20-24.

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 20.3.4.4.2), PDIR (Section 20.3.4.4.4), and data address profiling (Section 20.3.6.3).

In the core PMU of the 6th generation, 7th generation and 8th generation Intel Core processors, load latency facility and PDIR capabilities and data address profiling are unchanged relative to the 4th generation and 5th generation Intel Core processors. Similarly, precise store is replaced by data address profiling.

With format 0010b, a snapshot of the IA32_PERF_GLOBAL_STATUS may be useful to resolve the situations when more than one of IA32_PMICx have been configured to collect PEBS data and two consecutive overflows of the PEBS-enabled counters are sufficiently far apart in time. It is also possible for the image at 90H to indicate multiple PEBS-enabled counters have overflowed. In the latter scenario, software cannot to correlate the PEBS record entry to the multiple overflowed bits.

With PEBS record format encoding 0011b, offset 90H reports the “applicable counter” field, which is a multi-counter PEBS resolution index allowing software to correlate the PEBS record entry with the eventing PEBS overflow when multiple counters are configured to record PEBS records. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

20.3.8.1.2 PEBS Events

The list of precise events supported for PEBS in the Skylake, Kaby Lake and Coffee Lake microarchitectures is shown in Table 20-36.

Table 20-36. Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST ¹	01H
		ALL_CYCLES ²	01H
OTHER_ASSISTS	C1H	ANY	3FH
BR_INST_RETIRED	C4H	CONDITIONAL	01H
		NEAR_CALL	02H
		ALL_BRANCHES	04H
		NEAR_RETURN	08H
		NEAR_TAKEN	20H
		FAR_BRACHES	40H
BR_MISP_RETIRED	C5H	CONDITIONAL	01H
		ALL_BRANCHES	04H
		NEAR_TAKEN	20H
FRONTEND_RETIRED	C6H	<Programmable ³ >	01H
HLE_RETIRED	C8H	ABORTED	04H
RTM_RETIRED	C9H	ABORTED	04H
MEM_INST_RETIRED ²	D0H	LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H

Table 20-36. Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_LOAD_RETIRED ⁴	D1H	L1_HIT	01H
		L2_HIT	02H
		L3_HIT	04H
		L1_MISS	08H
		L2_MISS	10H
		L3_MISS	20H
		HIT_LFB	40H
MEM_LOAD_L3_HIT_RETIRED ²	D2H	XSNP_MISS	01H
		XSNP_HIT	02H
		XSNP_HITM	04H
		XSNP_NONE	08H

NOTES:

1. Only available on IA32_PMC1.
2. INST_RETIRED.ALL_CYCLES is configured with additional parameters of cmask = 10 and INV = 1
3. Subevents are specified using MSR_PEBS_FRONTEND, see Section 20.3.8.3
4. Instruction with at least one load uop experiencing the condition specified in the UMask.

20.3.8.1.3 Data Address Profiling

The PEBS Data address profiling on the 6th generation, 7th generation and 8th generation Intel Core processors is largely unchanged from the prior generation. When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H, and A8H, as shown in Table 20-26.

Table 20-37. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	<ul style="list-style-type: none"> ▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_INST_RETIRED.STLB_MISS_STORES, MEM_INST_RETIRED.ALL_STORES, MEM_INST_RETIRED.SPLIT_STORES. ▪ Other bits are zero.
Reserved	A8H	Always zero.

20.3.8.2 Frontend Retired Facility

The Skylake Core PMU has been extended to cover common microarchitectural conditions related to the front end pipeline in addition to providing a generic latency mechanism that can locate fetch bubbles without necessarily attributing them to a particular condition. The facility counts the events if the associated instruction reaches retirement (architecturally committed). Additionally, the user may opt to enable the PEBS facility to obtain precise information on the context of the event, e.g., EventingIP.

The supported frontend microarchitectural conditions require the following interfaces:

- The IA32_PERFEVTSELx MSR must select the FRONTEND_RETIRED event, EventSelect = C6H and UMASK = 01H.

- This event employs a new MSR, MSR_PEBS_FRONTEND, to specify the supported frontend event details, see Table 20-38.
 - If precise information is desired, program the PEBS_EN_PMCx field of IA32_PEBS_ENABLE MSR as required.
- Note the AnyThread field of IA32_PERFEVTSELx is ignored by the processor for the “FRONTEND_RETIRED” event. The sub-event encodings supported by MSR_PEBS_FRONTEND.EVTSEL is given in Table 20-38.

Table 20-38. FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL

Sub-Event Name	EVTSEL	Description
ANY_DSB_MISS	1H	Retired Instructions which experienced any decode stream buffer (DSB) miss.
DSB_MISS	11H	Retired Instructions which experienced a DSB miss that caused a fetch starvation cycle.
L11_MISS	12H	The fetch of retired Instructions which experienced Instruction L1 Cache true miss ¹ . Additional requests to the same cache line as an in-flight L11 cache miss will not be counted.
L2_MISS	13H	The fetch of retired Instructions which experienced L2 Cache true miss. Additional requests to the same cache line as an in-flight MLC cache miss will not be counted.
ITLB_MISS	14H	The fetch of retired Instructions which experienced ITLB true miss. Additional requests to the same cache line as an in-flight ITLB miss will not be counted.
STLB_MISS	15H	The fetch of retired Instructions which experienced STLB true miss. Additional requests to the same cache line as an in-flight STLB miss will not be counted.
IDQ_READ_BUBBLES	6H	An IDQ read bubble is defined as any one of the 4 allocation slots of IDQ that is not filled by the front-end on any cycle where there is no back end stall. Using the threshold and latency fields in MSR_PEBS_FRONTEND allows counting of IDQ read bubbles of various magnitude and duration. Latency controls the number of cycles and Threshold controls the number of allocation slots that contain bubbles. The event counts if and only if a sequence of at least FE_LATENCY consecutive cycles contain at least FE_TRESHOLD number of bubbles each.

NOTES:

1. A true miss is the first miss for a cacheline/page (excluding secondary misses that fall into same cacheline/page).

The layout of MSR_PEBS_FRONTEND is given in Table 20-39.

Table 20-39. MSR_PEBS_FRONTEND Layout

Bit Name	Offset	Description
EVTSEL	7:0	Encodes the sub-event within FrontEnd_Retired that can use PEBS facility, see Table 20-38.
IDQ_Bubble_Length	19:8	Specifies the threshold of continuously elapsed cycles for the specified width of bubbles when counting IDQ_READ_BUBBLES event.
IDQ_Bubble_Width	22:20	Specifies the threshold of simultaneous bubbles when counting IDQ_READ_BUBBLES event.
Reserved	63:23	Reserved

The FRONTEND_RETIRED event is designed to help software developers identify exact instructions that caused front-end issues. There are some instances in which the event will, by design, the under-counting scenarios include the following:

- The event counts only retired (non-speculative) front-end events, i.e., events from just true program execution path are counted.
- The event will count once per cacheline (at most). If a cacheline contains multiple instructions which caused front-end misses, the count will be only 1 for that line.
- If the multibyte sequence of an instruction spans across two cachelines and causes a miss it will be recorded once. If there were additional misses in the second cacheline, they will not be counted separately.

- If a multi-uop instruction exceeds the allocation width of one cycle, the bubbles associated with these uops will be counted once per that instruction.
- If 2 instructions are fused (macro-fusion), and either of them or both cause front-end misses, it will be counted once for the fused instruction.
- If a front-end (miss) event occurs outside instruction boundary (e.g., due to processor handling of architectural event), it may be reported for the next instruction to retire.

20.3.8.3 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 20.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 20-40.
- Supplier information (bits 29:16): see Table 20-41.
- Snoop response information (bits 37:30): see Table 20-42.

**Table 20-40. MSR_OFFCORE_RSP_x Request_Type Definition
(Skylake, Kaby Lake, and Coffee Lake Microarchitectures)**

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DMND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	14:3	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and uncacheable accesses.

Table 20-41 lists the supplier information field that applies to 6th generation, 7th generation and 8th generation Intel Core processors. (6th generation Intel Core processor CPUID signatures: 06_4EH and 06_5EH; 7th generation and 8th generation Intel Core processor CPUID signatures: 06_8EH and 06_9EH).

**Table 20-41. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(CPUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	L4_HIT	22	L4 Cache (if L4 is present in the processor).
	Reserved	25:23	Reserved
	DRAM	26	Local Node
	Reserved	29:27	Reserved
	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).

Table 20-42 lists the snoop information field that apply to processors with CPUID signatures 06_4EH, 06_5EH, 06_8EH, 06_9E, and 06_55H.

**Table 20-42. MSR_OFFCORE_RSP_x Snoop Info Field Definition
(CPUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9E, 06_55H)**

Subtype	Bit Name	Offset	Description
Snoop Info	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).
	SNOOP_NONE	31	No details on snoop-related information.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores. -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNOOP_HIT_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO). -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD). -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S). In the LLC Miss case, data is returned from DRAM.
	SNOOP_HIT_WITH_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	SNOOP_HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD). -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO). -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	SNOOP_NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

20.3.8.3.1 Off-core Response Performance Monitoring for the Intel® Xeon® Scalable Processor Family

The following tables list the requestor and supplier information fields that apply to the Intel® Xeon® Scalable Processor Family.

- Transaction request type encoding (bits 15:0): see Table 20-43.
- Supplier information (bits 29:16): see Table 20-44.
- Supplier information (bits 29:16) with support for Intel® Optane™ DC Persistent Memory support: see Table 20-45.
- Snoop response information has not been changed and is the same as in (bits 37:30): see Table 20-42.

Table 20-43. MSR_OFFCORE_RSP_x Request_Type Definition (Intel® Xeon® Scalable Processor Family)

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DEMAND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DEMAND_CODE_RD	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	3	Reserved.
PF_L2_DATA_RD	4	Counts the number of prefetch data reads into L2.
PF_L2_RFO	5	Counts the number of RFO Requests generated by the MLC prefetches to L2.
Reserved	6	Reserved.
PF_L3_DATA_RD	7	Counts the number of MLC data read prefetches into L3.
PF_L3_RFO	8	Counts the number of RFO requests generated by MLC prefetches to L3.
Reserved	9	Reserved.
PF_L1D_AND_SW	10	Counts data cacheline reads generated by hardware L1 data cache prefetcher or software prefetch requests.
Reserved	14:11	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

Table 20-44 lists the supplier information field that applies to the Intel Xeon Scalable Processor Family (CPUID signature: 06_55H).

Table 20-44. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	Reserved	25:22	Reserved
	L3_MISS_LOCAL_DRAM	26	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
Reserved	30	Reserved	

Table 20-45 lists the supplier information field that applies to the Intel Xeon Scalable Processor Family (CPUID signature: 06_55H, Steppings 0x5H - 0xFH).

**Table 20-45. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(CPUID Signature: 06_55H, Steppings 0x5H - 0xFH)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	LOCAL_PMM	22	Local home requests that were serviced by local PMM.
	REMOTE_HOP0_PMM	23	Hop 0 Remote supplier.
	REMOTE_HOP1_PMM	24	Hop 1 Remote supplier.
	REMOTE_HOP2P_PMM	25	Hop 2 or more Remote supplier.
	L3_MISS_LOCAL_DRAM	26	L3 Miss: Local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
Reserved		30	Reserved

20.3.8.4 Uncore Performance Monitoring Facilities on Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Cannon Lake microarchitecture introduces LLC support of up to six processor cores. To support six processor cores and eight LLC slices, existing MSR addresses have been rearranged and new CBo MSR addresses have been added. Uncore performance monitoring software drivers from prior generations of Intel Core processors will need to update the MSR addresses. The new MSR addresses and updated MSR addresses have been added to the Uncore PMU listing in Section 2.17.2, “MSRs Specific to 8th Generation Intel® Core™ i3 Processors” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

20.3.9 10th Generation Intel® Core™ Processor Performance Monitoring Facility

Some 10th generation Intel® Core™ processors and some 3rd generation Intel® Xeon® Scalable Processor Family are based on Ice Lake microarchitecture. Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture. For these processors, the core PMU supports architectural performance monitoring capability with version Id 5 (see Section 20.2.5) and a host of non-architectural monitoring capabilities.

The core PMU's capability is similar to those described in Section 20.3.1 through Section 20.3.8, with some differences and enhancements summarized in Table 20-46.

Table 20-46. Core PMU Summary of the Ice Lake Microarchitecture

Box	Ice Lake Microarchitecture	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Comment
Architectural Perfmon version	5	4	See Section 20.2.5.
Number of programmable counters per thread	8	4	Use CPUID to determine number of counters. See Section 20.2.1.
PEBS: Basic functionality	Yes	Yes	See Section 20.3.9.1.
PEBS record format encoding	0100b	0011b	See Section 20.6.2.4.2.
Extended PEBS	PEBS is extended to all Fixed and General Purpose counters and to all performance monitoring events.	No	See Section 20.9.1.
Adaptive PEBS	Yes	No	See Section 20.9.2.
Performance Metrics	Yes (4)	No	See Section 20.3.9.3.
PEBS-PDIR	IA32_FIXED0 only (Corresponding counter control MSRs must be enabled.)	IA32_PMC1 only.	

20.3.9.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 10th generation Intel Core processors provides a number of enhancements relative to PEBS in processors based on the Skylake, Kaby Lake, and Coffee Lake microarchitectures. Enhancement of the PEBS facility with Extended PEBS and Adaptive PEBS features is described in detail in Section 20.9.

The 3rd generation Intel Xeon Scalable Family of processors based on the Ice Lake microarchitecture introduce EPT-friendly PEBS. This allows EPT violations and other VM Exits to be taken on PEBS accesses to the DS Area. See Section 20.9.5 for details.

20.3.9.2 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 20.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 18-[N1].
- Response type encoding (bits 16-37) of
 - Supplier information: see Table [18-N2].
 - Snoop response information: see Table [18-N3].
- All transactions are tracked at cacheline granularity except some in request type OTHER.

**Table 20-47. MSR_OFFCORE_RSP_x Request_Type Definition
(Processors Based on Ice Lake Microarchitecture)**

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data and page table entry reads.
DEMAND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DEMAND_CODE_RD	2	Counts demand instruction fetches and instruction prefetches targeting the L1 instruction cache.
Reserved	3	Reserved

**Table 20-47. MSR_OFFCORE_RSP_x Request_Type Definition
(Processors Based on Ice Lake Microarchitecture)**

Bit Name	Offset	Description
HWPF_L2_DATA_RD	4	Counts hardware generated data read prefetches targeting the L2 cache.
HWPF_L2_RFO	5	Counts hardware generated prefetches for exclusive ownership (RFO) targeting the L2 cache.
Reserved	6	Reserved
HWPF_L3	9:7 and 13 ¹	Counts hardware generated prefetches of any type targeting the L3 cache.
HWPF_L1D_AND_SWPF	10	Counts hardware generated data read prefetches targeting the L1 data cache and the following software prefetches (PREFETCHNTA, PREFETCHT0/1/2).
STREAMING_WR	11	Counts streaming stores.
Reserved	12	Reserved
Reserved	14	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

NOTES:

1. All bits need to be set to 1 to count this type.

Ice Lake microarchitecture has added a new category of Response subtype, called a Combined Response Info. To count a feature in this type, all the bits specified must be set to 1.

A valid response type must be a non-zero value of the following expression:

Any | ['OR' of Combined Response Info Bits | (('OR' of Supplier Info Bits) & ('OR' of Snoop Info Bits)]]

If "ANY" bit[16] is set, other response type bits [17-39] are ignored.

Table 20-48 lists the supplier information field that applies to processors based on Ice Lake microarchitecture.

**Table 20-48. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Combined Response Info	DRAM	26, 31, 32 ¹	Requests that are satisfied by DRAM.
	NON_DRAM	26, 37 ¹	Requests that are satisfied by a NON_DRAM system component. This includes MMIO transactions.
	L3_MISS	22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 ¹	Requests that were not supplied by the L3 Cache. The event includes some currently reserved bits in anticipation of future memory designs.
Supplier Info	L3_HIT	18,19, 20 ¹	Requests that hit in L3 cache. Depending on the snoop response the L3 cache may have retrieved the cacheline from another core's cache.
Reserved		17, 21:25, 27:29	Reserved.

NOTES:

1. All bits need to be set to 1 to count this type.

Table 20-49 lists the snoop information field that applies to processors based on Ice Lake microarchitecture.

**Table 20-49. MSR_OFFCORE_RSP_x Snoop Info Field Definition
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Snoop Info	Reserved	30	Reserved.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was sent and none of the snooped caches contained the cacheline.
	SNOOP_HIT_NO_FWD	34	A snoop was sent and hit in at least one snooped cache. The unmodified cacheline was not forwarded back, because the L3 already has a valid copy.
	Reserved	35	Reserved.
	SNOOP_HITM	36	A snoop was sent and the cacheline was found modified in another core's caches. The modified cacheline was forwarded to the requesting core.

20.3.9.3 Performance Metrics

The Ice Lake core PMU provides built-in support for Top-down Microarchitecture Analysis (TMA) method level 1 metrics. These metrics are always available to cross-validate performance observations, freeing general purpose counters to count other events in high counter utilization scenarios. For more details about the method, refer to Top-Down Analysis Method chapter (Appendix B.1) of the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

A new MSR called MSR_PERF_METRICS reports the metrics directly. Software can check (and/or expose to its guests) the availability of the PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15). For additional details on this MSR, refer to Chapter 2, "Model-Specific Registers (MSRs)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

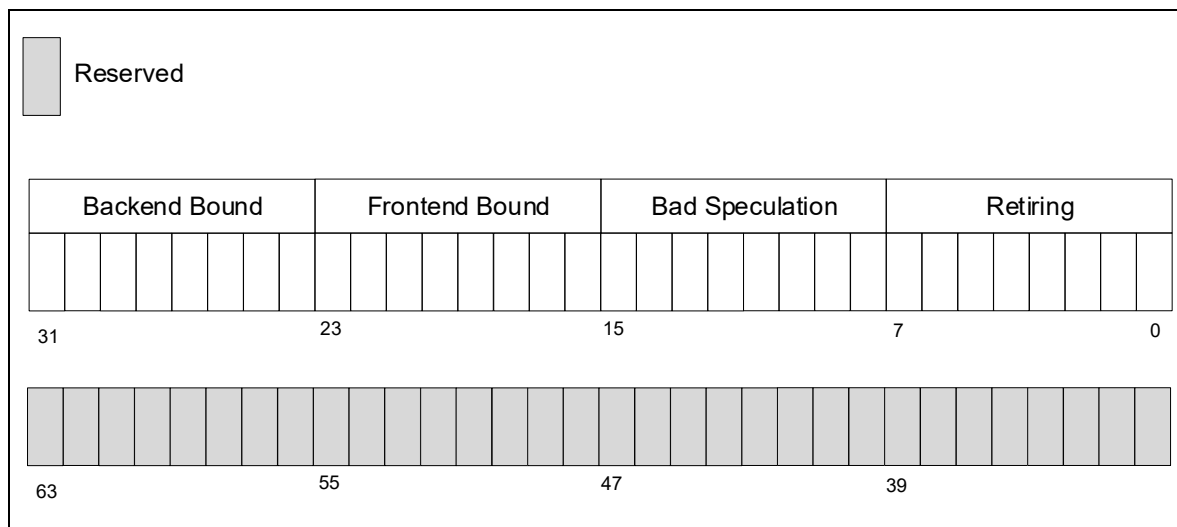


Figure 20-37. MSR_PERF_METRICS Definition

This register exposes the four TMA Level 1 metrics. The lower 32 bits are divided into four 8-bit fields, as shown by the above figure, each of which is an integer fraction of 255.

To support built-in performance metrics, new bits have been added to the following MSRs:

- IA32_PERF_GLOBAL_CTRL. EN_PERF_METRICS[48]: If this bit is set and fixed-function performance-monitoring counter 3 is enabled, built-in performance metrics are enabled.
- IA32_PERF_GLOBAL_STATUS_SET. SET_OVF_PERF_METRICS[48]: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS_RESET. RESET_OVF_PERF_METRICS[48]: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS. OVF_PERF_METRICS[48]: If this bit is set, it indicates that a PERF_METRICS-related resource has overflowed and a PMI is triggered¹. If this bit is clear, no such overflow has occurred.

NOTE

Software has to synchronize, e.g., re-start, fixed-function performance-monitoring counter 3 as well as PERF_METRICS when either bit 35 or 48 in IA32_PERF_GLOBAL_STATUS is set. Otherwise, PERF_METRICS may return undefined values.

The values in MSR_PERF_METRICS are derived from fixed-function performance-monitoring counter 3. Software should start both registers, PERF_METRICS and fixed-function performance-monitoring counter 3, from zero. Additionally, software is recommended to periodically clear both registers in order to maintain accurate measurements for certain scenarios that involve sampling metrics at high rates.

In order to save/restore PERF_METRICS, software should follow these guidelines:

- PERF_METRICS and fixed-function performance-monitoring counter 3 should be saved and restored together.
- To ensure that PERF_METRICS and fixed-function performance-monitoring counter 3 remain synchronized, both should be disabled during both save and restore. Software should enable/disable them atomically, with a single write to IA32_PERF_GLOBAL_CTRL to set/clear both EN_PERF_METRICS[bit 48] and EN_FIXED_CTR3[bit 35].
- On state restore, fixed-function performance-monitoring counter 3 must be restored **before** PERF_METRICS, otherwise undefined results may be observed.

20.3.10 12th and 13th Generation Intel® Core™ Processors, and 4th Generation Intel® Xeon® Scalable Processor Family Performance Monitoring Facility

The 12th generation Intel® Core™ processor supports Alder Lake performance hybrid architecture. These processors offer a unique combination of Performance and Efficient-cores (P-core and E-core). The P-core is based on Golden Cove microarchitecture and the E-core is based on Gracemont microarchitecture. The 13th generation Intel® Core™ processor supports Raptor Lake performance hybrid architecture, utilizing both Raptor Cove cores and enhanced Gracemont cores. The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture utilizing Golden Cove cores. These processors all report architectural performance monitoring version ID = 5 and support non-architectural monitoring capabilities described in this section.

20.3.10.1 P-core Performance Monitoring Unit

The P-core PMU's capability is similar to those described in Section 20.3.1 through Section 20.3.9, with some differences and enhancements summarized in Table 20-50.

1. An overflow of fixed-function performance-monitoring counter 3 should normally happen first if software follows Intel's recommendations.

Table 20-50. Core PMU Summary of the Golden Cove Microarchitecture

Box	Golden Cove Microarchitecture	Ice Lake Microarchitecture	Comment
Architectural Perfmon version	5	5	See Section 20.2.5.
Event-Counter Restrictions	Simplified identification		Counters 4-7 support a subset of events. See Section 20.3.10.1.2.
Performance Metrics	Yes (12)	Yes (4)	See Section 20.3.9.3.
PEBS: Baseline, record format	Yes 0100b	Yes 0100b	See Section 20.3.9.
PEBS: EPT-friendly	Yes	No; debuts in Ice Lake server microarchitecture	See Section 20.6.2.4.2.
PEBS: Precise Distribution	IA32_FIXED0 instruction-granularity PDist on IA32_PMC0	IA32_FIXED0 cycle-granularity No PDist	See Section 20.9.6.
PEBS: Load Latency	Instruction latency Cache latency Access info fields (5)	Instruction latency Access info fields (3)	See Section 20.9.7.
PEBS: Store Latency	Cache latency Access info fields (3)	None	See Section 20.9.8.
PEBS: Intel TSX support	Abort info fields (9)	Abort info fields (8)	See Section 20.3.6.5.1. (Intel Xeon processor only feature.)

20.3.10.1.1 P-core Perf Metrics Extensions

For 12th generation Intel Core processor P-cores, the core PMU supports the built-in metrics that were introduced in the Ice Lake microarchitecture PMU. This core PMU extends the PERF_METRICS MSR to feature TMA method level 2 metrics, as shown in Figure 20-38.

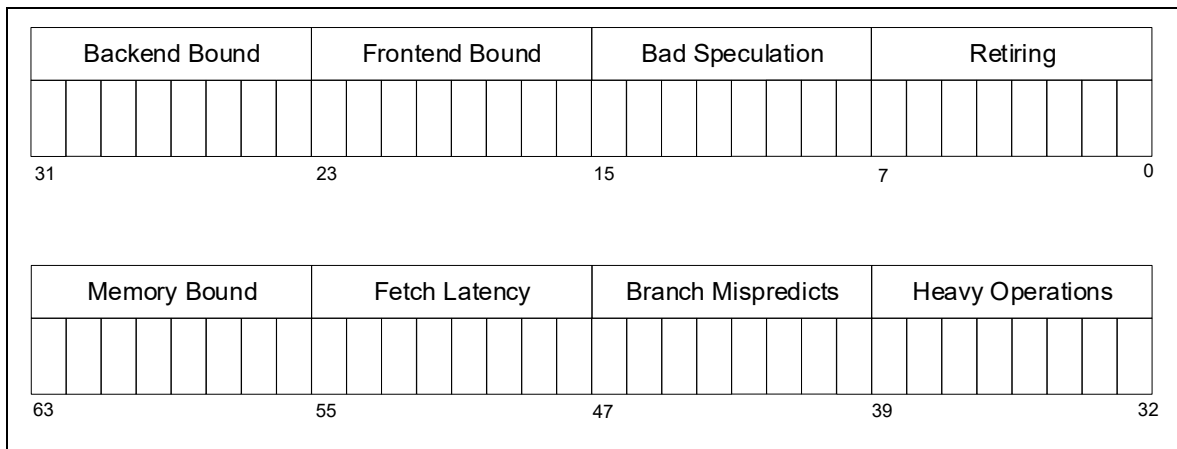


Figure 20-38. PERF_METRICS MSR Definition for 12th Generation Intel® Core™ Processor P-core

The lower half of the register is the TMA level 1 metrics (legacy). The upper half is also divided into four 8-bit fields, each of which is an integer fraction of 255. Additionally, each of the new level 2 metrics in the upper half is a subset of the corresponding level 1 metric in the lower half (that is, its parent node per the TMA hierarchy). This enables software to deduce the other four level 2 metrics by subtracting corresponding metrics as shown in Figure 20-39.

$\begin{aligned} \text{Light_Operations} &= \text{Retiring} - \text{Heavy_Operations} \\ \text{Machine_Clears} &= \text{Bad_Speculation} - \text{Branch_Mispredicts} \\ \text{Fetch_Bandwidth} &= \text{Frontend_Bound} - \text{Fetch_Latency} \\ \text{Core_Bound} &= \text{Backend_Bound} - \text{Memory_Bound} \end{aligned}$

Figure 20-39. Deducing Implied Level 2 Metrics in the Core PMU for 12th Generation Intel® Core™ Processor P-core

The PERF_METRICS MSR and fixed-function performance-monitoring counter 3 of the core PMU feature 12 metrics in total that cover all level 1 and level 2 nodes of the TMA hierarchy.

20.3.10.1.2 P-core Counter Restrictions Simplification

The 12th generation Intel Core processor P-core allows identification of performance monitoring events with counter restrictions based on event encodings. The general rule is: Event Codes < 0x90 are restricted to general-purpose performance-monitoring counters 0-3. Event Codes ≥ 0x90 are likely to have no restrictions. Table 20-51 lists the exceptions to this rule.

Table 20-51. Special Performance Monitoring Events with Counter Restrictions

Event Encoding ¹	Event Name	Counter Restriction
xx3C	CPU_CLK_UNHALTED.*	0-7 (No restriction for all architectural events.)
xx2E	LONGEST_LAT_CACHE.*	
xxDx	MEM*_RETIRED.*	0-3
01A3, 02A3, 08A3	Some CYCLE_ACTIVITY sub-events	0-3
02CD	MEM_TRANS_RETIRED.STORE_SAMPLE	0
04A4	TOPDOWN.BAD_SPEC_SLOTS	0
08A4	TOPDOWN.BR_MISPREDICT_SLOTS	
xxCE	AMX_OPS_RETIRED	0

NOTES:

1. Linux perf rUUEE syntax, where UU is the Unit Mask field and EE is the Event Select (also known as Event Code) field in the IA32_PERFEVTSELx MSRs.

20.3.10.1.3 P-core Off-core Response Facility

For the 12th generation Intel Core processor P-core, the Off-core Response (OCR) Facility is similar to that described in Section 20.3.9.2.

The following enhancements are introduced for the Request_Type of MSR_OFFCORE_RSP_x:

- WB (bits 3 and 12): Count writeback (modified or non-modified) transactions by core caches.
- HWPF_L1D (bit 10): Counts hardware generated data read prefetches targeting the L1 data cache (only).
- SWPF_READ (bit 14): Counts software generated data read prefetches by the PREFETCHNTA and PREFETCHT0/1/2 instructions.

20.3.10.2 E-core Performance Monitoring Unit

The core PMU capabilities on the 12th generation Intel Core processor E-core are summarized in Table 20-52 below.

Table 20-52. Core PMU Summary of the Gracemont Microarchitecture

Box	Gracemont Microarchitecture	Tremont Microarchitecture	Comment
Number of fixed-function performance-monitoring counters per core	3	3	Use CPUID to enumerate number of counters. See Section 20.2.1.
Number of general-purpose counters per core	6	4	Use CPUID to enumerate number of counters. See Section 20.2.1.
Architectural Performance Monitoring version ID	5	5	See Section 20.2.5.
PEBS record format encoding	0100b	0100b	See Section 20.5.5.
EPT-friendly PEBS support	Yes	No	See Section 20.9.5.
Extended PEBS	Yes	Yes	See Section 20.9.1.
Adaptive PEBS	Yes	Yes	See Section 20.9.2.
Precise distribution (PDist) PEBS	IA32_PMC0 and IA32_FIXED_CTRO	IA32_PMC0 and IA32_FIXED_CTRO	PDist eliminates skid, see Section 20.9.3, Section 20.9.4, and Section 20.9.6.
PEBS Latency	Load and Store Latency	No	See Section 20.3.10.2.1, Section 20.3.10.2.2, Section 20.9.7, and Section 20.9.8.
PEBS Output	DS Save Area or Intel® Processor Trace	DS Save Area or Intel® Processor Trace	See Section 20.5.5.2.1.
Offcore Response	MSR 01A6H and 01A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	See Section 20.5.5.4.

20.3.10.2.1 E-core PEBS Load Latency

The 12th generation Intel Core processor E-core includes PEBS Load Latency support similar to that described in Section 20.9.7.

When a programmable counter is configured to count MEM_UOPS_RETIRED.LOAD_LATENCY_ABOVE_THRESHOLD (IA32_PERFEVTSELx[15:0] = 0xD005, with CMASK=0 and INV=0), selected load operations whose latency exceeds the threshold provided in MSR_PEBS_LD_LAT_THRESHOLD (MSR 03F6H) will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 20.9.2.2.2). The formats of these fields are shown in Table 20-53 and Table 20-94.

Table 20-53. E-core PEBS Memory Access Info Encoding

Bit(s)	Field	Description
3:0	Data Source	The source of the data; see Table 20-54.
4	Lock	0: The operation was not part of a locked transaction. 1: The operation was part of a locked transaction.

Table 20-53. E-core PEBS Memory Access Info Encoding (Contd.)

Bit(s)	Field	Description
5	STLB_MISS	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
6	ST_FWD_BLK	0: Load did not get a store forward block. 1: Load got a store forward block.
63:7	Reserved	Reserved

For details on E-core PEBS memory access latency encoding, see the Access Latency Field in Table 20-94.

Table 20-54. E-core PEBS Data Source Encodings

Encoding	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request) and MMIO. Memory mapped I/O hit.
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. Outstanding core cache miss to same cache-line address was already underway. (Pending core cache hit.)
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HITE. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found (clean).
06H	L3 HITM. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a modified copy was found.
07H	Reserved.
08H	L3 HITF. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a shared or forwarding copy was found.
09H	Reserved.
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	Reserved.
0CH	Reserved.
0DH	Reserved.
0EH	I/O. Request of input/output operation.
0FH	The request was to un-cacheable memory.

20.3.10.2.2 E-core PEBS Store Latency

The 12th generation Intel Core processor E-core includes PEBS Store Latency support. When a programmable counter is configured to count MEM_UOPS_RETIRED.STORE_LATENCY (IA32_PERFEVTSELx[15:0] = 0xD006, with CMASK=0 and INV=0), all store operations will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 18.9.2.2.2). The formats of these fields are shown in Table 20-53 and Table 20-94.

20.3.10.2.3 E-core Precise Distribution (PDist) Support

The 12th generation Intel Core processor E-core supports PEBS with Precise Distribution (PDist) on IA32_PMC0 and IA32_FIXED_CTR0. All precise events support PDist save for UOPS_RETIRED. See Section 20.9.6 for additional details on PDist.

20.3.10.2.4 E-core Enhanced Off-core Response

Event number 0B7H support off-core response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. There are unique pairs of MSR_OFFCORE_RSPx registers per core. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 and bits 49:44 specify the request type of a transaction request to the uncore. This is described in Table 20-55.
- Bits 30:16 specify Response Type information or an L2 Hit, and is described in Table 20-75.
- If L2 misses, then bits 37:31 can be used to specify snoop response information and is described in Table 20-76.
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 20.5.2.3 for details.

Table 20-55. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPF_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPF_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPF_L2_CODE_RD	6	Counts all prefetch (that bring data to MLC only) code reads.
HWPF_L3_DATA_RD	7	Counts L3 cache hardware prefetch data reads (written to the L3 cache only).
HWPF_L3_RFO	8	Counts L3 cache hardware prefetch RFOs (written to the L3 cache only) .
HWPF_L3_CODE_RD	9	Counts L3 cache hardware prefetch code reads (written to the L3 cache only).
HWPF_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
RSVD	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WiL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (WCiL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (WCiLF).
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

20.3.10.3 Unhalted Reference Cycles

The Unhalted Reference Cycles architectural performance monitoring event is enhanced to count at TSC-rate in the 12th generation Intel Core processor P-core when used on a general-purpose PMC. This enhancement makes it consistent with the fixed-function counter 2 and the E-core. As a result, this event is kept enumerated in CPUID leaf 0AH.EBX (unlike prior hybrid parts).

20.4 PERFORMANCE MONITORING (INTEL® XEON™ PHI PROCESSORS)

NOTE

This section also applies to the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture.

20.4.1 Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring

The Intel® Xeon Phi™ processor 7200/5200/3200 series are based on the Knights Landing microarchitecture. The performance monitoring capabilities are distributed between its tiles (pair of processor cores) and untile (connecting many tiles in a physical processor package). Functional details of the tiles and untile of the Knights Landing microarchitecture can be found in Chapter 16 of Intel® 64 and IA-32 Architectures Optimization Reference Manual.

A complete description of the tile and untile PMU programming interfaces for Intel Xeon Phi processors based on the Knights Landing microarchitecture can be found in the Technical Document section at <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>.

A tile contains a pair of cores attached to a shared L2 cache and is similar to those found in Intel Atom® processors based on the Silvermont microarchitecture. The processor provides several new capabilities on top of the Silvermont performance monitoring facilities.

The processor supports architectural performance monitoring capability with version ID 3 (see Section 20.2.3) and a host of non-architectural performance monitoring capabilities. The processor provides two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

Non-architectural performance monitoring in the processor also uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 20-6 and described in Section 20.2.1.1 and Section 20.2.3. The processor supports AnyThread counting in three architectural performance monitoring events.

20.4.1.1 Enhancements of Performance Monitoring in the Intel® Xeon Phi™ Processor Tile

The Intel® Xeon Phi™ processor tile includes the following enhancements to the Silvermont microarchitecture.

- AnyThread support. This facility is limited to following three architectural events: Instructions Retired, Unhalted Core Cycles, Unhalted Reference Cycles using IA32_FIXED_CTR0-2 and Unhalted Core Cycles, Unhalted Reference Cycles using IA32_PERFEVTSELx.
- PEBS-DLA (Processor Event-Based Sampling-Data Linear Address) fields. The processor provides memory address in addition to the Silvermont PEBS record support on select events. The PEBS recording format as reported by IA32_PERF_CAPABILITIES [11:8] is 2.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor tile to subsystems outside the tile (untile). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx. Two cores do not share the off-core response MSRs. Knights Landing expands off-core response capability to match the processor untile changes.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests. This facility is updated to match the processor untile changes.

20.4.1.1.1 Processor Event-Based Sampling

The processor supports processor event based sampling (PEBS). PEBS is supported using IA32_PMC0 (see also Section 18.4.9, "BTS and DS Save Area").

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 20.6.2.4).

The list of PEBS events supported in the processor is shown in the following table.

Table 20-56. PEBS Performance Events for Knights Landing Microarchitecture

Event Name	Event Select	Sub-event	UMask	Data Linear Address Support
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		CALL	F9H	No
		REL_CALL	FDH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		FAR_BRANCH	BFH	No
		RETURN	F7H	No
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		RETURN	F7H	No
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H	Yes
		L2_MISS_LOADS	04H	Yes
		DLTB_MISS_LOADS	08H	Yes
RECYCLEQ	03H	LD_BLOCK_ST_FORWARD	01H	Yes
		LD_SPLITS	08H	Yes

The PEBS record format 2 supported by processors based on the Knights Landing microarchitecture is shown in Table 20-57, and each field in the PEBS record is 64 bits long.

Table 20-57. PEBS Record Format for Knights Landing Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	PSDLA
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP

Table 20-57. PEBS Record Format for Knights Landing Microarchitecture (Contd.)

Byte Offset	Field	Byte Offset	Field
58H	R9	B8H	Reserved

20.4.1.1.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 20-58 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 20-58. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMCO-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMCO-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

Some of the MSR_OFFCORE_RESP [0,1] register bits are not valid in this processor and their use is reserved. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 registers are defined in Table 20-59. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 20.5.2.3 for details.

Table 20-59. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers

Main	Sub-field	Bit	Name	Description
Request Type		0	DEMAND_DATA_RD	Demand cacheable data and L1 prefetch data reads.
		1	DEMAND_RFO	Demand cacheable data writes.
		2	DEMAND_CODE_RD	Demand code reads and prefetch code reads.
		3	Reserved	Reserved.
		4	Reserved	Reserved.
		5	PF_L2_RFO	L2 data RFO prefetches (includes PREFETCHW instruction).
		6	PF_L2_CODE_RD	L2 code HW prefetches.
		7	PARTIAL_READS	Partial reads (UC or WC).
		8	PARTIAL_WRITES	Partial writes (UC or WT or WP). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		9	UC_CODE_READS	UC code reads.
		10	BUS_LOCKS	Bus locks and split lock requests.
		11	FULL_STREAMING_STORES	Full streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		12	SW_PREFETCH	Software prefetches.
		13	PF_L1_DATA_RD	L1 data HW prefetches.
		14	PARTIAL_STREAMING_STORES	Partial streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
15	ANY_REQUEST	Account for any requests.		

Table 20-59. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers (Contd.)

Main	Sub-field	Bit	Name	Description	
Response Type	Any	16	ANY_RESPONSE	Account for any response.	
		Data Supply from Untile	17	NO_SUPP	No Supplier Details.
			18	Reserved	Reserved.
			19	L2_HIT_OTHER_TILE_NEAR	Other tile L2 hit E Near.
			20	Reserved	Reserved.
			21	MCDRAM_NEAR	MCDRAM Local.
			22	MCDRAM_FAR_OR_L2_HIT_OTHER_TILE_FAR	MCDRAM Far or Other tile L2 hit far.
			23	DRAM_NEAR	DRAM Local.
	24	DRAM_FAR	DRAM Far.		
	Data Supply from within same tile	25	L2_HITM_THIS_TILE	M-state.	
		26	L2_HITE_THIS_TILE	E-state.	
		27	L2_HITS_THIS_TILE	S-state.	
		28	L2_HITF_THIS_TILE	F-state.	
		29	Reserved	Reserved.	
		30	Reserved	Reserved.	
	Snoop Info; Only Valid in case of Data Supply from Untile	31	SNOOP_NONE	None of the cores were snooped.	
		32	NO_SNOOP_NEEDED	No snoop was needed to satisfy the request.	
		33	Reserved	Reserved.	
		34	Reserved	Reserved.	
		35	HIT_OTHER_TILE_FWD	Snoop request hit in the other tile with data forwarded.	
		36	HITM_OTHER_TILE	A snoop was needed and it HitM-ed in other core's L1 cache. HitM denotes a cache-line was in modified state before effect as a result of snoop.	
		37	NON_DRAM	Target was non-DRAM system address. This includes MMIO transactions.	
Outstanding requests	Weighted cycles	38	OUTSTANDING (Valid only for MSR_OFFCORE_RESP0. Should only be used on PMCO. This bit is reserved for MSR_OFFCORE_RESP1).	If set, counts total number of weighted cycles of any outstanding offcore requests with data response. Valid only for OFFCORE_RESP_0 event. Should only be used on PMCO. This bit is reserved for OFFCORE_RESP_1 event.	

20.4.1.1.3 Average Offcore Request Latency Measurement

Measurement of average latency of offcore transaction requests can be enabled using MSR_OFFCORE_RSP0.[bit 38] with the choice of request type specified in MSR_OFFCORE_RSP0.[bit 15:0].

Refer to Section 20.5.2.3, "Average Offcore Request Latency Measurement," for typical usage. Note that MSR_OFFCORE_RESPx registers are not shared between cores in Knights Landing. This allows one core to measure average latency while other core is measuring different offcore response events.

20.5 PERFORMANCE MONITORING (INTEL ATOM® PROCESSORS)

20.5.1 Performance Monitoring (45 nm and 32 nm Intel Atom® Processors)

45 nm and 32 nm Intel Atom processors report architectural performance monitoring versionID = 3 (supporting the aggregate capabilities of versionID 1, 2, and 3; see Section 20.2.3) and a host of non-architectural monitoring capabilities. These 45 nm and 32 nm Intel Atom processors provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters. CPUID.0AH:EAX[15:8] reports the MSRs available to software; see Section 20.2.1.

Non-architectural performance monitoring in Intel Atom processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

Architectural and non-architectural performance monitoring events in 45 nm and 32 nm Intel Atom processors support thread qualification using bit 21 (AnyThread) of IA32_PERFEVTSELx MSR, i.e., if IA32_PERFEVTSELx.AnyThread = 1, event counts include monitored conditions due to either logical processors in the same processor core.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 20-6 and described in Section 20.2.1.1 and Section 20.2.3.

Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields that provide the same qualifying actions like those listed in Table 20-77, Table 20-78, Table 20-79, and Table 20-80. One or more of these sub-fields may apply to specific events on an event-by-event basis. Precise Event Based Monitoring is supported using IA32_PMC0 (see also Section 18.4.9, "BTS and DS Save Area").

20.5.2 Performance Monitoring for Silvermont Microarchitecture

Intel processors based on the Silvermont microarchitecture report architectural performance monitoring versionID = 3 (see Section 20.2.3) and a host of non-architectural monitoring capabilities. Intel processors based on the Silvermont microarchitecture provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2). Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture.

Non-architectural performance monitoring in the Silvermont microarchitecture uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 20-6 and described in Section 20.2.1.1 and Section 20.2.3. Architectural and non-architectural performance monitoring events in the Silvermont microarchitecture ignore the AnyThread qualification regardless of its setting in IA32_PERFEVTSELx MSR.

20.5.2.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- The width of counter reported by CPUID.0AH:EAX[23:16] is 40 bits.

- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests.

20.5.2.1.1 Processor Event Based Sampling (PEBS)

In the Silvermont microarchitecture, the PEBS facility can be used with precise events. PEBS is supported using IA32_PMC0 (see also Section 18.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 20.6.2.4).

The list of precise events supported in the Silvermont microarchitecture is shown in Table 20-60.

Table 20-60. PEBS Performance Events for the Silvermont Microarchitecture

Event Name	Event Select	Sub-event	UMask
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
		RETURN	F7H
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H
		L2_MISS_LOADS	04H
		DLTB_MISS_LOADS	08H
		HITM	20H
REHABQ	03H	LD_BLOCK_ST_FORWARD	01H
		LD_SPLITS	08H

PEBS Record Format The PEBS record format supported by processors based on the Intel Silvermont microarchitecture is shown in Table 20-61, and each field in the PEBS record is 64 bits long.

Table 20-61. PEBS Record Format for the Silvermont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	Reserved
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP
58H	R9	B8H	Reserved

20.5.2.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 20-62 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

In the Silvermont microarchitecture, each MSR_OFFCORE_RSPx is shared by two processor cores.

Table 20-62. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMC0-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are shown in Figure 20-40 and Figure 20-41. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 20.5.2.3 for details.

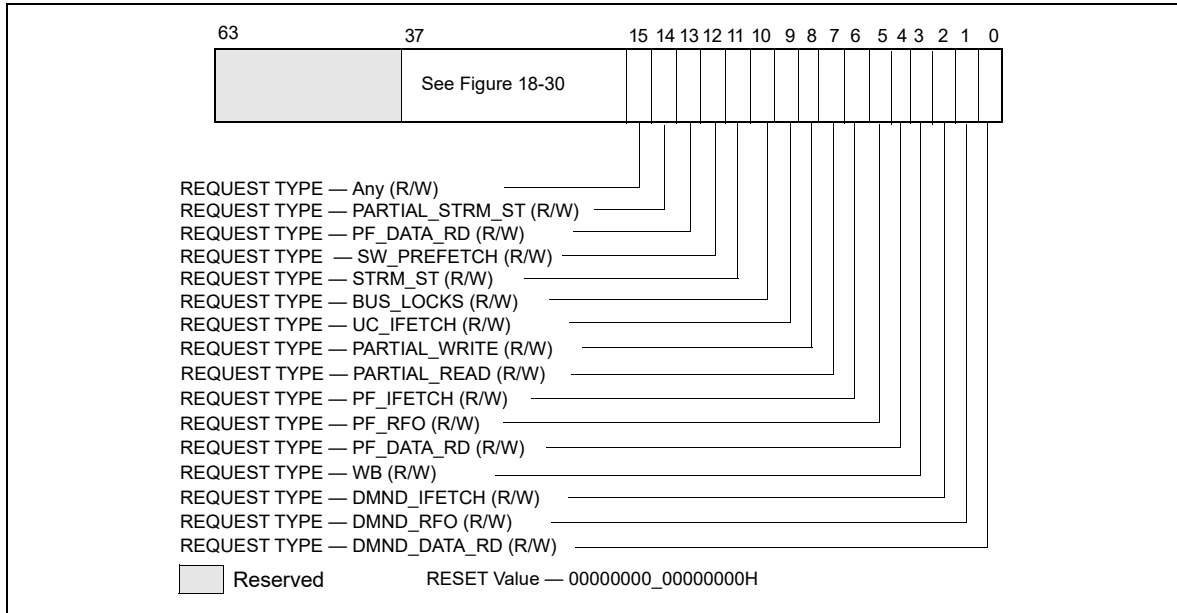


Figure 20-40. Request_Type Fields for MSR_OFFCORE_RSPx

Table 20-63. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PARTIAL_READ	7	Counts the number of demand reads of partial cache lines (including UC and WC).
PARTIAL_WRITE	8	Counts the number of demand RFO requests to write to partial cache lines (includes UC, WT, and WP).
UC_IFETCH	9	Counts the number of UC instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests.
STRM_ST	11	Streaming store requests.
SW_PREFETCH	12	Counts software prefetch requests.
PF_DATA_RD	13	Counts DCU hardware prefetcher data read requests.
PARTIAL_STRM_ST	14	Streaming store requests.
ANY	15	Any request that crosses IDI, including I/O.

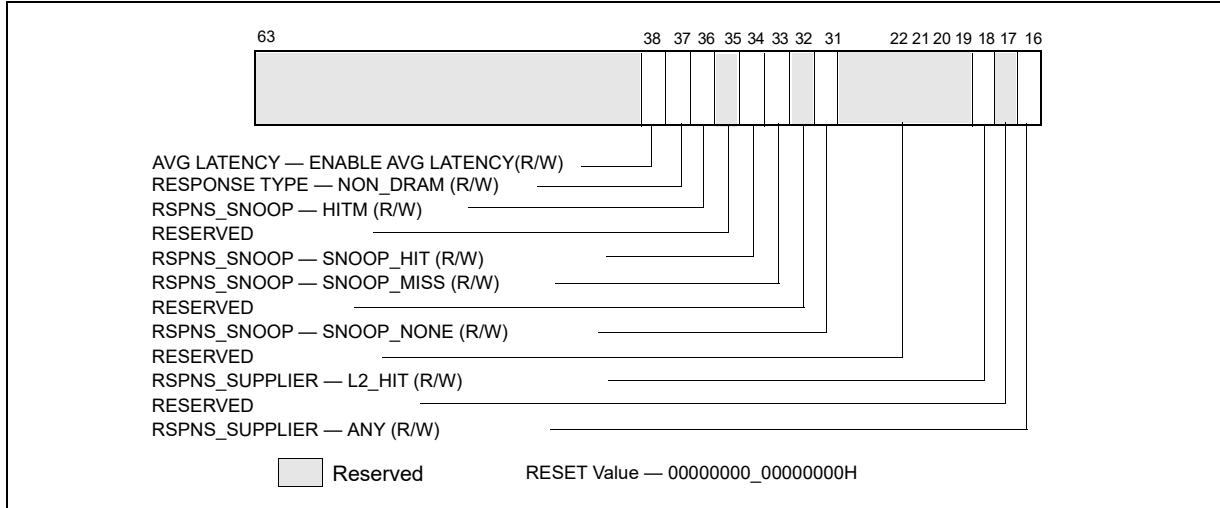


Figure 20-41. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSPx

To properly program this extra register, software must set at least one request type bit (Table 20-63) and a valid response type pattern (Table 20-64, Table 20-65). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 20-64. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	ANY_RESPONSE	16	Catch all value for any response types.
Supplier Info	Reserved	17	Reserved
	L2_HIT	18	Cache reference hit L2 in either M/E/S states.
	Reserved	30:19	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 20-65. MSR_OFFCORE_RSPx Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	Reserved	32	Reserved
	SNOOP_MISS	33	Counts the number of snoop misses when L2 misses.
	SNOOP_HIT	34	Counts the number of snoops hit in the other module where no modified copies were found.
	Reserved	35	Reserved

Table 20-65. MSR_OFFCORE_RSPx Snoop Info Field Definition (Contd.)

Subtype	Bit Name	Offset	Description
	HITM	36	Counts the number of snoops hit in the other module where modified copies were found in other core's L1 cache.
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.
	AVG_LATENCY	38	Enable average latency measurement by counting weighted cycles of outstanding offcore requests of the request type specified in bits 15:0 and any response (bits 37:16 cleared to 0). This bit is available in MSR_OFFCORE_RESP0. The weighted cycles is accumulated in the specified programmable counter IA32_PMCx and the occurrence of specified requests are counted in the other programmable counter.

20.5.2.3 Average Offcore Request Latency Measurement

Average latency for offcore transactions can be determined by using both MSR_OFFCORE_RSP registers. Using two performance monitoring counters, program the two OFFCORE_RESPONSE event encodings into the corresponding IA32_PERFEVTSELx MSRs. Count the weighted cycles via MSR_OFFCORE_RSP0 by programming a request type in MSR_OFFCORE_RSP0.[15:0] and setting MSR_OFFCORE_RSP0.OUTSTANDING[38] to 1, while setting the remaining bits to 0. Count the number of requests via MSR_OFFCORE_RSP1 by programming the same request type from MSR_OFFCORE_RSP0 into MSR_OFFCORE_RSP1[bit 15:0], and setting MSR_OFFCORE_RSP1.ANY_RESPONSE[16] = 1, while setting the remaining bits to 0. The average latency can be obtained by dividing the value of the IA32_PMCx register that counted weight cycles by the register that counted requests.

20.5.3 Performance Monitoring for Goldmont Microarchitecture

Intel Atom processors based on the Goldmont microarchitecture report architectural performance monitoring versionID = 4 (see Section 20.2.4) and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 20.2.4.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 20-6 and described in Section 20.2.1.1 and Section 20.2.3. The Goldmont microarchitecture does not support Hyper-Threading and thus architectural and non-architectural performance monitoring events ignore the AnyThread qualification regardless of its setting in the IA32_PERFEVTSELx MSR. However, Goldmont does not set the AnyThread deprecation bit (CPUID.0AH:EDX[15]).

The core PMU's capability is similar to that of the Silvermont microarchitecture described in Section 20.5.2, with some differences and enhancements summarized in Table 20-66.

Table 20-66. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 20.2.1.
# of general-purpose counters per core	4	2	Use CPUID to determine # of counters. See Section 20.2.1.
Counter width (R,W)	R:48, W: 32/48	R:40, W:32	See Section 20.2.2.
Architectural Performance Monitoring version ID	4	3	Use CPUID to determine # of counters. See Section 20.2.1.

Table 20-66. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
PMI Overhead Mitigation	<ul style="list-style-type: none"> Freeze_Perfmon_on_PMI with streamlined semantics. Freeze_LBR_on_PMI with streamlined semantics for branch profiling. 	<ul style="list-style-type: none"> Freeze_Perfmon_on_PMI with legacy semantics. Freeze_LBR_on_PMI with legacy semantics for branch profiling. 	See Section 18.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	<ul style="list-style-type: none"> Query via IA32_PERF_GLOBAL_STATUS Reset via IA32_PERF_GLOBAL_STATUS_RESET Set via IA32_PERF_GLOBAL_STATUS_SET 	<ul style="list-style-type: none"> Query via IA32_PERF_GLOBAL_STATUS Reset via IA32_PERF_GLOBAL_OVF_CTRL 	See Section 20.2.4.
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	<ul style="list-style-type: none"> Individual counter overflow PEBS buffer overflow ToPA buffer overflow CTR_Frz, LBR_Frz 	<ul style="list-style-type: none"> Individual counter overflow PEBS buffer overflow 	See Section 20.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	<ul style="list-style-type: none"> CTR_Frz, LBR_Frz 	No	See Section 20.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	No	See Section 20.2.4.3.
Processor Event Based Sampling (PEBS) Events	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 20-67.	See Section 20.5.2.1.1. General-Purpose Counter 0 only. Only supports precise events (see Table 20-60).	IA32_PMC0 only.
PEBS record format encoding	0011b	0010b	
Reduce skid PEBS	IA32_PMC0 only	No	
Data Address Profiling	Yes	No	
PEBS record layout	Table 20-68; enhanced fields at offsets 90H- 98H; and TSC record field at C0H.	Table 20-61.	
PEBS EventingIP	Yes	Yes	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register.	MSR 1A6H and 1A7H, shared by a pair of cores.	Nehalem supports 1A6H only.

20.5.3.1 Processor Event Based Sampling (PEBS)

Processor event based sampling (PEBS) on the Goldmont microarchitecture is enhanced over prior generations with respect to sampling support of precise events and non-precise events. In the Goldmont microarchitecture, PEBS is supported using IA32_PMC0 for all events (see Section 18.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor at the time the sample was generated.

Precise events work the same way on Goldmont microarchitecture as on the Silvermont microarchitecture. The record will be generated after an instruction that causes the event when the counter is already overflowed and will capture the architectural state at this point (see Section 20.6.2.4 and Section 18.4.9). The eventingIP in the record will indicate the instruction that caused the event. The list of precise events supported in the Goldmont microarchitecture is shown in Table 20-67.

In the Goldmont microarchitecture, the PEBS facility also supports the use of non-precise events to record processor state information into PEBS records with the same format as with precise events.

However, a non-precise event may not be attributable to a particular retired instruction or the time of instruction execution. When the counter overflows, a PEBS record will be generated at the next opportunity. Consider the event ICACHE.HIT. When the counter overflows, the processor is fetching future instructions. The PEBS record will be generated at the next opportunity and capture the state at the processor's current retirement point. It is likely that the instruction fetch that caused the event to increment was beyond that current retirement point. Other examples of non-precise events are CPU_CLK_UNHALTED.CORE_P and HARDWARE_INTERRUPTS.RECEIVED. CPU_CLK_UNHALTED.CORE_P will increment each cycle that the processor is awake. When the counter over-flows, there may be many instructions in various stages of execution. Additionally, zero, one or multiple instructions may be retired the cycle that the counter overflows. HARDWARE_INTERRUPTS.RECEIVED increments independent of any instructions being executed. For all non-precise events, the PEBS record will be generated at the next opportunity, after the counter has overflowed. The PEBS facility thus allows for identification of the instructions which were executing when the event overflowed.

After generating a record for a non-precise event, the PEBS facility reloads the counter and resumes execution, just as is done for precise events. Unlike interrupt-based sampling, which requires an interrupt service routine to collect the sample and reload the counter, the PEBS facility can collect samples even when interrupts are masked and without using NMI. Since a PEBS record is generated immediately when a counter for a non-precise event is enabled, it may also be generated after an overflow is set by an MSR write to IA32_PERF_GLOBAL_STATUS_SET.

Table 20-67. Precise Events Supported by the Goldmont Microarchitecture

Event Name	Event Select	Sub-event	UMask
LD_BLOCKS	03H	DATA_UNKNOWN	01H
		STORE_FORWARD	02H
		4K_ALIAS	04H
		UTLB_MISS	08H
		ALL_BLOCK	10H
MISALIGN_MEM_REF	13H	LOAD_PAGE_SPLIT	02H
		STORE_PAGE_SPLIT	04H
INST_RETIRED	C0H	ANY	00H
UOPS_RETITRED	C2H	ANY	00H
		LD_SPLITSMS	01H
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
RETURN	F7H		
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H

Table 20-67. Precise Events Supported by the Goldmont Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_UOPS_RETIRED	DOH	ALL_LOADS	81H
		ALL_STORES	82H
		ALL	83H
		DLTB_MISS_LOADS	11H
		DLTB_MISS_STORES	12H
		DLTB_MISS	13H
MEM_LOAD_UOPS_RETIRED	D1H	L1_HIT	01H
		L2_HIT	02H
		L1_MISS	08H
		L2_MISS	10H
		HITM	20H
		WCB_HIT	40H
		DRAM_HIT	80H

The PEBS record format supported by processors based on the Goldmont microarchitecture is shown in Table 20-68, and each field in the PEBS record is 64 bits long.

Table 20-68. PEBS Record Format for the Goldmont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counters
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Reserved
40H	R/EBP	A8H	Reserved
48H	R/ESP	BOH	EventingRIP
50H	R8	B8H	Reserved
58H	R9	COH	TSC
60H	R10		

On Goldmont microarchitecture, all 64 bits of architectural registers are written into the PEBS record regardless of processor mode.

With PEBS record format encoding 0011b, offset 90H reports the “Applicable Counter” field, which indicates which counters actually requested generating a PEBS record. This allows software to correlate the PEBS record entry properly with the instruction that caused the event even when multiple counters are configured to record PEBS records and multiple bits are set in the field. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

20.5.3.1.1 PEBS Data Linear Address Profiling

Goldmont supports the Data Linear Address field introduced in Haswell. It does not support the Data Source Encoding or Latency Value fields that are also part of Data Address Profiling; those fields are present in the record but are reserved.

For Goldmont microarchitecture, the Data Linear Address field will record the linear address of memory accesses in the previous instruction (e.g., the one that triggered a precise event that caused the PEBS record to be generated). Goldmont microarchitecture may record a Data Linear Address for the instruction that caused the event even for events not related to memory accesses. This may differ from other microarchitectures.

20.5.3.1.2 Reduced Skid PEBS

Processors based on Goldmont Plus microarchitecture support the Reduced Skid PEBS feature described in Section 20.9.4 on the IA32_PMC0 counter. Although Extended PEBS adds support for generating PEBS records for precise events on additional general-purpose and fixed-function performance counters, those counters do not support the Reduced Skid PEBS feature.

20.5.3.1.3 Enhancements to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62]

In addition to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62] being set when PEBS_Index reaches the PEBS_Interrupt_Theshold, the bit is also set when PEBS_Index is out of bounds. That is, the bit will be set when PEBS_Index < PEBS_Buffer_Base or PEBS_Index > PEBS_Absolute_Maximum. Note that when an out of bound condition is encountered, the overflow bits in IA32_PERF_GLOBAL_STATUS will be cleared according to Applicable Counters, however the IA32_PMCx values will not be reloaded with the Reset values stored in the DS_AREA.

20.5.3.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 20-62 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

The Goldmont microarchitecture provides unique pairs of MSR_OFFCORE_RSPx registers per core.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 specifies the request type of a transaction request to the uncore. This is described in Table 20-69.
- Bits 30:16 specifies common supplier information or an L2 Hit, and is described in Table 20-64.
- If L2 misses, then Bits 37:31 can be used to specify snoop response information and is described in Table 20-70.
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 20.5.2.3 for details.

Table 20-69. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts cacheline read requests due to demand reads (excludes prefetches).
DEMAND_RFO	1	Counts cacheline read for ownership (RFO) requests due to demand writes (excludes prefetches).
DEMAND_CODE_RD	2	Counts demand instruction cacheline and I-side prefetch requests that miss the instruction cache.
COREWB	3	Counts writeback transactions caused by L1 or L2 cache evictions.
PF_L2_DATA_RD	4	Counts data cacheline reads generated by hardware L2 cache prefetcher.
PF_L2_RFO	5	Counts reads for ownership (RFO) requests generated by L2 prefetcher.
Reserved	6	Reserved.

Table 20-69. MSR_OFFCORE_RSPx Request_Type Field Definition (Contd.)

Bit Name	Offset	Description
PARTIAL_READS	7	Counts demand data partial reads, including data in uncacheable (UC) or uncacheable (WC) write combining memory types.
PARTIAL_WRITES	8	Counts partial writes, including uncacheable (UC), write through (WT) and write protected (WP) memory type writes.
UC_CODE_READS	9	Counts code reads in uncacheable (UC) memory region.
BUS_LOCKS	10	Counts bus lock and split lock requests.
FULL_STREAMING_STORES	11	Counts full cacheline writes due to streaming stores.
SW_PREFETCH	12	Counts cacheline requests due to software prefetch instructions.
PF_L1_DATA_RD	13	Counts data cacheline reads generated by hardware L1 data cache prefetcher.
PARTIAL_STREAMING_STORES	14	Counts partial cacheline writes due to streaming stores.
ANY_REQUEST	15	Counts requests to the uncore subsystem.

To properly program this extra register, software must set at least one request type bit (Table 20-63) and a valid response type pattern (either Table 20-64 or Table 20-70). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 20-70. MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests

Subtype	Bit Name	Offset	Description
L2_MISS (Snoop Info)	Reserved	32:31	Reserved
	L2_MISS.SNOOP_MISS_OR_NO_SNOOP_NEEDED	33	A true miss to this module, for which a snoop request missed the other module or no snoop was performed/needed.
	L2_MISS.HIT_OTHER_CORE_NO_FWD	34	A snoop hit in the other processor module, but no data forwarding is required.
	Reserved	35	Reserved
	L2_MISS.HITM_OTHER_CORE	36	Counts the number of snoops hit in the other module or other core's L1 where modified copies were found.
	L2_MISS.NON_DRAM	37	Target was a non-DRAM system address. This includes MMIO transactions.
Outstanding requests ¹	OUTSTANDING	38	Counts weighted cycles of outstanding offcore requests of the request type specified in bits 15:0, from the time the XQ receives the request and any response is received. Bits 37:16 must be set to 0. This bit is only available in MSR_OFFCORE_RESP0.

NOTES:

1. See Section 20.5.2.3, "Average Offcore Request Latency Measurement" for details on how to use this bit to extract average latency.

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

Any_Response Bit | L2 Hit | 'OR' of Snoop Info Bits | Outstanding Bit

20.5.3.3 Average Offcore Request Latency Measurement

In Goldmont microarchitecture, measurement of average latency of offcore transaction requests is the same as described in Section 20.5.2.3.

20.5.4 Performance Monitoring for Goldmont Plus Microarchitecture

Intel Atom processors based on the Goldmont Plus microarchitecture report architectural performance monitoring versionID = 4 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 20.2.4.

Goldmont Plus performance monitoring capabilities are similar to Goldmont capabilities. The differences are in specific events and in which counters support PEBS. Goldmont Plus introduces the ability for fixed performance monitoring counters to generate PEBS records.

Goldmont Plus will set the AnyThread deprecation CPUID bit (CPUID.0AH:EDX[15]) to indicate that the Any-Thread bits in IA32_PERFVTSELx and IA32_FIXED_CTR_CTRL have no effect.

The core PMU's capability is similar to that of the Goldmont microarchitecture described in Section 20.6.3, with some differences and enhancements summarized in Table 20-71.

Table 20-71. Core PMU Comparison Between the Goldmont Plus and Goldmont Microarchitectures

Box	Goldmont Plus Microarchitecture	Goldmont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 20.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 20.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change.
Architectural Performance Monitoring version ID	4	4	No change.
Processor Event Based Sampling (PEBS) Events	All General-Purpose and Fixed counters. Each General-Purpose counter supports all events (precise and non-precise).	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 20-67.	Goldmont Plus supports PEBS on all counters.
PEBS record format encoding	0011b	0011b	No change.

20.5.4.1 Extended PEBS

The PEBS facility in Goldmont Plus microarchitecture provides a number of enhancements relative to PEBS in processors from previous generations. Enhancement of PEBS facility with the Extended PEBS feature are described in detail in section 18.9.

20.5.5 Performance Monitoring for Tremont Microarchitecture

Intel Atom processors based on the Tremont microarchitecture report architectural performance monitoring versionID = 5 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 5 capabilities are described in Section 20.2.5.

Tremont performance monitoring capabilities are similar to Goldmont Plus capabilities, with the following extensions:

- Support for Adaptive PEBS.
- Support for PEBS output to Intel® Processor Trace.
- Precise Distribution support on Fixed Counter0.
- Compatibility enhancements to off-core response MSRs, MSR_OFFCORE_RSPx.

The differences and enhancements between Tremont microarchitecture and Goldmont Plus microarchitecture are summarized in Table 20-72.

Table 20-72. Core PMU Comparison Between the Tremont and Goldmont Plus Microarchitectures

Box	Tremont Microarchitecture	Goldmont Plus Microarchitecture	Comment
# of fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 20.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 20.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change. See Section 20.2.2.
Architectural Performance Monitoring version ID	5	4	
PEBS record format encoding	0100b	0011b	See Section 20.6.2.4.2.
Reduce skid PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 only	
Extended PEBS	Yes	Yes	See Section 20.5.4.1.
Adaptive PEBS	Yes	No	See Section 20.9.2.
PEBS output	DS Save Area or Intel® Processor Trace	DS Save Area only	See Section 20.5.5.2.1.
PEBS record layout	See Section 20.9.2.3 for output to DS, Section 20.5.5.2.2 for output to Intel PT.	Table 20-68; enhanced fields at offsets 90H- 98H; and TSC record field at C0H.	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register.	

20.5.5.1 Adaptive PEBS

The PEBS record format and configuration interface has changed versus Goldmont Plus, as the Tremont microarchitecture includes support for the configurable Adaptive PEBS records; see Section 20.9.2.

20.5.5.2 PEBS output to Intel® Processor Trace

Intel Atom processors based on the Tremont microarchitecture introduce the following Precise Event-Based Sampling (PEBS) extensions:

- A mechanism to direct PEBS output into the Intel® Processor Trace (Intel® PT) output stream. In this scenario, the PEBS record is written in packetized form, in order to co-exist with other Intel PT trace data.
- New Performance Monitoring counter reload MSRs, which are used by PEBS in place of the counter reload values stored in the DS Management area when PEBS output is directed into the Intel PT output stream.

Processors that indicate support for Intel PT by setting CPUID.07H.0.EBX[25]=1, and set the new IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] bit, support these extensions.

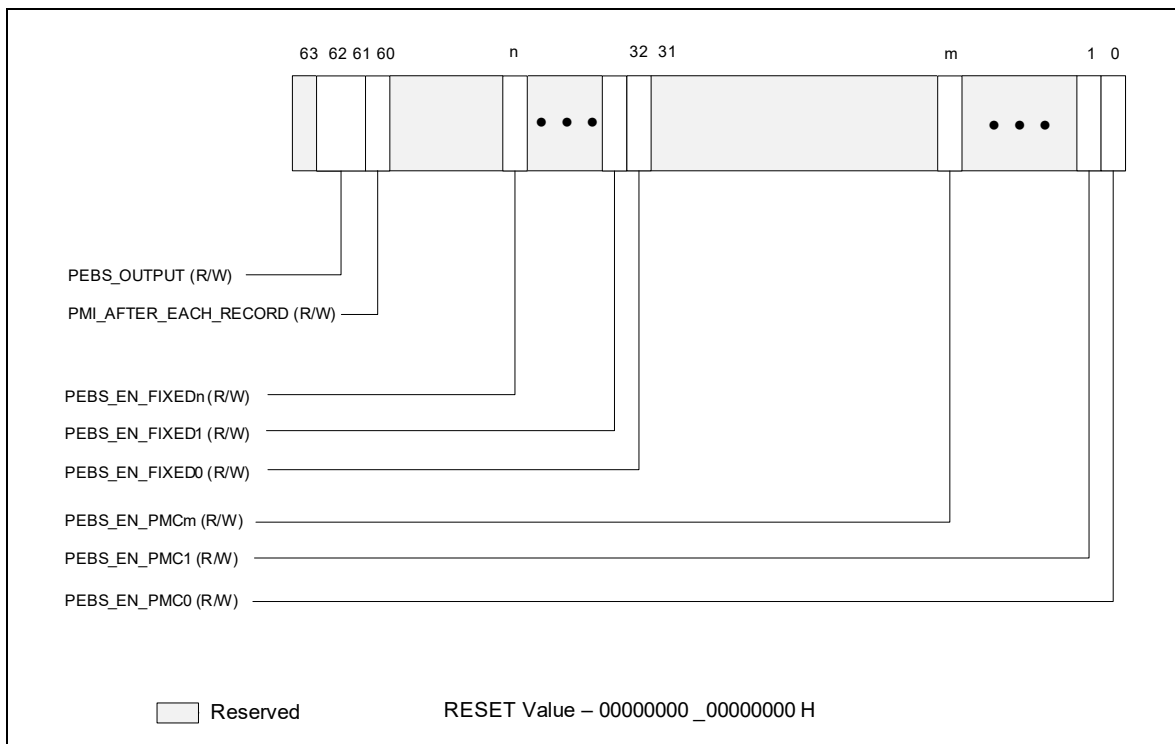
20.5.5.2.1 PEBS Configuration

PEBS output to Intel Processor Trace includes support for two new fields in IA32_PEBS_ENABLE.

Table 20-73. New Fields in IA32_PEBS_ENABLE

Field	Description
PMI_AFTER_EACH_RECORD[60]	Pend a PerfMon Interrupt (PMI) after each PEBS event.
PEBS_OUTPUT[62:61]	Specifies PEBS output destination. Encodings: 00B: DS Save Area. Matches legacy PEBS behavior, output location defined by IA32_DS_AREA. 01B: Intel PT trace output. 10B: Reserved. 11B: Reserved.

When PEBS_OUTPUT is set to 01B, the DS Management Area is not used and need not be configured. Instead, the output mechanism is configured through IA32_RTIT_CTL and other Intel PT MSRs, while counter reload values are configured in the MSR_RELOAD_PMCx MSRs. Details on configuring Intel PT can be found in Section 33.2.7.

**Figure 20-42. IA32_PEBS_ENABLE MSR with PEBS Output to Intel® Processor Trace**

20.5.5.2.2 PEBS Record Format in Intel® Processor Trace

The format of the PEBS record changes when output to Intel PT, as the PEBS state is packetized. Each PEBS grouping is emitted as a Block Begin (BBP) and following Block Item (BIP) packets. A PEBS grouping ends when either a new PEBS grouping begins (indicated by a BBP packet) or a Block End (BEP) packet is encountered. See Section 33.4.1.1 for details of these Intel PT packets.

Because the packet headers describe the state held in the packet payload, PEBS state ordering is not fixed. PEBS state groupings may be emitted in any order, and the PEBS state elements within those groupings may be emitted in any order. Further, there is no packet that provides indication of “Record Format” or “Record Size”.

If Intel PT tracing is not enabled (IA32_RTIT_STATUS.TriggerEn=0), any PEBS records triggered will be dropped. PEBS packets do not depend on ContextEn or FilterEn in IA32_RTIT_STATUS, any filtering of PEBS must be enabled from within the PerfMon configuration. Counter reload will occur in all scenarios where PEBS is triggered, regardless of TriggerEn.

The PEBS threshold mechanism for generating PerfMon Interrupts (PMIs) is not available in this mode. However, there exist other means to generate PMIs based on PEBS output. When the Intel PT ToPA output mechanism is chosen, a PMI can optionally be pended when a ToPA region is filled; see Section 33.2.7.2 for details. Further, software can opt to generate a PMI on each PEBS record by setting the new IA32_PEBS_ENABLE.PMI_AFTER_EACH_RECORD[60] bit.

The IA32_PERF_GLOBAL_STATUS.OvfDSBuffer bit will not be set in this mode.

20.5.5.2.3 PEBS Counter Reload

When PEBS output is directed into Intel PT (IA32_PEBS_ENABLE.PEBS_OUTPUT = 01B), new MSR_RELOAD_PMCx MSRs are used by the PEBS routine to reload PerfMon counters. The value from the associated reload MSR will be loaded to the appropriate counter on each PEBS event.

20.5.5.3 Precise Distribution Support on Fixed Counter 0

The Tremont microarchitecture supports the PDIR (Precise Distribution of Retired Instructions) facility, as described in Section 20.3.4.4.4, on Fixed Counter 0. Fixed Counter 0 counts the INST_RETIRED.ALL event. PEBS skid for Fixed Counter 0 will be precisely one instruction.

This is in addition to the reduced skid PEBS behavior on IA32_PMC0; see Section 20.5.3.1.2.

20.5.5.4 Compatibility Enhancements to Offcore Response MSRs

The Off-core Response facility is similar to that described in Section 20.5.3.2.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as shown below. RequestType bits are defined in Table 20-74, ResponseType bits in Table 20-75, and SnoopInfo bits in Table 20-76.

Table 20-74. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPf_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPf_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPf_L2_CODE_RD	6	Counts all prefetch (that bring data to L2 only) code reads.
Reserved	9:7	Reserved.
HWPf_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
Reserved	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WiL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (wCiL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (wCiLF).

Table 20-74. MSR_OFFCORE_RSPx Request Type Definition (Contd.)

Bit Name	Offset	Description
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

Table 20-75. MSR_OFFCORE_RSPx Response Type Definition

Bit Name	Offset	Description
ANY_RESPONSE	16	Catch all value for any response types.
L3_HIT_M	18	LLC/L3 Hit - M-state.
L3_HIT_E	19	LLC/L3 Hit - E-state.
L3_HIT_S	20	LLC/L3 Hit - S-state.
L3_HIT_F	21	LLC/L3 Hit - I-state.
LOCAL_DRAM	26	LLC/L3 Miss, DRAM Hit.
OUTSTANDING	63	Average latency of outstanding requests with the other counter counting number of occurrences; can also can be used to count occupancy.

Table 20-76. MSR_OFFCORE_RSPx Snoop Info Definition

Bit Name	Offset	Description
SNOOP_NONE	31	None of the cores were snooped. <ul style="list-style-type: none"> LLC miss and Dram data returned directly to the core.
SNOOP_NOT_NEEDED	32	No snoop needed to satisfy the request. <ul style="list-style-type: none"> LLC hit and CV bit(s) (core valid) was not set. LLC miss and Dram data returned directly to the core.
SNOOP_MISS	33	A snoop was sent but missed. <ul style="list-style-type: none"> LLC hit and CV bit(s) was set but snoop missed (silent data drop in core), data returned from LLC. LLC miss and Dram data returned directly to the core.
SNOOP_HIT_NO_FWD	34	A snoop was sent but no data forward. <ul style="list-style-type: none"> LLC hit and CV bit(s) was set but no data forward from the core, data returned from LLC. LLC miss and Dram data returned directly to the core.
SNOOP_HIT_WITH_FWD	35	A snoop was sent and non-modified data was forward. <ul style="list-style-type: none"> LLC hit and CV bit(s) was set, non-modified data was forward from core.
SNOOP_HITM	36	A snoop was sent and modified data was forward. <ul style="list-style-type: none"> LLC hit E or M and the CV bit(s) was set, modified data was forward from core.
NON_DRAM_BIT	37	Target was non-DRAM system address, MMIO access. <ul style="list-style-type: none"> LLC miss and Non-Dram data returned.

The Off-core Response capability behaves as follows:

- To specify a complete offcore response filter, software must properly program at least one RequestType and one ResponseType. A valid request type must have at least one bit set in the non-reserved bits of 15:0 or 49:44. A valid response type must be a non-zero value of one the following expressions:
 - Read requests:
Any_Response Bit | ('OR' of Supplier Info Bits) 'AND' ('OR' of Snoop Info Bits) | Outstanding Bit
 - Write requests:
Any_Response Bit | ('OR' of Supplier Info Bits) | Outstanding Bit
- When the ANY_RESPONSE bit in the ResponseType is set, all other response type bits will be ignored.
- True Demand Cacheable Loads include neither L1 Prefetches nor Software Prefetches.
- Bits 15:0 and Bits 49:44 specifies the request type of a transaction request to the uncore. This is described in Table 20-74.
- Bits 30:16 specifies common supplier information.
- "Outstanding Requests" (bit 63) is only available on MSR_OFFCORE_RSP0; a #GP fault will occur if software attempts to write a 1 to this bit in MSR_OFFCORE_RSP1. It is mutually exclusive with any ResponseType. Software must guarantee that all other ResponseType bits are set to 0 when the "Outstanding Requests" bit is set.
- "Outstanding Requests" bit 63 can enable measurement of the average latency of a specific type of off-core transaction; two programmable counters must be used simultaneously and the RequestType programming for MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 must be the same when using this Average Latency feature. See Section 20.5.2.3 for further details.

20.6 PERFORMANCE MONITORING (LEGACY INTEL PROCESSORS)

20.6.1 Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)

In Intel Core Solo and Intel Core Duo processors, non-architectural performance monitoring events are programmed using the same facilities (see Figure 20-1) used for architectural performance events.

Non-architectural performance events use event select values that are model-specific. Event mask (Umask) values are also specific to event logic units. Some microarchitectural conditions detectable by a Umask value may have specificity related to processor topology (see Section 9.6, "Detecting Hardware Multi-Threading Support and Topology," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A). As a result, the unit mask field (for example, IA32_PERFEVTSELx[bits 15:8]) may contain sub-fields that specify topology information of processor cores.

The sub-field layout within the Umask field may support two-bit encoding that qualifies the relationship between a microarchitectural condition and the originating core. This data is shown in Table 20-77. The two-bit encoding for core-specificity is only supported for a subset of Umask values (see: <https://perfmon-events.intel.com/>) and for Intel Core Duo processors. Such events are referred to as core-specific events.

Table 20-77. Core Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 15:14 Encoding	Description
11B	All cores
10B	Reserved
01B	This core
00B	Reserved

Some microarchitectural conditions allow detection specificity only at the boundary of physical processors. Some bus events belong to this category, providing specificity between the originating physical processor (a bus agent) versus other agents on the bus. Sub-field encoding for agent specificity is shown in Table 20-78.

Table 20-78. Agent Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13 Encoding	Description
0	This agent
1	Include all agents

Some microarchitectural conditions are detectable only from the originating core. In such cases, unit mask does not support core-specificity or agent-specificity encodings. These are referred to as core-only conditions.

Some microarchitectural conditions allow detection specificity that includes or excludes the action of hardware prefetches. A two-bit encoding may be supported to qualify hardware prefetch actions. Typically, this applies only to some L2 or bus events. The sub-field encoding for hardware prefetch qualification is shown in Table 20-79.

Table 20-79. HW Prefetch Qualification Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13:12 Encoding	Description
11B	All inclusive
10B	Reserved
01B	Hardware prefetch only
00B	Exclude hardware prefetch

Some performance events may (a) support none of the three event-specific qualification encodings (b) may support core-specificity and agent specificity simultaneously (c) or may support core-specificity and hardware prefetch qualification simultaneously. Agent-specificity and hardware prefetch qualification are mutually exclusive.

In addition, some L2 events permit qualifications that distinguish cache coherent states. The sub-field definition for cache coherency state qualification is shown in Table 20-80. If no bits in the MESI qualification sub-field are set for an event that requires setting MESI qualification bits, the event count will not increment.

Table 20-80. MESI Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	Description
Bit 11	Counts modified state
Bit 10	Counts exclusive state
Bit 9	Counts shared state
Bit 8	Counts Invalid state

20.6.2 Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)

In addition to architectural performance monitoring, processors based on the Intel Core microarchitecture support non-architectural performance monitoring events.

Architectural performance events can be collected using general-purpose performance counters. Non-architectural performance events can be collected using general-purpose performance counters (coupled with two IA32_PERFEVTSELx MSRs for detailed event configurations), or fixed-function performance counters (see Section 20.6.2.1). IA32_PERFEVTSELx MSRs are architectural; their layout is shown in Figure 20-1. Starting with Intel Core 2

processor T 7700, fixed-function performance counters and associated counter control and status MSR becomes part of architectural performance monitoring version 2 facilities (see also Section 20.2.2).

Non-architectural performance events in processors based on Intel Core microarchitecture use event select values that are model-specific. Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields identical to those listed in Table 20-77, Table 20-78, Table 20-79, and Table 20-80. One or more of these sub-fields may apply to specific events on an event-by-event basis.

In addition, the UMASK filed may also contain a sub-field that allows detection specificity related to snoop responses. Bits of the snoop response qualification sub-field are defined in Table 20-81.

Table 20-81. Bus Snoop Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	Description
Bit 11	HITM response
Bit 10	Reserved
Bit 9	HIT response
Bit 8	CLEAN response

There are also non-architectural events that support qualification of different types of snoop operation. The corresponding bit field for snoop type qualification are listed in Table 20-82.

Table 20-82. Snoop Type Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 9:8	Description
Bit 9	CMP2I snoops
Bit 8	CMP2S snoops

No more than one sub-field of MESI, snoop response, and snoop type qualification sub-fields can be supported in a performance event.

NOTE

Software must write known values to the performance counters prior to enabling the counters. The content of general-purpose counters and fixed-function counters are undefined after INIT or RESET.

20.6.2.1 Fixed-function Performance Counters

Processors based on Intel Core microarchitecture provide three fixed-function performance counters. Bits beyond the width of the fixed counter are reserved and must be written as zeros. Model-specific fixed-function performance counters on processors that support Architectural Perfmon version 1 are 40 bits wide.

Each of the fixed-function counter is dedicated to count a pre-defined performance monitoring events. See Table 20-2 for details of the PMC addresses and what these events count.

Programming the fixed-function performance counters does not involve any of the IA32_PERFEVTSELx MSRs, and does not require specifying any event masks. Instead, the MSR IA32_FIXED_CTR_CTRL provides multiple sets of 4-bit fields; each 4-bit field controls the operation of a fixed-function performance counter (PMC). See Figures 20-43. Two sub-fields are defined for each control. See Figure 20-43; bit fields are:

- **Enable field (low 2 bits in each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring 0.

When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring greater than 0.

Writing 0 to both bits stops the performance counter. Writing 11B causes the counter to increment irrespective of privilege levels.

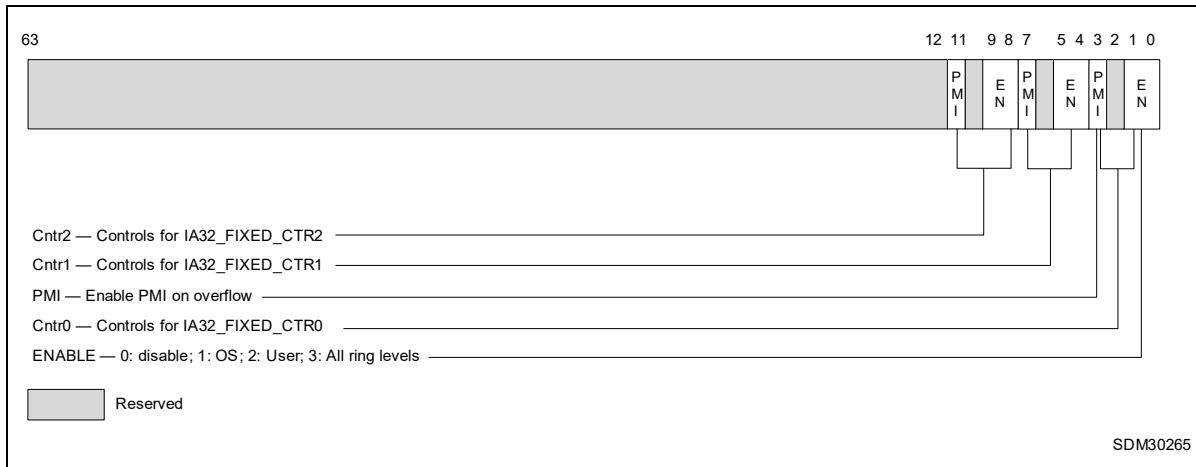


Figure 20-43. Layout of IA32_FIXED_CTR_CTRL MSR

- **PMI field (fourth bit in each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

20.6.2.2 Global Counter Control Facilities

Processors based on Intel Core microarchitecture provides simplified performance counter control that simplifies the most frequent operations in programming performance events, i.e., enabling/disabling event counting and checking the status of counter overflows. This is done by the following three MSRs:

- MSR_PERF_GLOBAL_CTRL enables/disables event counting for all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.
- MSR_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single RDMSR.
- MSR_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.

MSR_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting in each performance counter (see Figure 20-44). Each enable bit in MSR_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

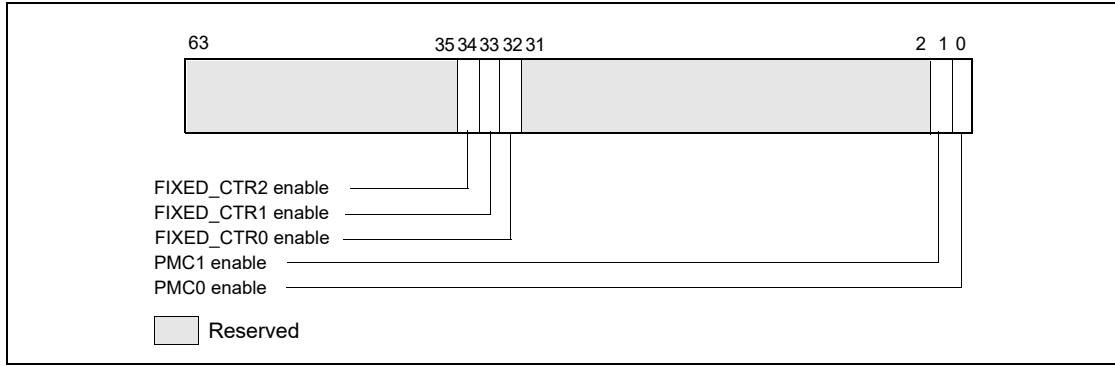


Figure 20-44. Layout of MSR_PERF_GLOBAL_CTRL MSR

MSR_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. MSR_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. MSR_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware (see Figure 20-45). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

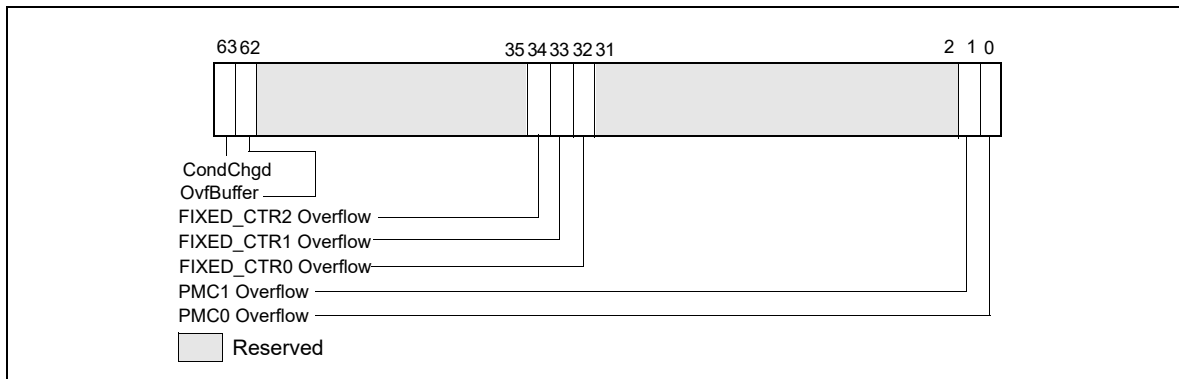


Figure 20-45. Layout of MSR_PERF_GLOBAL_STATUS MSR

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 18.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

MSR_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 20-46). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

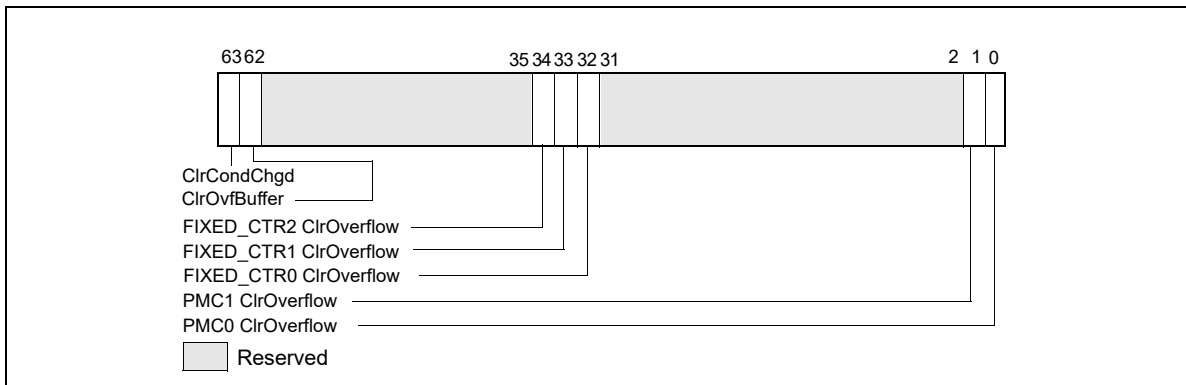


Figure 20-46. Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

20.6.2.3 At-Retirement Events

Many non-architectural performance events are impacted by the speculative nature of out-of-order execution. A subset of non-architectural performance events on processors based on Intel Core microarchitecture are enhanced with a tagging mechanism (similar to that found in Intel NetBurst[®] microarchitecture) that exclude contributions that arise from speculative execution. The at-retirement events available in processors based on Intel Core microarchitecture does not require special MSR programming control (see Section 20.6.3.6, “At-Retirement Counting”), but is limited to IA32_PMC0. See Table 20-83 for a list of events available to processors based on Intel Core microarchitecture.

Table 20-83. At-Retirement Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
ITLB_MISS_RETIRED	00H	C9H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

20.6.2.4 Processor Event Based Sampling (PEBS)

Processors based on Intel Core microarchitecture also support processor event based sampling (PEBS). This feature was introduced by processors based on Intel NetBurst microarchitecture.

PEBS uses a debug store mechanism and a performance monitoring interrupt to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 20.6.2.4.2 and Section 18.4.9).

In cases where the same instruction causes BTS and PEBS to be activated, PEBS is processed before BTS are processed. The PMI request is held until the processor completes processing of PEBS and BTS.

For processors based on Intel Core microarchitecture, precise events that can be used with PEBS are listed in Table 20-84. The procedure for detecting availability of PEBS is the same as described in Section 20.6.3.8.1.

Table 20-84. PEBS Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
INSTR_RETIRED.ANY_P	00H	C0H
X87_OPS_RETIRED.ANY	FEH	C1H
BR_INST_RETIRED.MISPRED	00H	C5H
SIMD_INST_RETIRED.ANY	1FH	C7H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

20.6.2.4.1 Setting up the PEBS Buffer

For processors based on Intel Core microarchitecture, PEBS is available using IA32_PMC0 only. Use the following procedure to set up the processor and IA32_PMC0 counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area. In processors based on Intel Core microarchitecture, PEBS records consist of 64-bit address entries. See Figure 18-8 to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS on PMC0 flag (bit 0) in IA32_PEBS_ENABLE MSR.
3. Set up the IA32_PMC0 performance counter and IA32_PERFEVTSEL0 for an event listed in Table 20-84.

20.6.2.4.2 PEBS Record Format

The PEBS record format may be extended across different processor implementations. The IA32_PERF_CAPABILITIES MSR defines a mechanism for software to handle the evolution of PEBS record format in processors that support architectural performance monitoring with version ID equals 2 or higher. The bit fields of IA32_PERF_CAPABILITIES are defined in Table 2-2 of Chapter 2, "Model-Specific Registers (MSRs)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4. The relevant bit fields that governs PEBS are:

- **PEBTrap [bit 6]:** When set, PEBS recording is trap-like. After the PEBS-enabled counter has overflowed, PEBS record is recorded for the next PEBS-able event at the completion of the sampled instruction causing the PEBS event. When clear, PEBS recording is fault-like. The PEBS record is recorded before the sampled instruction causing the PEBS event.
- **PEBSSaveArchRegs [bit 7]:** When set, PEBS will save architectural register and state information according to the encoded value of the PEBSRecordFormat field. When clear, only the return instruction pointer and flags are recorded. On processors based on Intel Core microarchitecture, this bit is always 1.
- **PEBSRecordFormat [bits 11:8]:** Valid encodings are:
 - 0000B: Only general-purpose registers, instruction pointer and RFLAGS registers are saved in each PEBS record (See Section 20.6.3.8).
 - 0001B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS and load latency data. (See Section 20.3.1.1.1).
 - 0010B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS, load latency data, and TSX tuning information. (See Section 20.3.6.2).
 - 0011B: PEBS record includes additional information of load latency data, TSX tuning information, TSC data, and the applicable counter field replaces IA32_PERF_GLOBAL_STATUS at offset 90H. (See Section 20.3.8.1.1).
 - 0100B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 20.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 20-64 with Counter Reset fields allocation for 8 general-purpose counters followed by 4 fixed-function counters.

- 0101B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 20.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 20-64 with Counter Reset fields allocation for 32 general-purpose counters followed by 16 fixed-function counters.

20.6.2.4.3 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the Interrupt-based event sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 18.4.9.1, “64 Bit Format of the DS Save Area,” for guidelines when writing the DS ISR.

The service routine can query MSR_PERF_GLOBAL_STATUS to determine which counter(s) caused of overflow condition. The service routine should clear overflow indicator by writing to MSR_PERF_GLOBAL_OVF_CTL.

A comparison of the sequence of requirements to program PEBS for processors based on Intel Core and Intel NetBurst microarchitectures is listed in Table 20-85.

Table 20-85. Requirements to Program PEBS

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Verify PEBS support of processor/OS.	<ul style="list-style-type: none"> ▪ IA32_MISC_ENABLE.EMON_AVAILABE (bit 7) is set. ▪ IA32_MISC_ENABLE.PEBS_UNAVAILABE (bit 12) is clear. 	
Ensure counters are in disabled.	<p>On initial set up or changing event configurations, write MSR_PERF_GLOBAL_CTRL MSR (38FH) with 0.</p> <p>On subsequent entries:</p> <ul style="list-style-type: none"> ▪ Clear all counters if “Counter Freeze on PMI” is not enabled. ▪ If IA32_DebugCTL.Freeze is enabled, counters are automatically disabled. <p>Counters MUST be stopped before writing.¹</p>	Optional
Disable PEBS.	Clear ENABLE PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Optional
Check overflow conditions.	Check MSR_PERF_GLOBAL_STATUS MSR (38EH) handle any overflow conditions.	Check OVF flag of each CCCR for overflow condition
Clear overflow status.	Clear MSR_PERF_GLOBAL_STATUS MSR (38EH) using IA32_PERF_GLOBAL_OVF_CTRL MSR (390H).	Clear OVF flag of each CCCR.
Write “sample-after” values.	Configure the counter(s) with the sample after value.	
Configure specific counter configuration MSR.	<ul style="list-style-type: none"> ▪ Set local enable bit 22 - 1. ▪ Do NOT set local counter PMI/INT bit, bit 20 - 0. ▪ Event programmed must be PEBS capable. 	<ul style="list-style-type: none"> ▪ Set appropriate OVF_PMI bits - 1. ▪ Only CCCR for MSR_IQ_COUNTER4 support PEBS.
Allocate buffer for PEBS states.	Allocate a buffer in memory for the precise information.	
Program the IA32_DS_AREA MSR.	Program the IA32_DS_AREA MSR.	
Configure the PEBS buffer management records.	Configure the PEBS buffer management records in the DS buffer management area.	
Configure/Enable PEBS.	Set Enable PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Configure MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_PEBS_MATRIX_HORZ as needed.
Enable counters.	Set Enable bits in MSR_PERF_GLOBAL_CTRL MSR (38FH).	Set each CCCR enable bit 12 - 1.

NOTES:

1. Counters read while enabled are not guaranteed to be precise with event counts that occur in timing proximity to the RDMSR.

20.6.2.4.4 Re-configuring PEBS Facilities

When software needs to reconfigure PEBS facilities, it should allow a quiescent period between stopping the prior event counting and setting up a new PEBS event. The quiescent period is to allow any latent residual PEBS records to complete its capture at their previously specified buffer address (provided by IA32_DS_AREA).

20.6.3 Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)

The performance monitoring mechanism provided in processors based on Intel NetBurst microarchitecture is different from that provided in the P6 family and Pentium processors. While the general concept of selecting, filtering, counting, and reading performance events through the WRMSR, RDMSR, and RDPMS instructions is unchanged, the setup mechanism and MSR layouts are incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMS instruction has been extended to support faster reading of counters and to read all performance counters available in processors based on Intel NetBurst microarchitecture.

The event monitoring mechanism consists of the following facilities:

- The IA32_MISC_ENABLE MSR, which indicates the availability in an Intel 64 or IA-32 processor of the performance monitoring and processor event-based sampling (PEBS) facilities.
- Event selection control (ESCR) MSRs for selecting events to be monitored with specific performance counters. The number available differs by family and model (43 to 45).
- 18 performance counter MSRs for counting events.
- 18 counter configuration control (CCCR) MSRs, with one CCCR associated with each performance counter. CCCRs sets up an associated performance counter for a specific method of counting.
- A debug store (DS) save area in memory for storing PEBS records.
- The IA32_DS_AREA MSR, which establishes the location of the DS save area.
- The debug store (DS) feature flag (bit 21) returned by the CPUID instruction, which indicates the availability of the DS mechanism.
- The MSR_PEBS_ENABLE MSR, which enables the PEBS facilities and replay tagging used in at-retirement event counting.
- A set of predefined events and event metrics that simplify the setting up of the performance counters to count specific events.

Table 20-86 lists the performance counters and their associated CCCRs, along with the ESCRs that select events to be counted for each performance counter. Predefined event metrics and events can be found at: <https://perfmon-events.intel.com/>.

Table 20-86. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Processors Based on Intel NetBurst Microarchitecture)

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCRO	7	3A0H
					MSR_FSB_ESCRO	6	3A2H
					MSR_MOB_ESCRO	2	3AAH
					MSR_PMH_ESCRO	4	3ACH
					MSR_BPU_ESCRO	0	3B2H
					MSR_IS_ESCRO	1	3B4H
					MSR_ITLB_ESCRO	3	3B6H
					MSR_IX_ESCRO	5	3C8H

**Table 20-86. Performance Counter MSRs and Associated CCCR and ESCR MSRs
(Processors Based on Intel NetBurst Microarchitecture) (Contd.)**

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_FLAME_COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAA_T_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_FLAME_COUNTER3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAA_T_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H

**Table 20-86. Performance Counter MSR and Associated CCCR and ESCR MSRs
(Processors Based on Intel NetBurst Microarchitecture) (Contd.)**

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0	4	3B8H
					MSR_CRU_ESCR2	5	3CCH
					MSR_CRU_ESCR4	6	3E0H
					MSR_IQ_ESCR0 ¹	0	3BAH
					MSR_RAT_ESCR0	2	3BCH
					MSR_SSU_ESCR0	3	3BEH
					MSR_ALF_ESCR0	1	3CAH
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0	4	3B8H
					MSR_CRU_ESCR2	5	3CCH
					MSR_CRU_ESCR4	6	3E0H
					MSR_IQ_ESCR0 ¹	0	3BAH
					MSR_RAT_ESCR0	2	3BCH
					MSR_SSU_ESCR0	3	3BEH
					MSR_ALF_ESCR0	1	3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1	4	3B9H
					MSR_CRU_ESCR3	5	3CDH
					MSR_CRU_ESCR5	6	3E1H
					MSR_IQ_ESCR1 ¹	0	3BBH
					MSR_RAT_ESCR1	2	3BDH
					MSR_ALF_ESCR1	1	3CBH
					MSR_IQ_COUNTER3	15	30FH
MSR_CRU_ESCR3	5	3CDH					
MSR_CRU_ESCR5	6	3E1H					
MSR_IQ_ESCR1 ¹	0	3BBH					
MSR_RAT_ESCR1	2	3BDH					
MSR_ALF_ESCR1	1	3CBH					
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H			
					MSR_CRU_ESCR2	5	3CCH
					MSR_CRU_ESCR4	6	3E0H
					MSR_IQ_ESCR0 ¹	0	3BAH
					MSR_RAT_ESCR0	2	3BCH
					MSR_SSU_ESCR0	3	3BEH
					MSR_ALF_ESCR0	1	3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1	4	3B9H
					MSR_CRU_ESCR3	5	3CDH
					MSR_CRU_ESCR5	6	3E1H
					MSR_IQ_ESCR1 ¹	0	3BBH
					MSR_RAT_ESCR1	2	3BDH
					MSR_ALF_ESCR1	1	3CBH

NOTES:

1. MSR_IQ_ESCR0 and MSR_IQ_ESCR1 are available only on early processor builds (family 0FH, models 01H-02H). These MSRs are not available on later versions.

The types of events that can be counted with these performance monitoring facilities are divided into two classes: non-retirement events and at-retirement events.

- Non-retirement events are events that occur any time during instruction execution (such as bus transactions or cache transactions).
- At-retirement events are events that are counted at the retirement stage of instruction execution, which allows finer granularity in counting events and capturing machine state.

The at-retirement counting mechanism includes facilities for tagging μ ops that have encountered a particular performance event during instruction execution. Tagging allows events to be sorted between those that occurred on an execution path that resulted in architectural state being committed at retirement as well as events that occurred on an execution path where the results were eventually cancelled and never committed to architectural state (such as, the execution of a mispredicted branch).

The Pentium 4 and Intel Xeon processor performance monitoring facilities support the three usage models described below. The first two models can be used to count both non-retirement and at-retirement events; the third model is used to count a subset of at-retirement events:

- **Event counting** — A performance counter is configured to count one or more types of events. While the counter is counting, software reads the counter at selected intervals to determine the number of events that have been counted between the intervals.
- **Interrupt-based event sampling** — A performance counter is configured to count one or more types of events and to generate an interrupt when it overflows. To trigger an overflow, the counter is preset to a modulus value that will cause the counter to overflow after a specific number of events have been counted. When the counter overflows, the processor generates a performance monitoring interrupt (PMI). The interrupt service routine for the PMI then records the return instruction pointer (RIP), resets the modulus, and restarts the counter. Code performance can be analyzed by examining the distribution of RIPs with a tool like the VTune™ Performance Analyzer.
- **Processor event-based sampling (PEBS)** — In PEBS, the processor writes a record of the architectural state of the processor to a memory buffer after the counter overflows. The records of architectural state provide additional information for use in performance tuning. Processor-based event sampling can be used to count only a subset of at-retirement events. PEBS captures more precise processor state information compared to interrupt based event sampling, because the latter need to use the interrupt service routine to re-construct the architectural states of processor.

The following sections describe the MSR and data structures used for performance monitoring in the Pentium 4 and Intel Xeon processors.

20.6.3.1 ESCR MSRs

The 45 ESCR MSRs (see Table 20-86) allow software to select specific events to be countered. Each ESCR is usually associated with a pair of performance counters (see Table 20-86) and each performance counter has several ESCRs associated with it (allowing the events counted to be selected from a variety of events).

Figure 20-47 shows the layout of an ESCR MSR. The functions of the flags and fields are:

- **USR flag, bit 2** — When set, events are counted when the processor is operating at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.
- **OS flag, bit 3** — When set, events are counted when the processor is operating at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the OS and USR flags are set, events are counted at all privilege levels.)

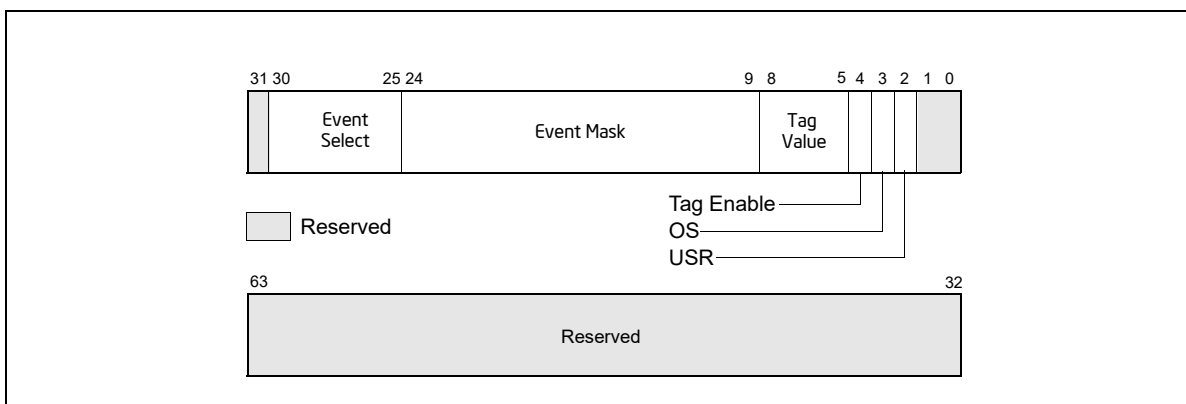


Figure 20-47. Event Selection Control Register (ESCR) for Pentium 4 and Intel® Xeon® Processors without Intel HT Technology Support

- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 20.6.3.6, "At-Retirement Counting."

- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

When setting up an ESCR, the event select field is used to select a specific class of events to count, such as retired branches. The event mask field is then used to select one or more of the specific events within the class to be counted. For example, when counting retired branches, four different events can be counted: branch not taken predicted, branch not taken mispredicted, branch taken predicted, and branch taken mispredicted. The OS and USR flags allow counts to be enabled for events that occur when operating system code and/or application code are being executed. If neither the OS nor USR flag is set, no events will be counted.

The ESCRs are initialized to all 0s on reset. The flags and fields of an ESCR are configured by writing to the ESCR using the WRMSR instruction. Table 20-86 gives the addresses of the ESCR MSRs.

Writing to an ESCR MSR does not enable counting with its associated performance counter; it only selects the event or events to be counted. The CCCR for the selected performance counter must also be configured. Configuration of the CCCR includes selecting the ESCR and enabling the counter.

20.6.3.2 Performance Counters

The performance counters in conjunction with the counter configuration control registers (CCCRs) are used for filtering and counting the events selected by the ESCRs. Processors based on Intel NetBurst microarchitecture provide 18 performance counters organized into 9 pairs. A pair of performance counters is associated with a particular subset of events and ESCR's (see Table 20-86). The counter pairs are partitioned into four groups:

- The BPU group, includes two performance counter pairs:
 - MSR_BPU_COUNTER0 and MSR_BPU_COUNTER1.
 - MSR_BPU_COUNTER2 and MSR_BPU_COUNTER3.
- The MS group, includes two performance counter pairs:
 - MSR_MS_COUNTER0 and MSR_MS_COUNTER1.
 - MSR_MS_COUNTER2 and MSR_MS_COUNTER3.
- The FLAME group, includes two performance counter pairs:
 - MSR_FLAME_COUNTER0 and MSR_FLAME_COUNTER1.
 - MSR_FLAME_COUNTER2 and MSR_FLAME_COUNTER3.
- The IQ group, includes three performance counter pairs:
 - MSR_IQ_COUNTER0 and MSR_IQ_COUNTER1.
 - MSR_IQ_COUNTER2 and MSR_IQ_COUNTER3.
 - MSR_IQ_COUNTER4 and MSR_IQ_COUNTER5.

The MSR_IQ_COUNTER4 counter in the IQ group provides support for the PEBS.

Alternate counters in each group can be cascaded: the first counter in one pair can start the first counter in the second pair and vice versa. A similar cascading is possible for the second counters in each pair. For example, within the BPU group of counters, MSR_BPU_COUNTER0 can start MSR_BPU_COUNTER2 and vice versa, and MSR_BPU_COUNTER1 can start MSR_BPU_COUNTER3 and vice versa (see Section 20.6.3.5.6, "Cascading Counters"). The cascade flag in the CCCR register for the performance counter enables the cascading of counters.

Each performance counter is 40-bits wide (see Figure 20-48). The RDPMC instruction is intended to allow reading of either the full counter-width (40-bits) or, if ECX[31] is set to 1, the low 32-bits of the counter. Reading the low 32-bits is faster than reading the full counter width and is appropriate in situations where the count is small enough to be contained in 32 bits. In such cases, counter bits 31:0 are written to EAX, while 0 is written to EDX.

The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

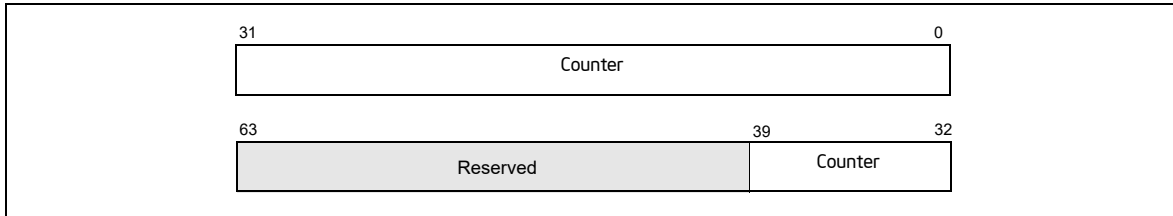


Figure 20-48. Performance Counter (Pentium 4 and Intel® Xeon® Processors)

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

Some uses of the performance counters require the counters to be preset before counting begins (that is, before the counter is enabled). This can be accomplished by writing to the counter using the WRMSR instruction. To set a counter to a specified number of counts before overflow, enter a 2s complement negative integer in the counter. The counter will then count from the preset value up to -1 and overflow. Writing to a performance counter in a Pentium 4 or Intel Xeon processor with the WRMSR instruction causes all 40 bits of the counter to be written.

20.6.3.3 CCCR MSRs

Each of the 18 performance counters has one CCCR MSR associated with it (see Table 20-86). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 20-49 shows the layout of an CCCR MSR. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset.
- **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.
- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 20.6.3.5.2, "Filtering Events"). The complement flag is not active unless the compare flag is set.
- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 20.6.3.5.2, "Filtering Events").
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

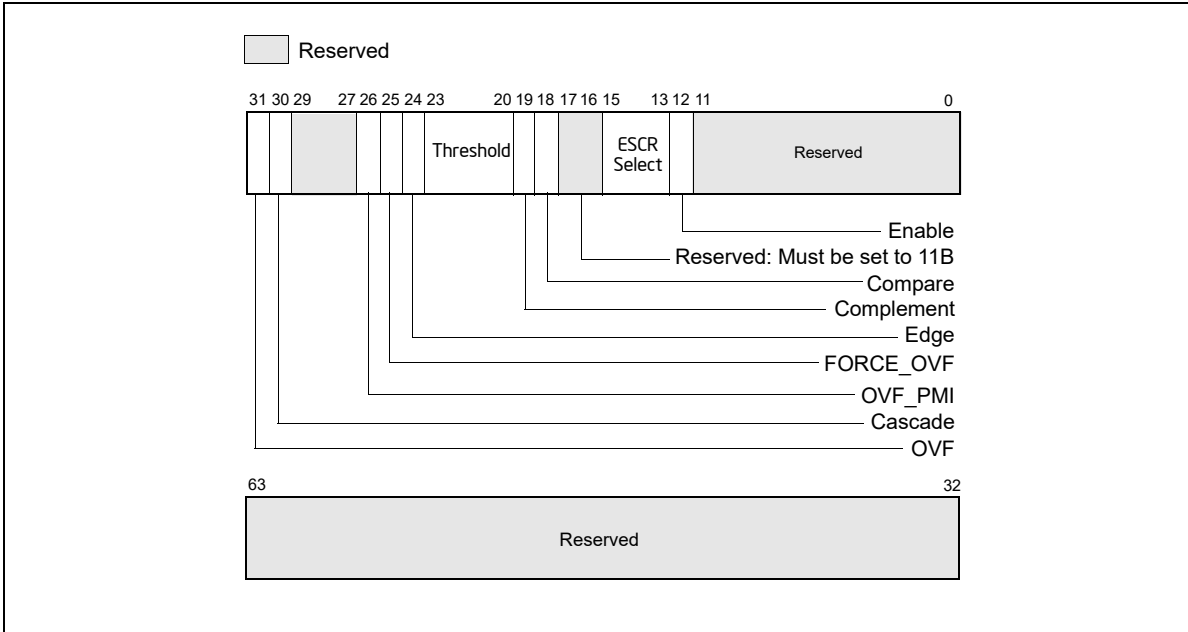


Figure 20-49. Counter Configuration Control Register (CCCR)

- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be generated when the counter overflows occurs; when clear, disables PMI generation. Note that the PMI is generated on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 20.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

The CCCRs are initialized to all 0s on reset.

The events that an enabled performance counter actually counts are selected and filtered by the following flags and fields in the ESCR and CCCR registers and in the qualification order given:

1. The event select and event mask fields in the ESCR select a class of events to be counted and one or more event types within the class, respectively.
2. The OS and USR flags in the ESCR selected the privilege levels at which events will be counted.
3. The ESCR select field of the CCCR selects the ESCR. Since each counter has several ESCRs associated with it, one ESCR must be chosen to select the classes of events that may be counted.
4. The compare and complement flags and the threshold field of the CCCR select an optional threshold to be used in qualifying an event count.
5. The edge flag in the CCCR allows events to be counted only on rising-edge transitions.

The qualification order in the above list implies that the filtered output of one “stage” forms the input for the next. For instance, events filtered using the privilege level flags can be further qualified by the compare and complement flags and the threshold field, and an event that matched the threshold criteria, can be further qualified by edge detection.

The uses of the flags and fields in the CCCRs are discussed in greater detail in Section 20.6.3.5, “Programming the Performance Counters for Non-Retirement Events.”

20.6.3.4 Debug Store (DS) Mechanism

The debug store (DS) mechanism was introduced with processors based on Intel NetBurst microarchitecture to allow various types of information to be collected in memory-resident buffers for use in debugging and tuning programs. The DS mechanism can be used to collect two types of information: branch records and processor event-based sampling (PEBS) records. The availability of the DS mechanism in a processor is indicated with the DS feature flag (bit 21) returned by the CPUID instruction.

See Section 18.4.5, “Branch Trace Store (BTS),” and Section 20.6.3.8, “Processor Event-Based Sampling (PEBS),” for a description of these facilities. Records collected with the DS mechanism are saved in the DS save area. See Section 18.4.9, “BTS and DS Save Area.”

20.6.3.5 Programming the Performance Counters for Non-Retirement Events

The basic steps to program a performance counter and to count events include the following:

1. Select the event or events to be counted.
2. For each event, select an ESCR that supports the event.
3. Match the CCCR Select value and ESCR name to a value listed in Table 20-86; select a CCCR and performance counter.
4. Set up an ESCR for the specific event or events to be counted and the privilege levels at which they are to be counted.
5. Set up the CCCR for the performance counter by selecting the ESCR and the desired event filters.
6. Set up the CCCR for optional cascading of event counts, so that when the selected counter overflows its alternate counter starts.
7. Set up the CCCR to generate an optional performance monitor interrupt (PMI) when the counter overflows. If PMI generation is enabled, the local APIC must be set up to deliver the interrupt to the processor and a handler for the interrupt must be in place.
8. Enable the counter to begin counting.

20.6.3.5.1 Selecting Events to Count

There is a set of at-retirement events for processors based on Intel NetBurst microarchitecture. For each event, setup information is provided. Table 20-87 gives an example of one of the events.

Table 20-87. Event Example

Event Name	Event Parameters	Parameter Value	Description
branch_retired			Counts the retirement of a branch. Specify one or more mask bits to select any combination of branch taken, not-taken, predicted, and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 15-3 for the addresses of the ESCR MSRs.
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 15-3.
	ESCR Event Select	06H	ESCR[31:25]
	ESCR Event Mask	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9] Branch Not-taken Predicted Branch Not-taken Mispredicted Branch Taken Predicted Branch Taken Mispredicted
	CCCR Select	05H	CCCR[15:13]

Table 20-87. Event Example (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	Requires Additional MSRs for Tagging	No	

Event Parameters are described below.

- **ESCR restrictions** — Lists the ESCRs that can be used to program the event. Typically only one ESCR is needed to count an event.
- **Counter numbers per ESCR** — Lists which performance counters are associated with each ESCR. Table 20-86 gives the name of the counter and CCCR for each counter number. Typically only one counter is needed to count the event.
- **ESCR event select** — Gives the value to be placed in the event select field of the ESCR to select the event.
- **ESCR event mask** — Gives the value to be placed in the Event Mask field of the ESCR to select sub-events to be counted. The parameter value column defines the documented bits with relative bit position offset starting from 0, where the absolute bit position of relative offset 0 is bit 9 of the ESCR. All undocumented bits are reserved and should be set to 0.
- **CCCR select** — Gives the value to be placed in the ESCR select field of the CCCR associated with the counter to select the ESCR to be used to define the event. This value is not the address of the ESCR; it is the number of the ESCR from the Number column in Table 20-86.
- **Event specific notes** — Gives additional information about the event, such as the name of the same or a similar event defined for the P6 family processors.
- **Can support PEBS** — Indicates if PEBS is supported for the event (only supplied for at-retirement events).
- **Requires additional MSR for tagging** — Indicates which if any additional MSRs must be programmed to count the events (only supplied for the at-retirement events).

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The following procedure shows how to set up a performance counter for basic counting; that is, the counter is set up to count a specified event indefinitely, wrapping around whenever it reaches its maximum count. This procedure is continued through the following four sections.

An event to be counted can be selected as follows:

1. Select the event to be counted.
2. Select the ESCR to be used to select events to be counted from the ESCRs field.
3. Select the number of the counter to be used to count the event from the Counter Numbers Per ESCR field.
4. Determine the name of the counter and the CCCR associated with the counter, and determine the MSR addresses of the counter, CCCR, and ESCR from Table 20-86.
5. Use the WRMSR instruction to write the ESCR Event Select and ESCR Event Mask values into the appropriate fields in the ESCR. At the same time set or clear the USR and OS flags in the ESCR as desired.
6. Use the WRMSR instruction to write the CCCR Select value into the appropriate field in the CCCR.

NOTE

Typically all the fields and flags of the CCCR will be written with one WRMSR instruction; however, in this procedure, several WRMSR writes are used to more clearly demonstrate the uses of the various CCCR fields and flags.

This setup procedure is continued in the next section, Section 20.6.3.5.2, “Filtering Events.”

20.6.3.5.2 Filtering Events

Each counter receives up to 4 input lines from the processor hardware from which it is counting events. The counter treats these inputs as binary inputs (input 0 has a value of 1, input 1 has a value of 2, input 3 has a value of 4, and input 3 has a value of 8). When a counter is enabled, it adds this binary input value to the counter value on each clock cycle. For each clock cycle, the value added to the counter can then range from 0 (no event) to 15.

For many events, only the 0 input line is active, so the counter is merely counting the clock cycles during which the 0 input is asserted. However, for some events two or more input lines are used. Here, the counter's threshold setting can be used to filter events. The compare, complement, threshold, and edge fields control the filtering of counter increments by input value.

If the compare flag is set, then a “greater than” or a “less than or equal to” comparison of the input value vs. a threshold value can be made. The complement flag selects “less than or equal to” (flag set) or “greater than” (flag clear). The threshold field selects a threshold value of from 0 to 15. For example, if the complement flag is cleared and the threshold field is set to 6, then any input value of 7 or greater on the 4 inputs to the counter will cause the counter to be incremented by 1, and any value less than 7 will cause an increment of 0 (or no increment) of the counter. Conversely, if the complement flag is set, any value from 0 to 6 will increment the counter and any value from 7 to 15 will not increment the counter. Note that when a threshold condition has been satisfied, the input to the counter is always 1, not the input value that is presented to the threshold filter.

The edge flag provides further filtering of the counter inputs when a threshold comparison is being made. The edge flag is only active when the compare flag is set. When the edge flag is set, the resulting output from the threshold filter (a value of 0 or 1) is used as an input to the edge filter. Each clock cycle, the edge filter examines the last and current input values and sends a count to the counter only when it detects a “rising edge” event; that is, a false-to-true transition. Figure 20-50 illustrates rising edge filtering.

The following procedure shows how to configure a CCCR to filter events using the threshold filter and the edge filter. This procedure is a continuation of the setup procedure introduced in Section 20.6.3.5.1, “Selecting Events to Count.”

7. (Optional) To set up the counter for threshold filtering, use the WRMSR instruction to write values in the CCCR compare and complement flags and the threshold field:
 - Set the compare flag.
 - Set or clear the complement flag for less than or equal to or greater than comparisons, respectively.
 - Enter a value from 0 to 15 in the threshold field.
8. (Optional) Select rising edge filtering by setting the CCCR edge flag.

This setup procedure is continued in the next section, Section 20.6.3.5.3, “Starting Event Counting.”

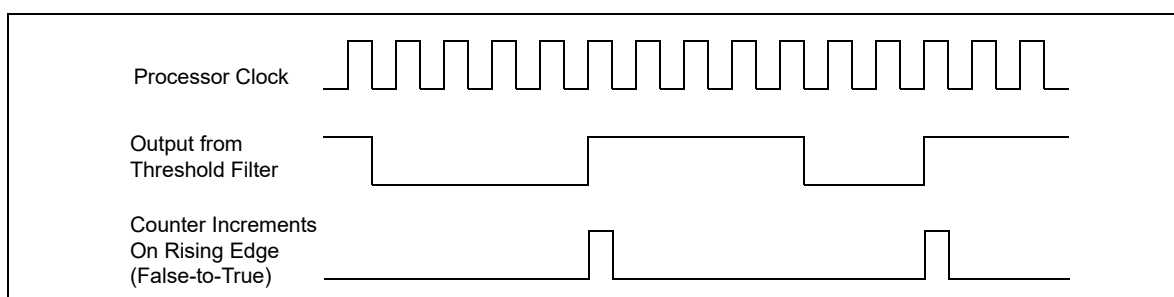


Figure 20-50. Effects of Edge Filtering

20.6.3.5.3 Starting Event Counting

Event counting by a performance counter can be initiated in either of two ways. The typical way is to set the enable flag in the counter's CCCR. Following the instruction to set the enable flag, event counting begins and continues until it is stopped (see Section 20.6.3.5.5, "Halting Event Counting").

The following procedural step shows how to start event counting. This step is a continuation of the setup procedure introduced in Section 20.6.3.5.2, "Filtering Events."

9. To start event counting, use the WRMSR instruction to set the CCCR enable flag for the performance counter.

This setup procedure is continued in the next section, Section 20.6.3.5.4, "Reading a Performance Counter's Count."

The second way that a counter can be started by using the cascade feature. Here, the overflow of one counter automatically starts its alternate counter (see Section 20.6.3.5.6, "Cascading Counters").

20.6.3.5.4 Reading a Performance Counter's Count

Performance counters can be read using either the RDPMC or RDMSR instructions. The enhanced functions of the RDPMC instruction (including fast read) are described in Section 20.6.3.2, "Performance Counters." These instructions can be used to read a performance counter while it is counting or when it is stopped.

The following procedural step shows how to read the event counter. This step is a continuation of the setup procedure introduced in Section 20.6.3.5.3, "Starting Event Counting."

10. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 20-86 used as an operand.

This setup procedure is continued in the next section, Section 20.6.3.5.5, "Halting Event Counting."

20.6.3.5.5 Halting Event Counting

After a performance counter has been started (enabled), it continues counting indefinitely. If the counter overflows (goes one count past its maximum count), it wraps around and continues counting. When the counter wraps around, it sets its OVF flag to indicate that the counter has overflowed. The OVF flag is a sticky flag that indicates that the counter has overflowed at least once since the OVF bit was last cleared.

To halt counting, the CCCR enable flag for the counter must be cleared.

The following procedural step shows how to stop event counting. This step is a continuation of the setup procedure introduced in Section 20.6.3.5.4, "Reading a Performance Counter's Count."

11. To stop event counting, execute a WRMSR instruction to clear the CCCR enable flag for the performance counter.

To halt a cascaded counter (a counter that was started when its alternate counter overflowed), either clear the Cascade flag in the cascaded counter's CCCR MSR or clear the OVF flag in the alternate counter's CCCR MSR.

20.6.3.5.6 Cascading Counters

As described in Section 20.6.3.2, "Performance Counters," eighteen performance counters are implemented in pairs. Nine pairs of counters and associated CCCRs are further organized as four blocks: BPU, MS, FLAME, and IQ (see Table 20-86). The first three blocks contain two pairs each. The IQ block contains three pairs of counters (12 through 17) with associated CCCRs (MSR_IQ_CCCR0 through MSR_IQ_CCCR5).

The first 8 counter pairs (0 through 15) can be programmed using ESCRs to detect performance monitoring events. Pairs of ESCRs in each of the four blocks allow many different types of events to be counted. The cascade flag in the CCCR MSR allows nested monitoring of events to be performed by cascading one counter to a second counter located in another pair in the same block (see Figure 20-49 for the location of the flag).

Counters 0 and 1 form the first pair in the BPU block. Either counter 0 or 1 can be programmed to detect an event via MSR_MO B_ESCR0. Counters 0 and 2 can be cascaded in any order, as can counters 1 and 3. It's possible to set up 4 counters in the same block to cascade on two pairs of independent events. The pairing described also applies to subsequent blocks. Since the IQ PUB has two extra counters, cascading operates somewhat differently if 16 and 17 are involved. In the IQ block, counter 16 can only be cascaded from counter 14 (not from 12); counter 14

cannot be cascaded from counter 16 using the CCCR cascade bit mechanism. Similar restrictions apply to counter 17.

Example 20-1. Counting Events

Assume a scenario where counter X is set up to count 200 occurrences of event A; then counter Y is set up to count 400 occurrences of event B. Each counter is set up to count a specific event and overflow to the next counter. In the above example, counter X is preset for a count of -200 and counter Y for a count of -400; this setup causes the counters to overflow on the 200th and 400th counts respectively.

Continuing this scenario, counter X is set up to count indefinitely and wraparound on overflow. This is described in the basic performance counter setup procedure that begins in Section 20.6.3.5.1, "Selecting Events to Count." Counter Y is set up with the cascade flag in its associated CCCR MSR set to 1 and its enable flag set to 0.

To begin the nested counting, the enable bit for the counter X is set. Once enabled, counter X counts until it overflows. At this point, counter Y is automatically enabled and begins counting. Thus counter X overflows after 200 occurrences of event A. Counter Y then starts, counting 400 occurrences of event B before overflowing. When performance counters are cascaded, the counter Y would typically be set up to generate an interrupt on overflow. This is described in Section 20.6.3.5.8, "Generating an Interrupt on Overflow."

The cascading counters mechanism can be used to count a single event. The counting begins on one counter then continues on the second counter after the first counter overflows. This technique doubles the number of event counts that can be recorded, since the contents of the two counters can be added together.

20.6.3.5.7 EXTENDED CASCADING

Extended cascading is a model-specific feature in the Intel NetBurst microarchitecture with CPUID DisplayFamily_DisplayModel 0F_02, 0F_03, 0F_04, 0F_06. This feature uses bit 11 in CCCRs associated with the IQ block. See Table 20-88.

Table 20-88. CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1 2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INT00	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INT03	Allow counter 5 to cascade into counter 3
MSR_IQ_CCCR4:11	CASCNT5INT04	Allow counter 5 to cascade into counter 4
MSR_IQ_CCCR5:11	CASCNT4INT05	Allow counter 4 to cascade into counter 5

The extended cascading feature can be adapted to the Interrupt based sampling usage model for performance monitoring. However, it is known that performance counters do not generate PMI in cascade mode or extended cascade mode due to an erratum. This erratum applies to processors with CPUID DisplayFamily_DisplayModel signature of 0F_02. For processors with CPUID DisplayFamily_DisplayModel signature of 0F_00 and 0F_01, the erratum applies to processors with stepping encoding greater than 09H.

Counters 16 and 17 in the IQ block are frequently used in processor event-based sampling or at-retirement counting of events indicating a stalled condition in the pipeline. Neither counter 16 or 17 can initiate the cascading of counter pairs using the cascade bit in a CCCR.

Extended cascading permits performance monitoring tools to use counters 16 and 17 to initiate cascading of two counters in the IQ block. Extended cascading from counter 16 and 17 is conceptually similar to cascading other counters, but instead of using CASCADE bit of a CCCR, one of the four CASCNTxINT0y bits is used.

Example 20-2. Scenario for Extended Cascading

A usage scenario for extended cascading is to sample instructions retired on logical processor 1 after the first 4096 instructions retired on logical processor 0. A procedure to program extended cascading in this scenario is outlined below:

1. Write the value 0 to counter 12.
2. Write the value 04000603H to MSR_CRU_ESCR0 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 1).
3. Write the value 04038800H to MSR_IQ_CCCR0. This enables CASCNT4INT00 and OVF_PMI. An ISR can sample on instruction addresses in this case (do not set ENABLE, or CASCADE).
4. Write the value FFFF000H into counter 16.1.
5. Write the value 0400060CH to MSR_CRU_ESCR2 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 0).
6. Write the value 00039000H to MSR_IQ_CCCR4 (set ENABLE bit, but not OVF_PMI).

Another use for cascading is to locate stalled execution in a multithreaded application. Assume MOB replays in thread B cause thread A to stall. Getting a sample of the stalled execution in this scenario could be accomplished by:

1. Set up counter B to count MOB replays on thread B.
2. Set up counter A to count resource stalls on thread A; set its force overflow bit and the appropriate CASCNTx-INTOy bit.
3. Use the performance monitoring interrupt to capture the program execution data of the stalled thread.

20.6.3.5.8 Generating an Interrupt on Overflow

Any performance counter can be configured to generate a performance monitor interrupt (PMI) if the counter overflows. The PMI interrupt service routine can then collect information about the state of the processor or program when overflow occurred. This information can then be used with a tool like the Intel® VTune™ Performance Analyzer to analyze and tune program performance.

To enable an interrupt on counter overflow, the OVR_PMI flag in the counter's associated CCCR MSR must be set. When overflow occurs, a PMI is generated through the local APIC. (Here, the performance counter entry in the local vector table [LVT] is set up to deliver the interrupt generated by the PMI to the processor.)

The PMI service routine can use the OVF flag to determine which counter overflowed when multiple counters have been configured to generate PMIs. Also, note that these processors mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

When generating interrupts on overflow, the performance counter being used should be preset to value that will cause an overflow after a specified number of events are counted plus 1. The simplest way to select the preset value is to write a negative number into the counter, as described in Section 20.6.3.5.6, "Cascading Counters." Here, however, if an interrupt is to be generated after 100 event counts, the counter should be preset to minus 100 plus 1 (-100 + 1), or -99. The counter will then overflow after it counts 99 events and generate an interrupt on the next (100th) event counted. The difference of 1 for this count enables the interrupt to be generated immediately after the selected event count has been reached, instead of waiting for the overflow to be propagation through the counter.

Because of latency in the microarchitecture between the generation of events and the generation of interrupts on overflow, it is sometimes difficult to generate an interrupt close to an event that caused it. In these situations, the FORCE_OVF flag in the CCCR can be used to improve reporting. Setting this flag causes the counter to overflow on every counter increment, which in turn triggers an interrupt after every counter increment.

20.6.3.5.9 Counter Usage Guideline

There are some instances where the user must take care to configure counting logic properly, so that it is not powered down. To use any ESCR, even when it is being used just for tagging, (any) one of the counters that the particular ESCR (or its paired ESCR) can be connected to should be enabled. If this is not done, 0 counts may result. Likewise, to use any counter, there must be some event selected in a corresponding ESCR (other than no_event, which generally has a select value of 0).

20.6.3.6 At-Retirement Counting

At-retirement counting provides a means counting only events that represent work committed to architectural state and ignoring work that was performed speculatively and later discarded.

One example of this speculative activity is branch prediction. When a branch misprediction occurs, the results of instructions that were decoded and executed down the mispredicted path are canceled. If a performance counter was set up to count all executed instructions, the count would include instructions whose results were canceled as well as those whose results committed to architectural state.

To provide finer granularity in event counting in these situations, the performance monitoring facilities provided in the Pentium 4 and Intel Xeon processors provide a mechanism for tagging events and then counting only those tagged events that represent committed results. This mechanism is called “at-retirement counting.”

There are predefined at-retirement events and event metrics that can be used to for tagging events when using at retirement counting. The following terminology is used in describing at-retirement counting:

- **Bogus, non-bogus, retire** — In at-retirement event descriptions, the term “bogus” refers to instructions or μ ops that must be canceled because they are on a path taken from a mispredicted branch. The terms “retired” and “non-bogus” refer to instructions or μ ops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and μ ops are either bogus or non-bogus, but not both. Several of the Pentium 4 and Intel Xeon processors’ performance monitoring events (such as, `Instruction_Retired` and `Uops_Retired`) can count instructions or μ ops that are retired based on the characterization of bogus” versus non-bogus.
- **Tagging** — Tagging is a means of marking μ ops that have encountered a particular performance event so they can be counted at retirement. During the course of execution, the same event can happen more than once per μ op and a direct count of the event would not provide an indication of how many μ ops encountered that event. The tagging mechanisms allow a μ op to be tagged once during its lifetime and thus counted once at retirement. The retired suffix is used for performance metrics that increment a count once per μ op, rather than once per event. For example, a μ op may encounter a cache miss more than once during its life time, but a “Miss Retired” metric (that counts the number of retired μ ops that encountered a cache miss) will increment only once for that μ op. A “Miss Retired” metric would be useful for characterizing the performance of the cache hierarchy for a particular instruction sequence. Details of various performance metrics and how these can be constructed using the Pentium 4 and Intel Xeon processors performance events are provided in the *Intel Pentium 4 Processor Optimization Reference Manual* (see Section 1.4, “Related Literature”).
- **Replay** — To maximize performance for the common case, the Intel NetBurst microarchitecture aggressively schedules μ ops for execution before all the conditions for correct execution are guaranteed to be satisfied. In the event that all of these conditions are not satisfied, μ ops must be reissued. The mechanism that the Pentium 4 and Intel Xeon processors use for this reissuing of μ ops is called replay. Some examples of replay causes are cache misses, dependence violations, and unforeseen resource constraints. In normal operation, some number of replays is common and unavoidable. An excessive number of replays is an indication of a performance problem.
- **Assist** — When the hardware needs the assistance of microcode to deal with some event, the machine takes an assist. One example of this is an underflow condition in the input operands of a floating-point operation. The hardware must internally modify the format of the operands in order to perform the computation. Assists clear the entire machine of μ ops before they begin and are costly.

20.6.3.6.1 Using At-Retirement Counting

Processors based on Intel NetBurst microarchitecture allow counting both events and μ ops that encountered a specified event. For a subset of the at-retirement events, a μ op may be tagged when it encounters that event. The tagging mechanisms can be used in Interrupt-based event sampling, and a subset of these mechanisms can be used in PEBS. There are four independent tagging mechanisms, and each mechanism uses a different event to count μ ops tagged with that mechanism:

- **Front-end tagging** — This mechanism pertains to the tagging of μ ops that encountered front-end events (for example, trace cache and instruction counts) and are counted with the `Front_end_event` event.
- **Execution tagging** — This mechanism pertains to the tagging of μ ops that encountered execution events (for example, instruction types) and are counted with the `Execution_Event` event.

- **Replay tagging** — This mechanism pertains to tagging of μ ops whose retirement is replayed (for example, a cache miss) and are counted with the `Replay_event` event. Branch mispredictions are also tagged with this mechanism.
- **No tags** — This mechanism does not use tags. It uses the `Instr_retired` and the `Uops_retired` events.

Each tagging mechanism is independent from all others; that is, a μ op that has been tagged using one mechanism will not be detected with another mechanism's tagged- μ op detector. For example, if μ ops are tagged using the front-end tagging mechanisms, the `Replay_event` will not count those as tagged μ ops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of μ ops to be counted at retirement through execution tagging.

The independence of tagging mechanisms does not hold when using PEBS. When using PEBS, only one tagging mechanism should be used at a time.

Certain kinds of μ ops that cannot be tagged, including I/O, uncacheable and locked accesses, returns, and far transfers.

There are performance monitoring events that support at-retirement counting: specifically the `Front_end_event`, `Execution_event`, `Replay_event`, `Inst_retired`, and `Uops_retired` events. The following sections describe the tagging mechanisms for using these events to tag μ op and count tagged μ ops.

20.6.3.6.2 Tagging Mechanism for `Front_end_event`

The `Front_end_event` counts μ ops that have been tagged as encountering any of the following events:

- **μ op decode events** — Tagging μ ops for μ op decode events requires specifying bits in the `ESCR` associated with the performance-monitoring event, `Uop_type`.
- **Trace cache events** — Tagging μ ops for trace cache events may require specifying certain bits in the `MSR_TC_PRECISE_EVENT` MSR.

The MSRs that are supported by the front-end tagging mechanism must be set and one or both of the `NBOGUS` and `BOGUS` bits in the `Front_end_event` event mask must be set to count events. None of the events currently supported requires the use of the `MSR_TC_PRECISE_EVENT` MSR.

20.6.3.6.3 Tagging Mechanism For `Execution_event`

The execution tagging mechanism differs from other tagging mechanisms in how it causes tagging. One *upstream* `ESCR` is used to specify an event to detect and to specify a tag value (bits 5 through 8) to identify that event. A second *downstream* `ESCR` is used to detect μ ops that have been tagged with that tag value identifier using `Execution_event` for the event selection.

The upstream `ESCR` that counts the event must have its tag enable flag (bit 4) set and must have an appropriate tag value mask entered in its tag value field. The 4-bit tag value mask specifies which of tag bits should be set for a particular μ op. The value selected for the tag value should coincide with the event mask selected in the downstream `ESCR`. For example, if a tag value of 1 is set, then the event mask of `NBOGUS0` should be enabled, correspondingly in the downstream `ESCR`. The downstream `ESCR` detects and counts tagged μ ops. The normal (not tag value) mask bits in the downstream `ESCR` specify which tag bits to count. If any one of the tag bits selected by the mask is set, the related counter is incremented by one. The tag enable and tag value bits are irrelevant for the downstream `ESCR` used to select the `Execution_event`.

The four separate tag bits allow the user to simultaneously but distinctly count up to four execution events at retirement. (This applies for interrupt-based event sampling. There are additional restrictions for PEBS as noted in Section 20.6.3.8.3, "Setting Up the PEBS Buffer.") It is also possible to detect or count combinations of events by setting multiple tag value bits in the upstream `ESCR` or multiple mask bits in the downstream `ESCR`. For example, use a tag value of 3H in the upstream `ESCR` and use `NBOGUS0/NBOGUS1` in the downstream `ESCR` event mask.

20.6.3.7 Tagging Mechanism for `Replay_event`

The replay mechanism enables tagging of μ ops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of μ op that may experience the replay in the `MSR_PEBS_MATRIX_VERT` MSR and selecting the type of event in the `MSR_PEBS_ENABLE` MSR. Replay tagging must also be enabled with the `UOP_Tag` flag (bit 24) in the `MSR_PEBS_ENABLE` MSR.

The replay tags defined in Table A-5 also enable Processor Event-Based Sampling (PEBS, see Section 18.4.9). Each of these replay tags can also be used in normal sampling by not setting Bit 24 nor Bit 25 in IA_32_PEBS_ENABLE_MSR. Each of these metrics requires that the Replay_Event be used to count the tagged μ ops.

20.6.3.8 Processor Event-Based Sampling (PEBS)

The debug store (DS) mechanism in processors based on Intel NetBurst microarchitecture allow two types of information to be collected for use in debugging and tuning programs: PEBS records and BTS records. See Section 18.4.5, “Branch Trace Store (BTS),” for a description of the BTS mechanism.

PEBS permits the saving of precise architectural information associated with one or more performance events in the precise event records buffer, which is part of the DS save area (see Section 18.4.9, “BTS and DS Save Area”). To use this mechanism, a counter is configured to overflow after it has counted a preset number of events. After the counter overflows, the processor copies the current state of the general-purpose and EFLAGS registers and instruction pointer into a record in the precise event records buffer. The processor then resets the count in the performance counter and restarts the counter. When the precise event records buffer is nearly full, an interrupt is generated, allowing the precise event records to be saved. A circular buffer is not supported for precise event records.

PEBS is supported only for a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can only be carried out using the one performance counter, the MSR_IQ_COUNTER4 MSR.

In processors based on Intel Core microarchitecture, a similar PEBS mechanism is also supported using IA32_PMC0 and IA32_PERFEVTSEL0 MSRs (See Section 20.6.2.4).

20.6.3.8.1 Detection of the Availability of the PEBS Facilities

The DS feature flag (bit 21) returned by the CPUID instruction indicates (when set) the availability of the DS mechanism in the processor, which supports the PEBS (and BTS) facilities. When this bit is set, the following PEBS facilities are available:

- The PEBS_UNAVAILABLE flag in the IA32_MISC_ENABLE MSR indicates (when clear) the availability of the PEBS facilities, including the MSR_PEBS_ENABLE MSR.
- The enable PEBS flag (bit 24) in the MSR_PEBS_ENABLE MSR allows PEBS to be enabled (set) or disabled (clear).
- The IA32_DS_AREA MSR can be programmed to point to the DS save area.

20.6.3.8.2 Setting Up the DS Save Area

Section 18.4.9.2, “Setting Up the DS Save Area,” describes how to set up and enable the DS save area. This procedure is common for PEBS and BTS.

20.6.3.8.3 Setting Up the PEBS Buffer

Only the MSR_IQ_COUNTER4 performance counter can be used for PEBS. Use the following procedure to set up the processor and this counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, and precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area (see Figure 18-5) to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS flag (bit 24) in MSR_PEBS_ENABLE MSR.
3. Set up the MSR_IQ_COUNTER4 performance counter and its associated CCCR and one or more ESCRs for PEBS.

20.6.3.8.4 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the non-precise event-based sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 18.4.9.5, “Writing the DS Interrupt Service Routine,” for guidelines for writing the DS ISR.

20.6.3.8.5 Other DS Mechanism Implications

The DS mechanism is not available in the SMM. It is disabled on transition to the SMM mode. Similarly the DS mechanism is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT. The DS mechanism is available in real address mode.

20.6.3.9 Operating System Implications

The DS mechanism can be used by the operating system as a debugging extension to facilitate failure analysis. When using this facility, a 25 to 30 times slowdown can be expected due to the effects of the trace store occurring on every taken branch.

Depending upon intended usage, the instruction pointers that are part of the branch records or the PEBS records need to have an association with the corresponding process. One solution requires the ability for the DS specific operating system module to be chained to the context switch. A separate buffer can then be maintained for each process of interest and the MSR pointing to the configuration area saved and setup appropriately on each context switch.

If the BTS facility has been enabled, then it must be disabled and state stored on transition of the system to a sleep state in which processor context is lost. The state must be restored on return from the sleep state.

It is required that an interrupt gate be used for the DS interrupt as opposed to a trap gate to prevent the generation of an endless interrupt loop.

Pages that contain buffers must have mappings to the same physical address for all processes/logical processors, such that any change to CR3 will not change DS addresses. If this requirement cannot be satisfied (that is, the feature is enabled on a per thread/process basis), then the operating system must ensure that the feature is enabled/disabled appropriately in the context switch code.

20.6.4 Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

The performance monitoring capability of processors based on Intel NetBurst microarchitecture and supporting Intel Hyper-Threading Technology is similar to that described in Section 20.6.3. However, the capability is extended so that:

- Performance counters can be programmed to select events qualified by logical processor IDs.
- Performance monitoring interrupts can be directed to a specific logical processor within the physical processor.

The sections below describe performance counters, event qualification by logical processor ID, and special purpose bits in ESCRs/CCCRs. They also describe MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_TC_PRECISE_EVENT.

20.6.4.1 ESCR MSRs

Figure 20-51 shows the layout of an ESCR MSR in processors supporting Intel Hyper-Threading Technology.

The functions of the flags and fields are as follows:

- **T1_USR flag, bit 0** — When set, events are counted when thread 1 (logical processor 1) is executing at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.

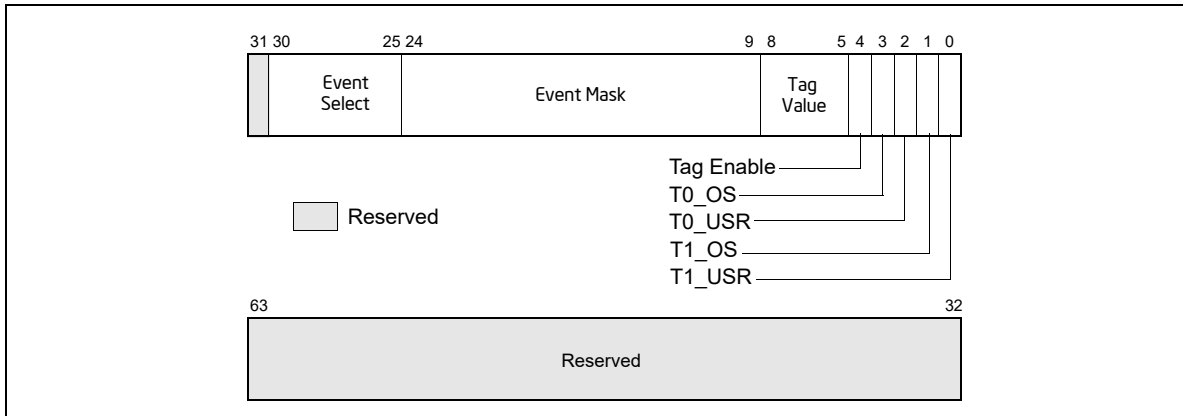


Figure 20-51. Event Selection Control Register (ESCR) for the Pentium 4 Processor, Intel® Xeon® Processor, and Intel® Xeon® Processor MP Supporting Hyper-Threading Technology

- **T1_OS flag, bit 1** — When set, events are counted when thread 1 (logical processor 1) is executing at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the T1_OS and T1_USR flags are set, thread 1 events are counted at all privilege levels.)
- **T0_USR flag, bit 2** — When set, events are counted when thread 0 (logical processor 0) is executing at a CPL of 1, 2, or 3.
- **T0_OS flag, bit 3** — When set, events are counted when thread 0 (logical processor 0) is executing at CPL of 0. (When both the T0_OS and T0_USR flags are set, thread 0 events are counted at all privilege levels.)
- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 20.6.3.6, “At-Retirement Counting.”
- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

The T0_OS and T0_USR flags and the T1_OS and T1_USR flags allow event counting and sampling to be specified for a specific logical processor (0 or 1) within an Intel Xeon processor MP (See also: Section 9.4.5, “Identifying Logical Processors in an MP System,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).

Not all performance monitoring events can be detected within an Intel Xeon processor MP on a per logical processor basis (see Section 20.6.4.4, “Performance Monitoring Events”). Some sub-events (specified by an event mask bits) are counted or sampled without regard to which logical processor is associated with the detected event.

20.6.4.2 CCCR MSRs

Figure 20-52 shows the layout of a CCCR MSR in processors supporting Intel Hyper-Threading Technology. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset
- **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
- **Active thread field, bits 16 and 17** — Enables counting depending on which logical processors are active (executing a thread). This field enables filtering of events based on the state (active or inactive) of the logical processors. The encodings of this field are as follows:
 - 00** — None. Count only when neither logical processor is active.

01 — Single. Count only when one logical processor is active (either 0 or 1).

10 — Both. Count only when both logical processors are active.

11 — Any. Count when either logical processor is active.

A halted logical processor or a logical processor in the “wait for SIPI” state is considered inactive.

- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.

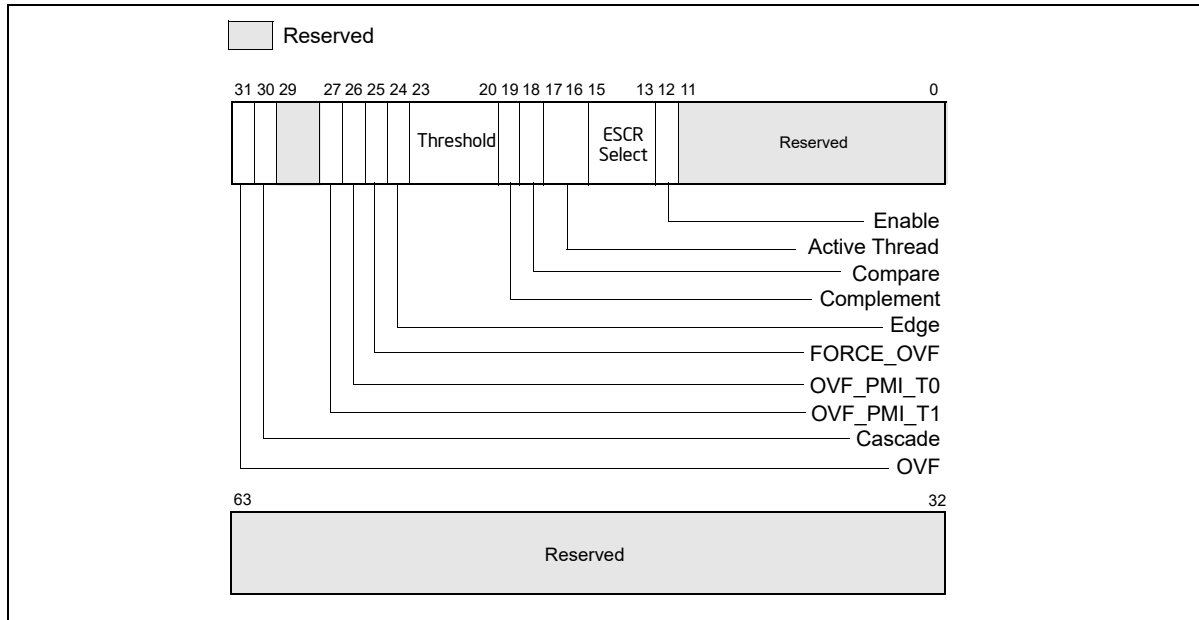


Figure 20-52. Counter Configuration Control Register (CCCR)

- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 20.6.3.5.2, “Filtering Events”). The compare flag is not active unless the compare flag is set.
- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 20.6.3.5.2, “Filtering Events”).
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.
- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI_T0 flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 0 when the counter overflows occurs; when clear, disables PMI generation for logical processor 0. Note that the PMI is generate on the next event count after the counter has overflowed.
- **OVF_PMI_T1 flag, bit 27** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 1 when the counter overflows occurs; when clear, disables PMI generation for logical processor 1. Note that the PMI is generate on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 20.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.

- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

20.6.4.3 IA32_PEBS_ENABLE MSR

In a processor supporting Intel Hyper-Threading Technology and based on the Intel NetBurst microarchitecture, PEBS is enabled and qualified with two bits in the MSR_PEBS_ENABLE MSR: bit 25 (ENABLE_PEBS_MY_THR) and 26 (ENABLE_PEBS_OTH_THR) respectively. These bits do not explicitly identify a specific logical processor by logic processor ID(T0 or T1); instead, they allow a software agent to enable PEBS for subsequent threads of execution on the same logical processor on which the agent is running ("my thread") or for the other logical processor in the physical package on which the agent is not running ("other thread").

PEBS is supported for only a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can be carried out only with two performance counters: MSR_IQ_CCCR4 (MSR address 370H) for logical processor 0 and MSR_IQ_CCCR5 (MSR address 371H) for logical processor 1.

Performance monitoring tools should use a processor affinity mask to bind the kernel mode components that need to modify the ENABLE_PEBS_MY_THR and ENABLE_PEBS_OTH_THR bits in the MSR_PEBS_ENABLE MSR to a specific logical processor. This is to prevent these kernel mode components from migrating between different logical processors due to OS scheduling.

20.6.4.4 Performance Monitoring Events

When Intel Hyper-Threading Technology is active, many performance monitoring events can be qualified by the logical processor ID, which corresponds to bit 0 of the initial APIC ID. This allows for counting an event in any or all of the logical processors. However, not all the events have this logic processor specificity, or thread specificity.

Here, each event falls into one of two categories:

- **Thread specific (TS)** — The event can be qualified as occurring on a specific logical processor.
- **Thread independent (TI)** — The event cannot be qualified as being associated with a specific logical processor.

If for example, a TS event occurred in logical processor T0, the counting of the event (as shown in Table 20-89) depends only on the setting of the T0_USR and T0_OS flags in the ESCR being used to set up the event counter. The T1_USR and T1_OS flags have no effect on the count.

Table 20-89. Effect of Logical Processor and CPL Qualification for Logical-Processor-Specific (TS) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while T1 in USR	Counts while T1 in OS or USR	Counts while T1 in OS
T0_OS/T0_USR = 01	Counts while T0 in USR	Counts while T0 in USR or T1 in USR	Counts while (a) T0 in USR or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 11	Counts while T0 in OS or USR	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in OS
T0_OS/T0_USR = 10	Counts T0 in OS	Counts T0 in OS or T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS

When a bit in the event mask field is TI, the effect of specifying bit-0-3 of the associated ESCR are described in Table 15-6. For events that are marked as TI, the effect of selectively specifying T0_USR, T0_OS, T1_USR, T1_OS bits is shown in Table 20-90.

Table 20-90. Effect of Logical Processor and CPL Qualification for Non-logical-Processor-specific (TI) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 01	Counts while (a) T0 in USR or (b) T1 in USR	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 11	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 0	Counts while (a) T0 in OS or (b) T1 in OS	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS

20.6.4.5 Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

20.6.4.5.1 Non-Halted Clockticks

Use the following procedure to program ESCRs and CCCRs to obtain non-halted clockticks on processors based on Intel NetBurst microarchitecture:

1. Select an ESCR for the global_power_events and specify the RUNNING sub-event mask and the desired T0_OS/T0_USR/T1_OS/T1_USR bits for the targeted processor.
2. Select an appropriate counter.
3. Enable counting in the CCCR for that counter by setting the enable bit.

20.6.4.5.2 Non-Sleep Clockticks

Performance monitoring counters can be configured to count clockticks whenever the performance monitoring hardware is not powered-down. To count Non-sleep Clockticks with a performance-monitoring counter, do the following:

1. Select one of the 18 counters.
2. Select any of the ESCRs whose events the selected counter can count. Set its event select to anything other than "no_event"; the counter may be disabled if this is not done.
3. Turn threshold comparison on in the CCCR by setting the compare bit to "1".
4. Set the threshold to "15" and the complement to "1" in the CCCR. Since no event can exceed this threshold, the threshold condition is met every cycle and the counter counts every cycle. Note that this overrides any qualification (e.g., by CPL) specified in the ESCR.
5. Enable counting in the CCCR for the counter by setting the enable bit.

In most cases, the counts produced by the non-halted and non-sleep metrics are equivalent if the physical package supports one logical processor and is not placed in a power-saving state. Operating systems may execute an HLT instruction and place a physical processor in a power-saving state.

On processors that support Intel Hyper-Threading Technology (Intel HT Technology), each physical package can support two or more logical processors. Current implementation of Intel HT Technology provides two logical processors for each physical processor. While both logical processors can execute two threads simultaneously, one logical processor may halt to allow the other logical processor to execute without sharing execution resources between two logical processors.

Non-halted Clockticks can be set up to count the number of processor clock cycles for each logical processor whenever the logical processor is not halted (the count may include some portion of the clock cycles for that logical processor to complete a transition to a halted state). Physical processors that support Intel HT Technology enter into a power-saving state if all logical processors halt.

The Non-sleep Clockticks mechanism uses a filtering mechanism in CCCRs. The mechanism will continue to increment as long as one logical processor is not halted or in a power-saving state. Applications may cause a processor to enter into a power-saving state by using an OS service that transfers control to an OS's idle loop. The idle loop then may place the processor into a power-saving state after an implementation-dependent period if there is no work for the processor.

20.6.5 Performance Monitoring and Dual-Core Technology

The performance monitoring capability of dual-core processors duplicates the microarchitectural resources of a single-core processor implementation. Each processor core has dedicated performance monitoring resources.

In the case of Pentium D processor, each logical processor is associated with dedicated resources for performance monitoring. In the case of Pentium processor Extreme edition, each processor core has dedicated resources, but two logical processors in the same core share performance monitoring resources (see Section 20.6.4, "Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture").

20.6.6 Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache

The 64-bit Intel Xeon processor MP with up to 8-MByte L3 cache has a CPUID signature of family [0FH], model [03H or 04H]. Performance monitoring capabilities available to Pentium 4 and Intel Xeon processors with the same values (see Section 20.1 and Section 20.6.4) apply to the 64-bit Intel Xeon processor MP with an L3 cache.

The level 3 cache is connected between the system bus and IOQ through additional control logic. See Figure 20-53.

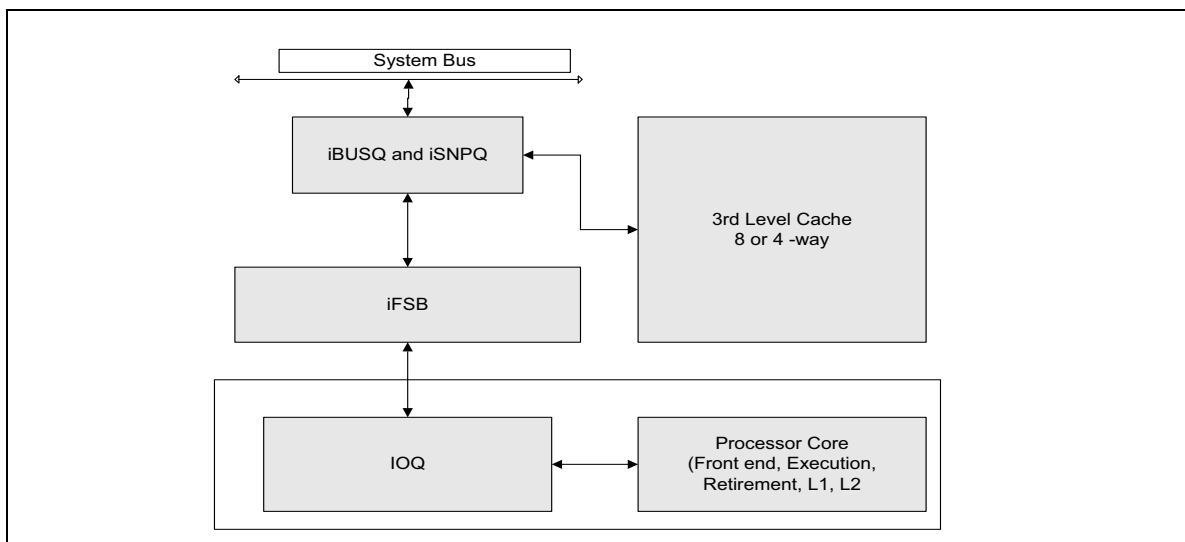


Figure 20-53. Block Diagram of 64-bit Intel® Xeon® Processor MP with 8-MByte L3

Additional performance monitoring capabilities and facilities unique to 64-bit Intel Xeon processor MP with an L3 cache are described in this section. The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs), each dedicated to a specific event. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values.

The lower 32-bits of the MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers. These performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

The performance monitoring capabilities consist of four events. These are:

- IBUSQ event** — This event detects the occurrence of micro-architectural conditions related to the iBUSQ unit. It provides two MSRs: MSR_IFSB_IBUSQ0 and MSR_IFSB_IBUSQ1. Configure sub-event qualification and enable/disable functions using the high 32 bits of these MSRs. The low 32 bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32 bits. See Figure 20-54.

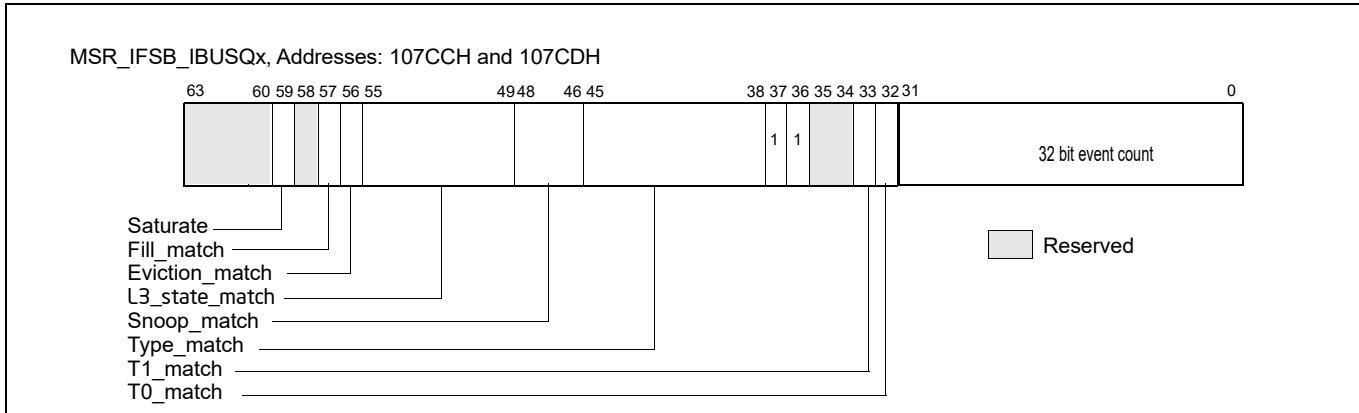


Figure 20-54. MSR_IFSB_IBUSQx, Addresses: 107CCH and 107CDH

- ISNPQ event** — This event detects the occurrence of microarchitectural conditions related to the iSNPQ unit. It provides two MSRs: MSR_IFSB_ISNPQ0 and MSR_IFSB_ISNPQ1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the MSRs. The low 32-bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32-bits. See Figure 20-55.

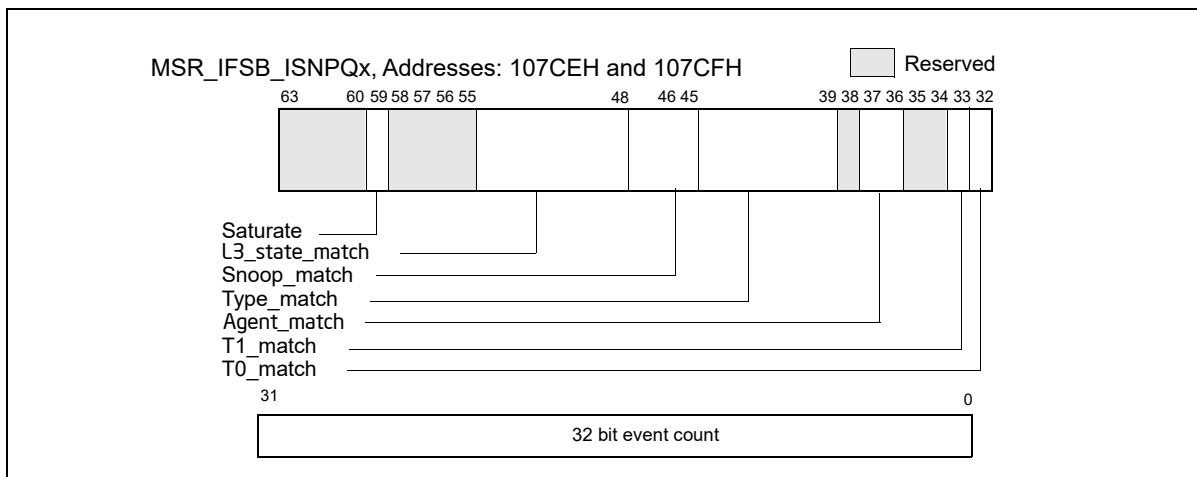


Figure 20-55. MSR_IFSB_ISNPQx, Addresses: 107CEH and 107CFH

- EFSB event** — This event can detect the occurrence of micro-architectural conditions related to the iFSB unit or system bus. It provides two MSRs: MSR_EFSB_DRDY0 and MSR_EFSB_DRDY1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the 64-bit MSR. The low 32-bit act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the qualification bits in the upper 32-bits of the MSR. See Figure 20-56.

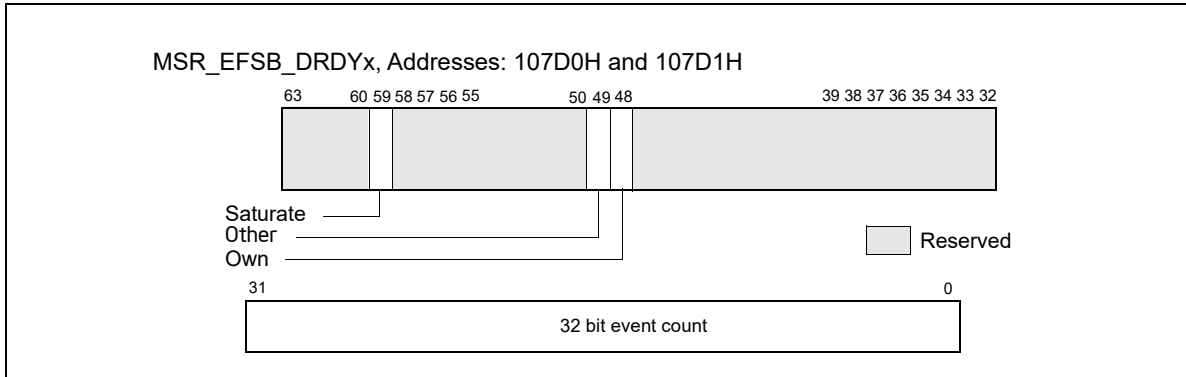


Figure 20-56. MSR_EFSB_DRDYx, Addresses: 107D0H and 107D1H

- IBUSQ Latency event** — This event accumulates weighted cycle counts for latency measurement of transactions in the iBUSQ unit. The count is enabled by setting MSR_IFSB_CTRL6[bit 26] to 1; the count freezes after software sets MSR_IFSB_CTRL6[bit 26] to 0. MSR_IFSB_CNTR7 acts as a 64-bit event counter for this event. See Figure 20-57.

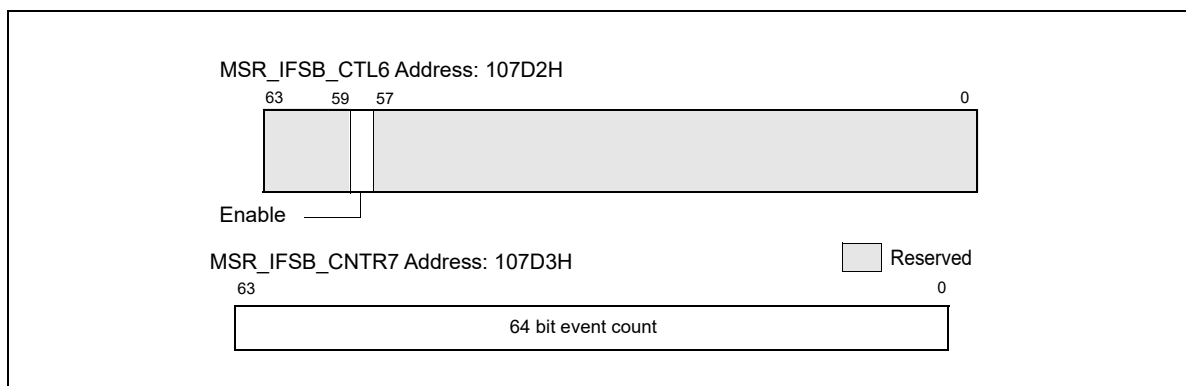


Figure 20-57. MSR_IFSB_CTL6, Address: 107D2H; MSR_IFSB_CNTR7, Address: 107D3H

20.6.7 Performance Monitoring on L3 and Caching Bus Controller Sub-Systems

The Intel Xeon processor 7400 series and Dual-Core Intel Xeon processor 7100 series employ a distinct L3/caching bus controller sub-system. These sub-system have a unique set of performance monitoring capability and programming interfaces that are largely common between these two processor families.

Intel Xeon processor 7400 series are based on 45 nm enhanced Intel Core microarchitecture. The CPUID signature is indicated by DisplayFamily_DisplayModel value of 06_1DH (see CPUID instruction in Chapter 3, “Instruction Set Reference, A-L” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). Intel Xeon processor 7400 series have six processor cores that share an L3 cache.

Dual-Core Intel Xeon processor 7100 series are based on Intel NetBurst microarchitecture, have a CPUID signature of family [0FH], model [06H] and a unified L3 cache shared between two cores. Each core in an Intel Xeon processor 7100 series supports Intel Hyper-Threading Technology, providing two logical processors per core.

Both Intel Xeon processor 7400 series and Intel Xeon processor 7100 series support multi-processor configurations using system bus interfaces. In Intel Xeon processor 7400 series, the L3/caching bus controller sub-system provides three Simple Direct Interface (SDI) to service transactions originated the XQ-replacement SDI logic in each dual-core modules. In Intel Xeon processor 7100 series, the IOQ logic in each processor core is replaced with a Simple Direct Interface (SDI) logic. The L3 cache is connected between the system bus and the SDI through additional control logic. See Figure 20-58 for the block configuration of six processor cores and the L3/Caching bus

controller sub-system in Intel Xeon processor 7400 series. Figure 20-58 shows the block configuration of two processor cores (four logical processors) and the L3/Caching bus controller sub-system in Intel Xeon processor 7100 series.

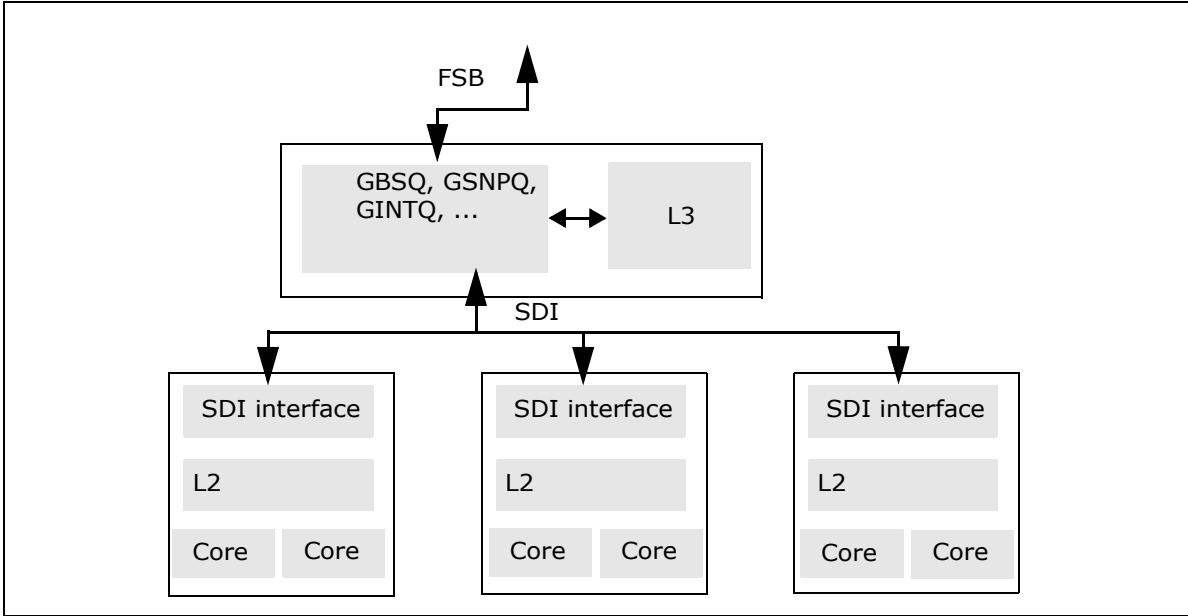


Figure 20-58. Block Diagram of the Intel® Xeon® Processor 7400 Series

Almost all of the performance monitoring capabilities available to processor cores with the same CPUID signatures (see Section 20.1 and Section 20.6.4) apply to Intel Xeon processor 7100 series. The MSR's used by performance monitoring interface are shared between two logical processors in the same processor core.

The performance monitoring capabilities available to processor with DisplayFamily_DisplayModel signature 06_17H also apply to Intel Xeon processor 7400 series. Each processor core provides its own set of MSR's for performance monitoring interface.

The IOQ_allocation and IOQ_active_entries events are not supported in Intel Xeon processor 7100 series and 7400 series. Additional performance monitoring capabilities applicable to the L3/caching bus controller sub-system are described in this section.

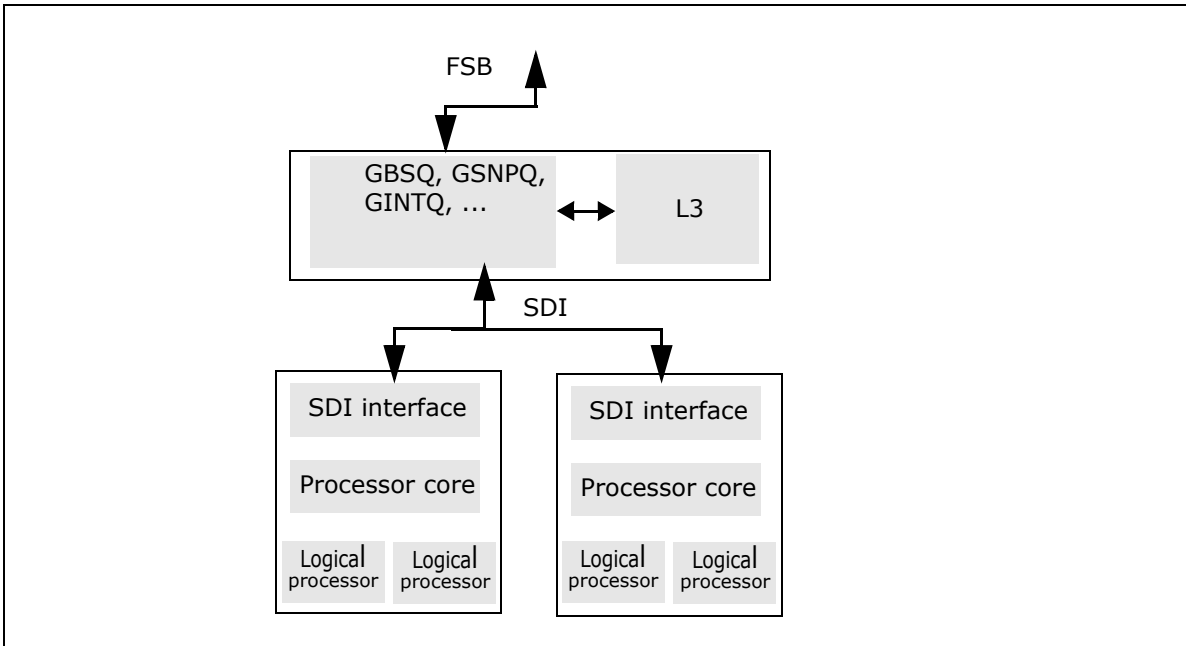


Figure 20-59. Block Diagram of the Intel® Xeon® Processor 7100 Series

20.6.7.1 Overview of Performance Monitoring with L3/Caching Bus Controller

The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs). There are eight event select/counting MSRs that are dedicated to counting events associated with specified microarchitectural conditions. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values. In addition, an MSR MSR_EMON_L3_GL_CTL provides simplified interface to control freezing, resetting, re-enabling operation of any combination of these event select/counting MSRs.

The eight MSRs dedicated to count occurrences of specific conditions are further divided to count three sub-classes of microarchitectural conditions:

- Two MSRs (MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1) are dedicated to counting GBSQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Two MSRs (MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3) are dedicated to counting GSNPQ events. Up to two GSNPQ events can be programmed and counted simultaneously.
- Four MSRs (MSR_EMON_L3_CTR_CTL4, MSR_EMON_L3_CTR_CTL5, MSR_EMON_L3_CTR_CTL6, and MSR_EMON_L3_CTR_CTL7) are dedicated to counting external bus operations.

The bit fields in each of eight MSRs share the following common characteristics:

- Bits 63:32 is the event control field that includes an event mask and other bit fields that control counter operation. The event mask field specifies details of the microarchitectural condition, and its definition differs across GBSQ, GSNPQ, FSB.
- Bits 31:0 is the event count field. If the specified condition is met during each relevant clock domain of the event logic, the matched condition signals the counter logic to increment the associated event count field. The lower 32-bits of these 8 MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers.

In Dual-Core Intel Xeon processor 7100 series, the uncore performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

In Intel Xeon processor 7400 series, RDPMC with ECX between 2 and 9 can be used to access the eight uncore performance counter/control registers.

20.6.7.2 GBSQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1 is given in Figure 20-60. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following eight attributes:

- Agent_Select (bits 35:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, each bit specifies a logical processor in the physical package. The lower two bits corresponds to two logical processors in the first processor core, the upper two bits corresponds to two logical processors in the second processor core. 0FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each bit of [34:32] specifies the SDI logic of a dual-core module as the originator of the transaction. A value of 0111B in bits [35:32] specifies transaction from any processor core.

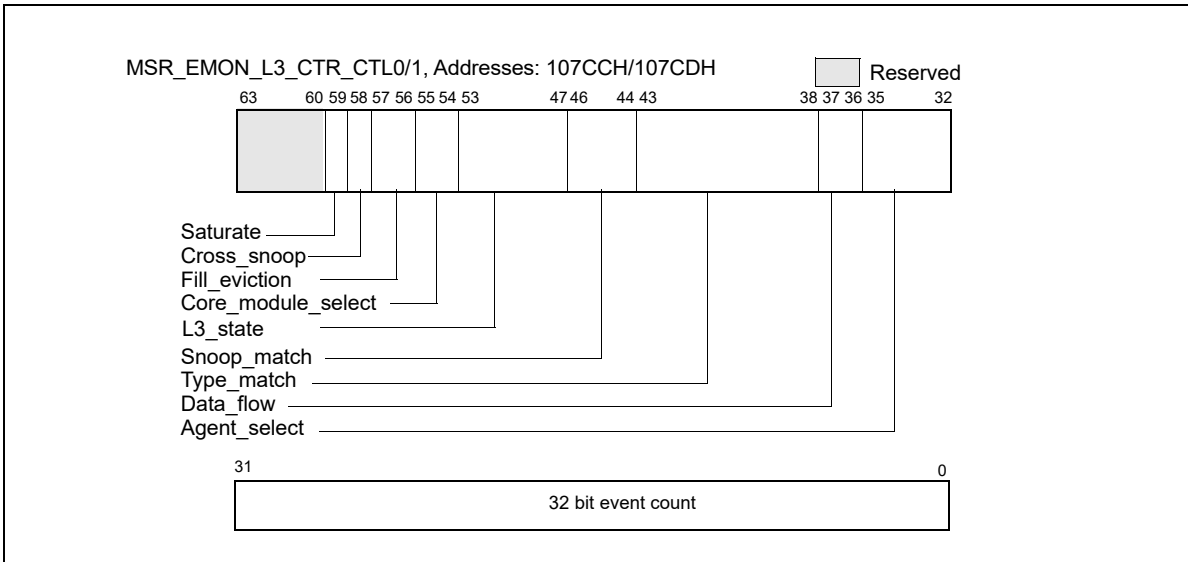


Figure 20-60. MSR_EMON_L3_CTR_CTL0/1, Addresses: 107CCH/107CDH

- Data_Flow (bits 37:36): Bit 36 specifies demand transactions, bit 37 specifies prefetch transactions.
- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include all transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L3_State (bits 53:47): Each bit specifies an L2 coherency state.
- Core_Module_Select (bits 55:54): The valid encodings for L3 lookup differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series,

- 00B: Match transactions from any core in the physical package
- 01B: Match transactions from this core only
- 10B: Match transactions from the other core in the physical package
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series,

- 00B: Match transactions from any dual-core module in the physical package
- 01B: Match transactions from this dual-core module only
- 10B: Match transactions from either one of the other two dual-core modules in the physical package

- 11B: Match transaction from more than one dual-core modules in the physical package
- Fill_Eviction (bits 57:56): The valid encodings are
 - 00B: Match any transactions
 - 01B: Match transactions that fill L3
 - 10B: Match transactions that fill L3 without an eviction
 - 11B: Match transaction fill L3 with an eviction
- Cross_Snoop (bit 58): The encodings are
 - 0B: Match any transactions
 - 1B: Match cross snoop transactions

For each counting clock domain, if all eight attributes match, event logic signals to increment the event count field.

20.6.7.3 GSNPQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3 is given in Figure 20-61. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following six attributes:

- Agent_Select (bits 37:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.
- For Intel Xeon processor 7100 series, each of the lowest 4 bits specifies a logical processor in the physical package. The lowest two bits corresponds to two logical processors in the first processor core, the next two bits corresponds to two logical processors in the second processor core. Bit 36 specifies other symmetric agent transactions. Bit 37 specifies central agent transactions. 3FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each of the lowest 3 bits specifies a dual-core module in the physical package. Bit 37 specifies central agent transactions.

- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include any transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L2_State (bits 53:47): Each bit specifies an L3 coherency state.
- Core_Module_Select (bits 56:54): Bit 56 enables Core_Module_Select matching. If bit 56 is clear, Core_Module_Select encoding is ignored. The valid encodings for the lower two bits (bit 55, 54) differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one core (irrespective which core) in the physical package
- 01B: Match transactions from this core and not the other core
- 10B: Match transactions from the other core in the physical package, but not this core
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one dual-core module (irrespective which module) in the physical package.
- 01B: Match transactions from one or more dual-core modules.
- 10B: Match transactions from two or more dual-core modules.
- 11B: Match transaction from all three dual-core modules in the physical package.

- Block_Snoop (bit 57): specifies blocked snoop.

For each counting clock domain, if all six attributes match, event logic signals to increment the event count field.

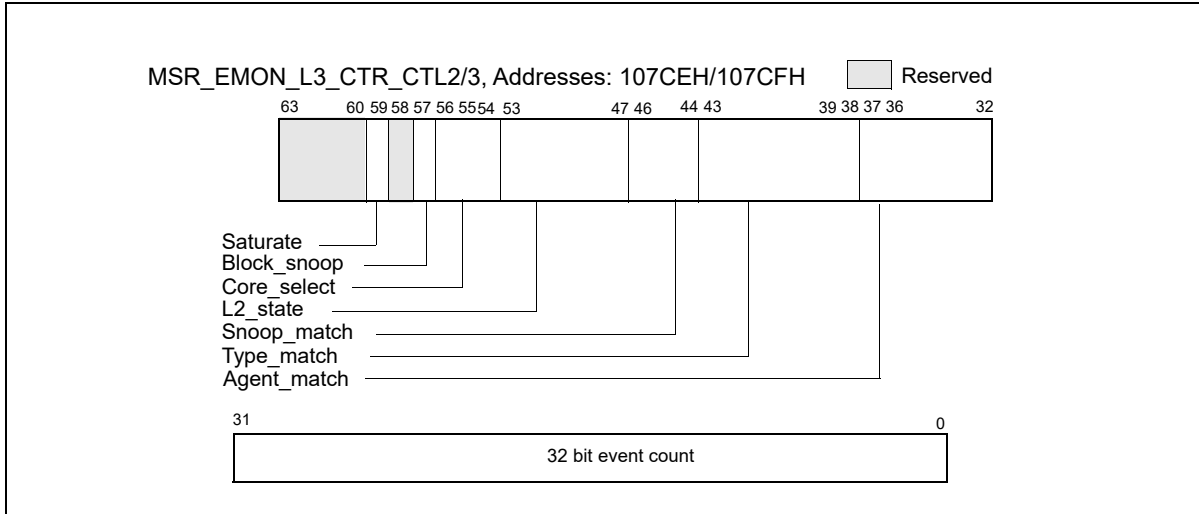


Figure 20-61. MSR_EMON_L3_CTR_CTL2/3, Addresses: 107CEH/107CFH

20.6.7.4 FSB Event Interface

The layout of MSR_EMON_L3_CTR_CTL4 through MSR_EMON_L3_CTR_CTL7 is given in Figure 20-62. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) is organized as follows:

- Bit 58: must set to 1.
- FSB_Submask (bits 57:32): Specifies FSB-specific sub-event mask.

The FSB sub-event mask defines a set of independent attributes. The event logic signals to increment the associated event count field if one of the attribute matches. Some of the sub-event mask bit counts durations. A duration event increments at most once per cycle.

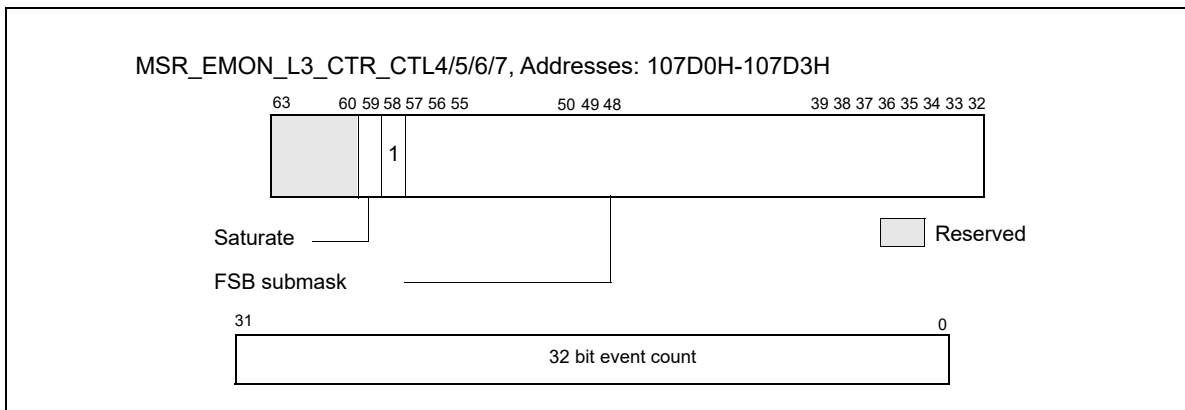


Figure 20-62. MSR_EMON_L3_CTR_CTL4/5/6/7, Addresses: 107D0H-107D3H

20.6.7.4.1 FSB Sub-Event Mask Interface

- FSB_type (bit 37:32): Specifies different FSB transaction types originated from this physical package.
- FSB_L_clear (bit 38): Count clean snoop results from any source for transaction originated from this physical package.
- FSB_L_hit (bit 39): Count HIT snoop results from any source for transaction originated from this physical package.

- FSB_L_hitm (bit 40): Count HITM snoop results from any source for transaction originated from this physical package.
- FSB_L_defer (bit 41): Count DEFER responses to this processor's transactions.
- FSB_L_retry (bit 42): Count RETRY responses to this processor's transactions.
- FSB_L_snoop_stall (bit 43): Count snoop stalls to this processor's transactions.
- FSB_DBSY (bit 44): Count DBSY assertions by this processor (without a concurrent DRDY).
- FSB_DRDY (bit 45): Count DRDY assertions by this processor.
- FSB_BNR (bit 46): Count BNR assertions by this processor.
- FSB_IOQ_empty (bit 47): Counts each bus clocks when the IOQ is empty.
- FSB_IOQ_full (bit 48): Counts each bus clocks when the IOQ is full.
- FSB_IOQ_active (bit 49): Counts each bus clocks when there is at least one entry in the IOQ.
- FSB_WW_data (bit 50): Counts back-to-back write transaction's data phase.
- FSB_WW_issue (bit 51): Counts back-to-back write transaction request pairs issued by this processor.
- FSB_WR_issue (bit 52): Counts back-to-back write-read transaction request pairs issued by this processor.
- FSB_RW_issue (bit 53): Counts back-to-back read-write transaction request pairs issued by this processor.
- FSB_other_DBSY (bit 54): Count DBSY assertions by another agent (without a concurrent DRDY).
- FSB_other_DRDY (bit 55): Count DRDY assertions by another agent.
- FSB_other_snoop_stall (bit 56): Count snoop stalls on the FSB due to another agent.
- FSB_other_BNR (bit 57): Count BNR assertions from another agent.

20.6.7.5 Common Event Control Interface

The MSR_EMON_L3_GL_CTL MSR provides simplified access to query overflow status of the GBSQ, GSNPQ, FSB event counters. It also provides control bit fields to freeze, unfreeze, or reset those counters. The following bit fields are supported:

- GL_freeze_cmd (bit 0): Freeze the event counters specified by the GL_event_select field.
- GL_unfreeze_cmd (bit 1): Unfreeze the event counters specified by the GL_event_select field.
- GL_reset_cmd (bit 2): Clear the event count field of the event counters specified by the GL_event_select field. The event select field is not affected.
- GL_event_select (bit 23:16): Selects one or more event counters to subject to specified command operations indicated by bits 2:0. Bit 16 corresponds to MSR_EMON_L3_CTR_CTL0, bit 23 corresponds to MSR_EMON_L3_CTR_CTL7.
- GL_event_status (bit 55:48): Indicates the overflow status of each event counters. Bit 48 corresponds to MSR_EMON_L3_CTR_CTL0, bit 55 corresponds to MSR_EMON_L3_CTR_CTL7.

In the event control field (bits 63:32) of each MSR, if the saturate control (bit 59, see Figure 20-60 for example) is set, the event logic forces the value FFFF_FFFFH into the event count field instead of incrementing it.

20.6.8 Performance Monitoring (P6 Family Processor)

The P6 family processors provide two 40-bit performance counters, allowing two types of events to be monitored simultaneously. These can either count events or measure duration. When counting events, a counter increments each time a specified event takes place or a specified number of events takes place. When measuring duration, it counts the number of processor clocks that occur while a specified condition is true. The counters can count events or measure durations that occur at any privilege level.

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The performance-monitoring counters are supported by four MSRs: the performance event select MSRs (PerfEvtSel0 and PerfEvtSel1) and the performance counter MSRs (PerfCtr0 and PerfCtr1). These registers can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0. The PerfCtr0 and PerfCtr1 MSRs can be read from any privilege level using the RDPMC (read performance-monitoring counters) instruction.

NOTE

The PerfEvtSel0, PerfEvtSel1, PerfCtr0, and PerfCtr1 MSRs and the events listed for P6 family processors are model-specific for P6 family processors. They are not guaranteed to be available in other IA-32 processors.

20.6.8.1 PerfEvtSel0 and PerfEvtSel1 MSRs

The PerfEvtSel0 and PerfEvtSel1 MSRs control the operation of the performance-monitoring counters, with one register used to set up each counter. They specify the events to be counted, how they should be counted, and the privilege levels at which counting should take place. Figure 20-63 shows the flags and fields in these MSRs.

The functions of the flags and fields in the PerfEvtSel0 and PerfEvtSel1 MSRs are as follows:

- **Event select field (bits 0 through 7)** — Selects the event logic unit to detect certain microarchitectural conditions.
- **Unit mask (UMASK) field (bits 8 through 15)** — Further qualifies the event logic unit selected in the event select field to detect a specific microarchitectural condition. For example, for some cache events, the mask is used as a MESI-protocol qualifier of cache states.

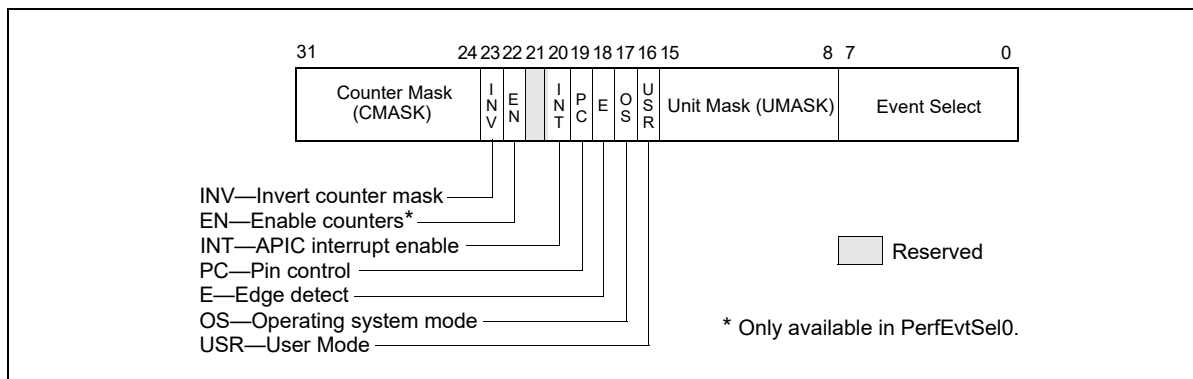


Figure 20-63. PerfEvtSel0 and PerfEvtSel1 MSRs

- **USR (user mode) flag (bit 16)** — Specifies that events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of events. The processor counts the number of deasserted to asserted transitions of any condition that can be expressed by the other fields. The mechanism is limited in that it does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

- **PC (pin control) flag (bit 19)** — When set, the processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.
- **INT (APIC interrupt enable) flag (bit 20)** — When set, the processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — This flag is only present in the PerfEvtSel0 MSR. When set, performance counting is enabled in both performance-monitoring counters; when clear, both counters are disabled.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When nonzero, the processor compares this mask to the number of events counted during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. This mask can be used to count events only if multiple occurrences happen per clock (for example, two or more instructions retired per clock). If the counter-mask field is 0, then the counter is incremented each cycle by the number of events that occurred that cycle.

20.6.8.2 PerfCtr0 and PerfCtr1 MSRs

The performance-counter MSRs (PerfCtr0 and PerfCtr1) contain the event or duration counts for the selected events being counted. The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

The WRMSR instruction cannot arbitrarily write to the performance-monitoring counter MSRs (PerfCtr0 and PerfCtr1). Instead, the lower-order 32 bits of each MSR may be written with any value, and the high-order 8 bits are sign-extended according to the value of bit 31. This operation allows writing both positive and negative values to the performance counters.

20.6.8.3 Starting and Stopping the Performance-Monitoring Counters

The performance-monitoring counters are started by writing valid setup information in the PerfEvtSel0 and/or PerfEvtSel1 MSRs and setting the enable counters flag in the PerfEvtSel0 MSR. If the setup is valid, the counters begin counting following the execution of a WRMSR instruction that sets the enable counter flag. The counters can be stopped by clearing the enable counters flag or by clearing all the bits in the PerfEvtSel0 and PerfEvtSel1 MSRs. Counter 1 alone can be stopped by clearing the PerfEvtSel1 MSR.

20.6.8.4 Event and Time-Stamp Monitoring Software

To use the performance-monitoring counters and time-stamp counter, the operating system needs to provide an event-monitoring device driver. This driver should include procedures for handling the following operations:

- Feature checking.
- Initialize and start counters.
- Stop counters.
- Read the event counters.
- Read the time-stamp counter.

The event monitor feature determination procedure must check whether the current processor supports the performance-monitoring counters and time-stamp counter. This procedure compares the family and model of the processor returned by the CPUID instruction with those of processors known to support performance monitoring. (The Pentium and P6 family processors support performance counters.) The procedure also checks the MSR and TSC flags returned to register EDX by the CPUID instruction to determine if the MSRs and the RDTSC instruction are supported.

The initialize and start counters procedure sets the PerfEvtSel0 and/or PerfEvtSel1 MSRs for the events to be counted and the method used to count them and initializes the counter MSRs (PerfCtr0 and PerfCtr1) to starting counts. The stop counters procedure stops the performance counters (see Section 20.6.8.3, “Starting and Stopping the Performance-Monitoring Counters”).

The read counters procedure reads the values in the PerfCtr0 and PerfCtr1 MSRs, and a read time-stamp counter procedure reads the time-stamp counter. These procedures would be provided in lieu of enabling the RDTSC and RDPMC instructions that allow application code to read the counters.

20.6.8.5 Monitoring Counter Overflow

The P6 family processors provide the option of generating a local APIC interrupt when a performance-monitoring counter overflows. This mechanism is enabled by setting the interrupt enable flag in either the PerfEvtSel0 or the PerfEvtSel1 MSR. The primary use of this option is for statistical performance sampling.

To use this option, the operating system should do the following things on the processor for which performance events are required to be monitored:

- Provide an interrupt vector for handling the counter-overflow interrupt.
- Initialize the APIC PERF local vector entry to enable handling of performance-monitor counter overflow events.
- Provide an entry in the IDT that points to a stub exception handler that returns without executing any instructions.
- Provide an event monitor driver that provides the actual interrupt handler and modifies the reserved IDT entry to point to its interrupt routine.

When interrupted by a counter overflow, the interrupt handler needs to perform the following actions:

- Save the instruction pointer (EIP register), code-segment selector, TSS segment selector, counter values and other relevant information at the time of the interrupt.
- Reset the counter to its initial setting and return from the interrupt.

An event monitor application utility or another application program can read the information collected for analysis of the performance of the profiled application.

20.6.9 Performance Monitoring (Pentium Processors)

The Pentium processor provides two 40-bit performance counters, which can be used to count events or measure duration. The counters are supported by three MSRs: the control and event select MSR (CESR) and the performance counter MSRs (CTR0 and CTR1). These can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0.

Each counter has an associated external pin (PM0/BP0 and PM1/BP1), which can be used to indicate the state of the counter to external hardware.

NOTES

The CESR, CTR0, and CTR1 MSRs and the events listed for Pentium processors are model-specific for the Pentium processor.

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

20.6.9.1 Control and Event Select Register (CESR)

The 32-bit control and event select MSR (CESR) controls the operation of performance-monitoring counters CTR0 and CTR1 and the associated pins (see Figure 20-64). To control each counter, the CESR register contains a 6-bit event select field (ES0 and ES1), a pin control flag (PC0 and PC1), and a 3-bit counter control field (CC0 and CC1). The functions of these fields are as follows:

- **ES0 and ES1 (event select) fields (bits 0-5, bits 16-21)** — Selects (by entering an event code in the field) up to two events to be monitored.

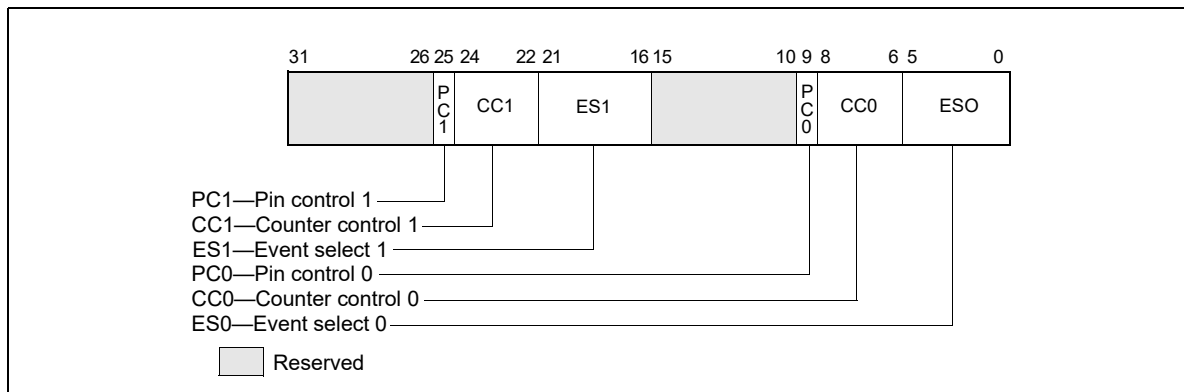


Figure 20-64. CESR MSR (Pentium Processor Only)

- **CC0 and CC1 (counter control) fields (bits 6-8, bits 22-24)** — Controls the operation of the counter. Control codes are as follows:

- 000 — Count nothing (counter disabled).
- 001 — Count the selected event while CPL is 0, 1, or 2.
- 010 — Count the selected event while CPL is 3.
- 011 — Count the selected event regardless of CPL.
- 100 — Count nothing (counter disabled).
- 101 — Count clocks (duration) while CPL is 0, 1, or 2.
- 110 — Count clocks (duration) while CPL is 3.
- 111 — Count clocks (duration) regardless of CPL.

The highest order bit selects between counting events and counting clocks (duration); the middle bit enables counting when the CPL is 3; and the low-order bit enables counting when the CPL is 0, 1, or 2.

- **PC0 and PC1 (pin control) flags (bits 9, 25)** — Selects the function of the external performance-monitoring counter pin (PM0/BP0 and PM1/BP1). Setting one of these flags to 1 causes the processor to assert its associated pin when the counter has overflowed; setting the flag to 0 causes the pin to be asserted when the counter has been incremented. These flags permit the pins to be individually programmed to indicate the overflow or incremented condition. The external signaling of the event on the pins will lag the internal event by a few clocks as the signals are latched and buffered.

While a counter need not be stopped to sample its contents, it must be stopped and cleared or preset before switching to a new event. It is not possible to set one counter separately. If only one event needs to be changed, the CESR register must be read, the appropriate bits modified, and all bits must then be written back to CESR. At reset, all bits in the CESR register are cleared.

20.6.9.2 Use of the Performance-Monitoring Pins

When performance-monitor pins PM0/BP0 and/or PM1/BP1 are configured to indicate when the performance-monitor counter has incremented and an "occurrence event" is being counted, the associated pin is asserted (high) each time the event occurs. When a "duration event" is being counted, the associated PM pin is asserted for the

entire duration of the event. When the performance-monitor pins are configured to indicate when the counter has overflowed, the associated PM pin is asserted when the counter has overflowed.

When the PM0/BP0 and/or PM1/BP1 pins are configured to signal that a counter has incremented, it should be noted that although the counters may increment by 1 or 2 in a single clock, the pins can only indicate that the event occurred. Moreover, since the internal clock frequency may be higher than the external clock frequency, a single external clock may correspond to multiple internal clocks.

A “count up to” function may be provided when the event pin is programmed to signal an overflow of the counter. Because the counters are 40 bits, a carry out of bit 39 indicates an overflow. A counter may be preset to a specific value less than $2^{40} - 1$. After the counter has been enabled and the prescribed number of events has transpired, the counter will overflow.

Approximately 5 clocks later, the overflow is indicated externally and appropriate action, such as signaling an interrupt, may then be taken.

The PM0/BP0 and PM1/BP1 pins also serve to indicate breakpoint matches during in-circuit emulation, during which time the counter increment or overflow function of these pins is not available. After RESET, the PM0/BP0 and PM1/BP1 pins are configured for performance monitoring, however a hardware debugger may reconfigure these pins to indicate breakpoint matches.

20.6.9.3 Events Counted

Events that performance-monitoring counters can be set to count and record (using CTR0 and CTR1) are divided in two categories: occurrence and duration:

- **Occurrence events** — Counts are incremented each time an event takes place. If PM0/BP0 or PM1/BP1 pins are used to indicate when a counter increments, the pins are asserted each clock counters increment. But if an event happens twice in one clock, the counter increments by 2 (the pins are asserted only once).
- **Duration events** — Counters increment the total number of clocks that the condition is true. When used to indicate when counters increment, PM0/BP0 and/or PM1/BP1 pins are asserted for the duration.

20.7 COUNTING CLOCKS

The count of cycles, also known as clockticks, forms the basis for measuring how long a program takes to execute. Clockticks are also used as part of efficiency ratios like cycles per instruction (CPI). Processor clocks may stop ticking under circumstances like the following:

- The processor is halted when there is nothing for the CPU to do. For example, the processor may halt to save power while the computer is servicing an I/O request. When Intel Hyper-Threading Technology is enabled, both logical processors must be halted for performance-monitoring counters to be powered down.
- The processor is asleep as a result of being halted or because of a power-management scheme. There are different levels of sleep. In the some deep sleep levels, the time-stamp counter stops counting.

In addition, processor core clocks may undergo transitions at different ratios relative to the processor’s bus clock frequency. Some of the situations that can cause processor core clock to undergo frequency transitions include:

- TM2 transitions.
- Enhanced Intel SpeedStep Technology transitions (P-state transitions).

For Intel processors that support TM2, the processor core clocks may operate at a frequency that differs from the Processor Base frequency (as indicated by processor frequency information reported by CPUID instruction). See Section 20.7.2 for more detail.

Due to the above considerations there are several important clocks referenced in this manual:

- **Base Clock** — The frequency of this clock is the frequency of the processor when the processor is not in turbo mode, and not being throttled via Intel SpeedStep.
- **Maximum Clock** — This is the maximum frequency of the processor when turbo mode is at the highest point.
- **Bus Clock** — These clockticks increment at a fixed frequency and help coordinate the bus on some systems.

- **Core Crystal Clock** — This is a clock that runs at fixed frequency; it coordinates the clocks on all packages across the system.
- **Non-halted Clockticks** — Measures clock cycles in which the specified logical processor is not halted and is not in any power-saving state. When Intel Hyper-Threading Technology is enabled, ticks can be measured on a per-logical-processor basis. There are also performance events on dual-core processors that measure clockticks per logical processor when the processor is not halted.
- **Non-sleep Clockticks** — Measures clock cycles in which the specified physical processor is not in a sleep mode or in a power-saving state. These ticks cannot be measured on a logical-processor basis.
- **Time-stamp Counter** — See Section 18.17, “Time-Stamp Counter”.
- **Reference Clockticks** — TM2 or Enhanced Intel SpeedStep technology are two examples of processor features that can cause processor core clockticks to represent non-uniform tick intervals due to change of bus ratios. Performance events that counts clockticks of a constant reference frequency was introduced Intel Core Duo and Intel Core Solo processors. The mechanism is further enhanced on processors based on Intel Core microarchitecture.

Some processor models permit clock cycles to be measured when the physical processor is not in deep sleep (by using the time-stamp counter and the RDTSC instruction). Note that such ticks cannot be measured on a per-logical-processor basis. See Section 18.17, “Time-Stamp Counter,” for detail on processor capabilities.

The first two methods use performance counters and can be set up to cause an interrupt upon overflow (for sampling). They may also be useful where it is easier for a tool to read a performance counter than to use a time stamp counter (the timestamp counter is accessed using the RDTSC instruction).

For applications with a significant amount of I/O, there are two ratios of interest:

- **Non-halted CPI** — Non-halted clockticks/instructions retired measures the CPI for phases where the CPU was being used. This ratio can be measured on a logical-processor basis when Intel Hyper-Threading Technology is enabled.
- **Nominal CPI** — Time-stamp counter ticks/instructions retired measures the CPI over the duration of a program, including those periods when the machine halts while waiting for I/O.

20.7.1 Non-Halted Reference Clockticks

Software can use UnHalted Reference Cycles on either a general purpose performance counter using event mask 0x3C and UMASK 0x01 or on fixed function performance counter 2 to count at a constant rate. These events count at a consistent rate irrespective of P-state, TM2, or frequency transitions that may occur to the processor. The UnHalted Reference Cycles event may count differently on the general purpose event and fixed counter.

20.7.2 Cycle Counting and Opportunistic Processor Operation

As a result of the state transitions due to opportunistic processor performance operation (see Chapter 15, “Power and Thermal Management”), a logical processor or a processor core can operate at frequency different from the Processor Base frequency.

The following items are expected to hold true irrespective of when opportunistic processor operation causes state transitions:

- The time stamp counter operates at a fixed-rate frequency of the processor.
- The IA32_MPERF counter increments at a fixed frequency irrespective of any transitions caused by opportunistic processor operation.
- The IA32_FIXED_CTR2 counter increments at the same TSC frequency irrespective of any transitions caused by opportunistic processor operation.
- The Local APIC timer operation is unaffected by opportunistic processor operation.
- The TSC, IA32_MPERF, and IA32_FIXED_CTR2 operate at close to the maximum non-turbo frequency, which is equal to the product of scalable bus frequency and maximum non-turbo ratio.

20.7.3 Determining the Processor Base Frequency

For Intel processors in which the nominal core crystal clock frequency is enumerated in CPUID.15H.ECX and the core crystal clock ratio is encoded in CPUID.15H (see Table 3-8 “Information Returned by CPUID Instruction”), the nominal TSC frequency can be determined by using the following equation:

$$\text{Nominal TSC frequency} = (\text{CPUID.15H.ECX}[31:0] * \text{CPUID.15H.EBX}[31:0]) \div \text{CPUID.15H.EAX}[31:0]$$

For Intel processors in which CPUID.15H.EBX[31:0] ÷ CPUID.0x15.EAX[31:0] is enumerated but CPUID.15H.ECX is not enumerated, Table 20-91 can be used to look up the nominal core crystal clock frequency.

Table 20-91. Nominal Core Crystal Clock Frequency

Processor Families/Processor Number Series ¹	Nominal Core Crystal Clock Frequency
Intel® Xeon® Scalable Processor Family with CPUID signature 06_55H.	25 MHz
6th and 7th generation Intel® Core™ processors and Intel® Xeon® W Processor Family.	24 MHz
Next Generation Intel Atom® processors based on Goldmont Microarchitecture with CPUID signature 06_5CH (does not include Intel Xeon processors).	19.2 MHz

NOTES:

1. For any processor in which CPUID.15H is enumerated and MSR_PLATFORM_INFO[15:8] (which gives the scalable bus frequency) is available, a more accurate frequency can be obtained by using CPUID.15H.

20.7.3.1 For Intel® Processors Based on Sandy Bridge, Ivy Bridge, Haswell, and Broadwell Microarchitectures

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 100 MHz.

20.7.3.2 For Intel® Processors Based on Nehalem Microarchitecture

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 133.33 MHz.

20.7.3.3 For Intel Atom® Processors Based on Silvermont Microarchitecture (Including Intel Processors Based on Airmont Microarchitecture)

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by the scalable bus frequency. The scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] for Intel Atom processors based on the Silvermont microarchitecture, and in bit field MSR_FSB_FREQ[3:0] for processors based on the Airmont microarchitecture; see Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

20.7.3.4 For Intel® Core™ 2 Processor Family and for Intel® Xeon® Processors Based on Intel Core Microarchitecture

For processors based on Intel Core microarchitecture, the scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] at (0CDH), see Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4. The maximum resolved bus ratio can be read from the following bit field:

- If XE operation is disabled, the maximum resolved bus ratio can be read in MSR_PLATFORM_ID[12:8]. It corresponds to the Processor Base frequency.

- If XE operation is enabled, the maximum resolved bus ratio is given in MSR_PERF_STATUS[44:40], it corresponds to the maximum XE operation frequency configured by BIOS.

XE operation of an Intel 64 processor is implementation specific. XE operation can be enabled only by BIOS. If MSR_PERF_STATUS[31] is set, XE operation is enabled. The MSR_PERF_STATUS[31] field is read-only.

20.8 IA32_PERF_CAPABILITIES MSR ENUMERATION

The layout of IA32_PERF_CAPABILITIES MSR is shown in Figure 20-65; it provides enumeration of a variety of interfaces:

- IA32_PERF_CAPABILITIES.LBR_FMT[bits 5:0]: encodes the LBR format, details are described in Section 18.4.8.1.
- IA32_PERF_CAPABILITIES.PEBSTrap[6]: Trap/Fault-like indicator of PEBS recording assist; see Section 20.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBSArchRegs[7]: Indicator of PEBS assist save architectural registers; see Section 20.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBS_FMT[bits 11:8]: Specifies the encoding of the layout of PEBS records; see Section 20.6.2.4.2.
- IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[12]: Indicates IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if 1, see Section 20.8.1.
- IA32_PERF_CAPABILITIES.FULL_WRITE[13]: Indicates the processor supports IA32_A_PMCx interface for updating bits 32 and above of IA32_PMCx; see Section 20.2.6.
- IA32_PERF_CAPABILITIES.PEBS_BASELINE [bit 14]: If set, the following is true:
 - The IA32_PEBS_ENABLE MSR (address 3F1H) exists and all architecturally enumerated fixed and general-purpose counters have corresponding bits in IA32_PEBS_ENABLE that enable generation of PEBS records. The general-purpose counter bits start at bit IA32_PEBS_ENABLE[0], and the fixed counter bits start at bit IA32_PEBS_ENABLE[32].
 - The format of the PEBS record is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT; see Section 20.6.2.4.2.
 - Extended PEBS is supported. All counters support the PEBS facility, and all events (both precise and non-precise) can generate PEBS records when PEBS is enabled for that counter. Note that not all events may be available on all counters.
 - Adaptive PEBS is supported. The PEBS_DATA_CFG MSR (address 3F2H) and adaptive record enable bits (IA32_PERFEVTSELx.Adaptive_Record and IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record) are supported. The definition of the PEBS_DATA_CFG MSR, including which bits are supported and how they affect the record, is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT; see Section 20.9.2.3.
 - NOTE: Software is recommended to feature PEBS Baseline when the following is true: IA32_PERF_CAPABILITIES.PEBS_BASELINE[14] && IA32_PERF_CAPABILITIES.PEBS_FMT[11:8] ≥ 4.
- IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE[15]: If set, indicates that the architecture provides built in support for TMA L1 metrics through the PERF_METRICS MSR, see Section 20.3.9.3.
- IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16]: If set on parts that enumerate support for Intel PT (CPUID.0x7.0.EBX[25]=1), setting IA32_PEBS_ENABLE.PEBS_OUTPUT to 01B will result in PEBS output being written into the Intel PT trace stream. See Section 20.5.5.2.

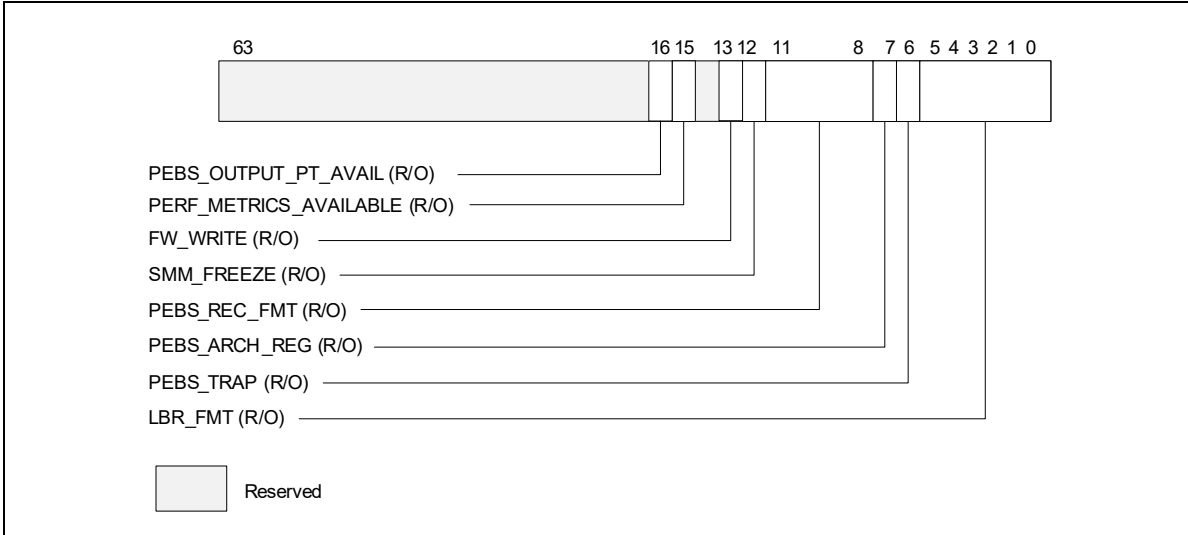


Figure 20-65. Layout of IA32_PERF_CAPABILITIES MSR

20.8.1 Filtering of SMM Handler Overhead

When performance monitoring facilities and/or branch profiling facilities (see Section 18.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)”) are enabled, these facilities capture event counts, branch records and branch trace messages occurring in a logical processor. The occurrence of interrupts, instruction streams due to various interrupt handlers all contribute to the results recorded by these facilities.

If CPUID.01H:ECX.PDCM[bit 15] is 1, the processor supports the IA32_PERF_CAPABILITIES MSR. If IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is 1, the processor supports the ability for system software using performance monitoring and/or branch profiling facilities to filter out the effects of servicing system management interrupts.

If the FREEZE_WHILE_SMM capability is enabled on a logical processor and after an SMI is delivered, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler.

The enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its servicing.

It is the responsibility of the SMM code to ensure the state of the performance monitoring and branch profiling facilities are preserved upon entry or until prior to exiting the SMM. If any of this state is modified due to actions by the SMM code, the SMM code is required to restore such state to the values present at entry to the SMM handler.

System software is allowed to set IA32_DEBUGCTL.FREEZE_WHILE_SMM[bit 14] to 1 only supported as indicated by IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] reporting 1.

20.9 PEBS FACILITY

20.9.1 Extended PEBS

- The Extended PEBS feature supports Processor Event Based Sampling (PEBS) on all counters, both fixed function and general purpose; and all performance monitoring events, both precise and non-precise. PEBS can be enabled for the general purpose counters using PEBS_EN_PMCi bits of IA32_PEBS_ENABLE (i = 0, 1,..m). PEBS can be enabled for 'i' fixed function counters using the PEBS_EN_FIXEDi bits of IA32_PEBS_ENABLE (i = 0, 1, ...n).

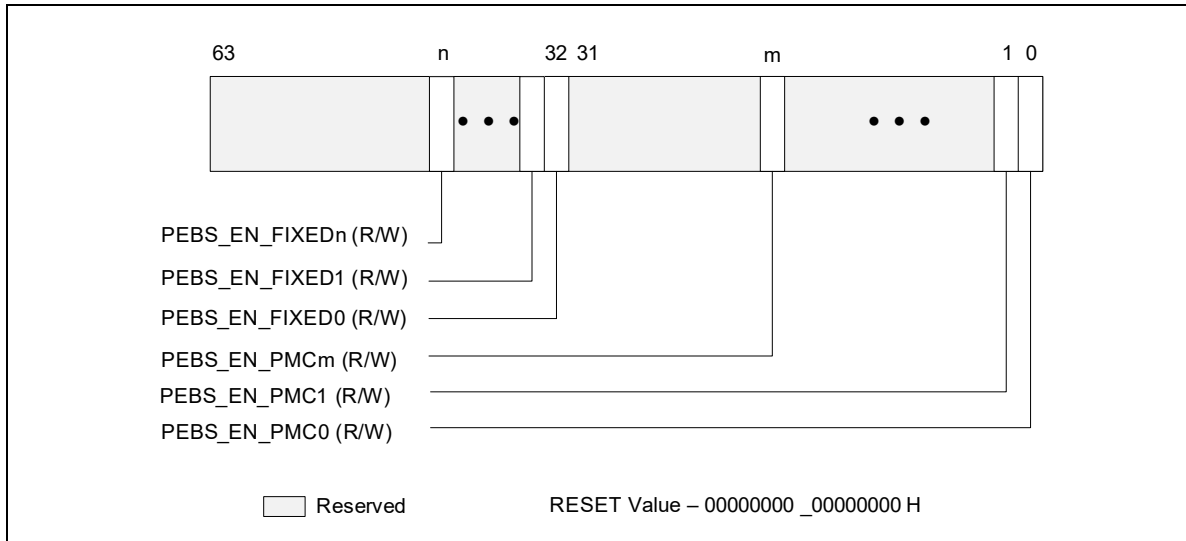


Figure 20-66. Layout of IA32_PEBS_ENABLE MSR

A PEBS record due to a precise event will be generated after an instruction that causes the event when the counter has already overflowed. A PEBS record due to a non-precise event will occur at the next opportunity after the counter has overflowed, including immediately after an overflow is set by an MSR write.

Currently, IA32_FIXED_CTR0 counts instructions retired and is a precise event. IA32_FIXED_CTR1, IA32_FIXED_CTR2 ... IA32_FIXED_CTR m count as non-precise events.

The Applicable Counter field in the Basic Info Group of the PEBS record indicates which counters caused the PEBS record to be generated. It is in the same format as the enable bits for each counter in IA32_PEBS_ENABLE. As an example, an Applicable Counter field with bits 2 and 32 set would indicate that both general purpose counter 2 and fixed function counter 0 generated the PEBS record.

- To properly use PEBS for the additional counters, software will need to set up the counter reset values in PEBS portion of the DS_BUFFER_MANAGEMENT_AREA data structure that is indicated by the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 20-67. When a counter generates a PEBS records, the appropriate counter reset values will be loaded into that counter. In the above example where general purpose counter 2 and fixed function counter 0 generated the PEBS record, general purpose counter 2 would be reloaded with the value contained in PEBS GP Counter 2 Reset (offset 50H) and fixed function counter 0 would be reloaded with the value contained in PEBS Fixed Counter 0 Reset (offset 80H).

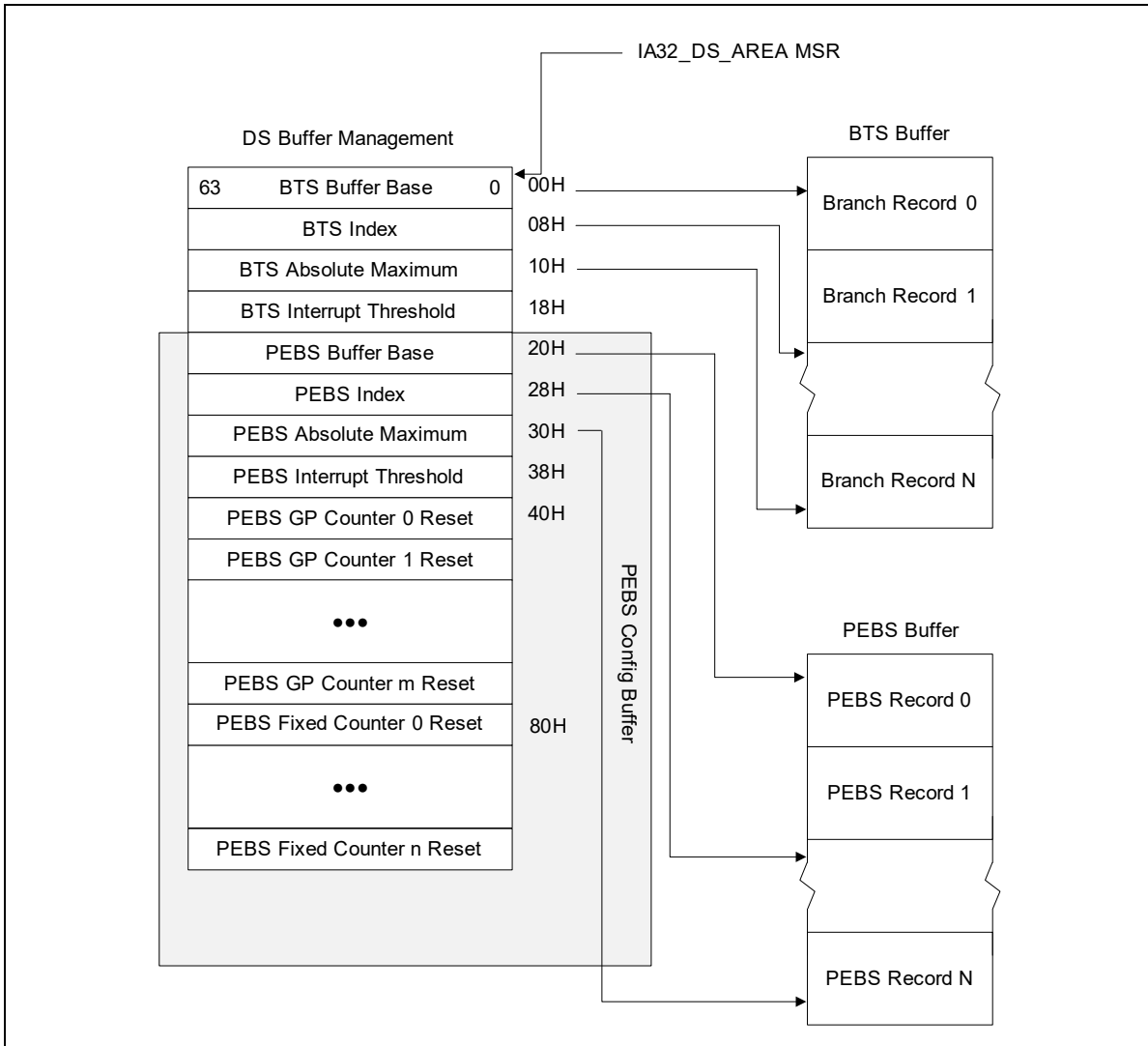


Figure 20-67. PEBS Programming Environment

Extended PEBS support debuts on Intel Atom[®] processors based on the Goldmont Plus microarchitecture and future Intel[®] Core[™] processors based on the Ice Lake microarchitecture.

20.9.2 Adaptive PEBS

The PEBS facility has been enhanced to collect the following CPU state in addition to GPRs, EventingIP, TSC, and memory access related information collected by legacy PEBS:

- XMM registers
- LBR records (TO/FROM/INFO)

The PEBS record is restructured where fields are grouped into Basic group, Memory group, GPR group, XMM group, and LBR group. A new register MSR_PEBS_DATA_CFG provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.

By default, the PEBS record will only contain the Basic group. Optionally, each counter can be configured to generate a PEBS records with the groups specified in MSR_PEBS_DATA_CFG.

Details and examples for the Adaptive PEBS capability follow below.

20.9.2.1 Adaptive_Record Counter Control

- IA32_PERFEVTSELx.Adaptive_Record[34]: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_PMCx is set for the corresponding GP counter, an overflow of PMCx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

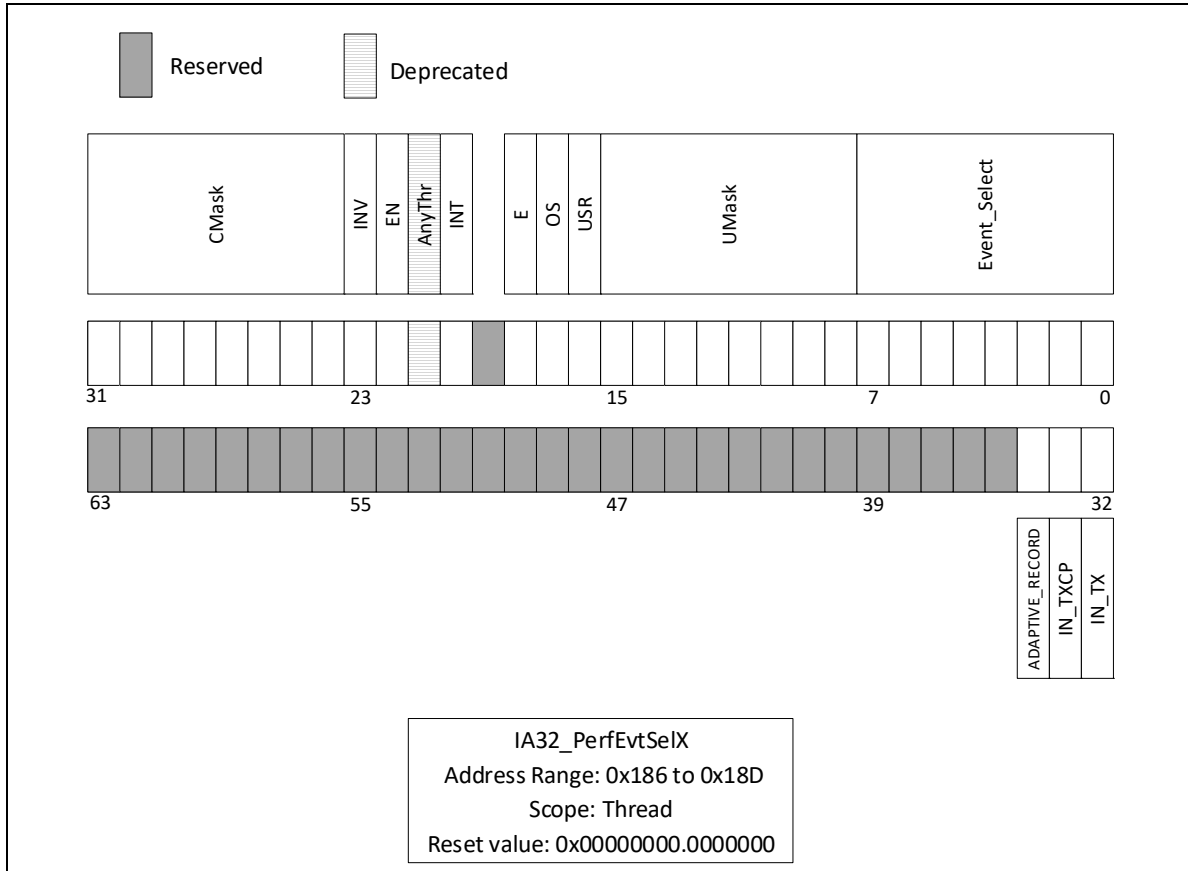


Figure 20-68. Layout of IA32_PerfEvtSelX MSR Supporting Adaptive PEBS

- IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_FIXEDx is set for the corresponding Fixed counter, an overflow of FixedCtrx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

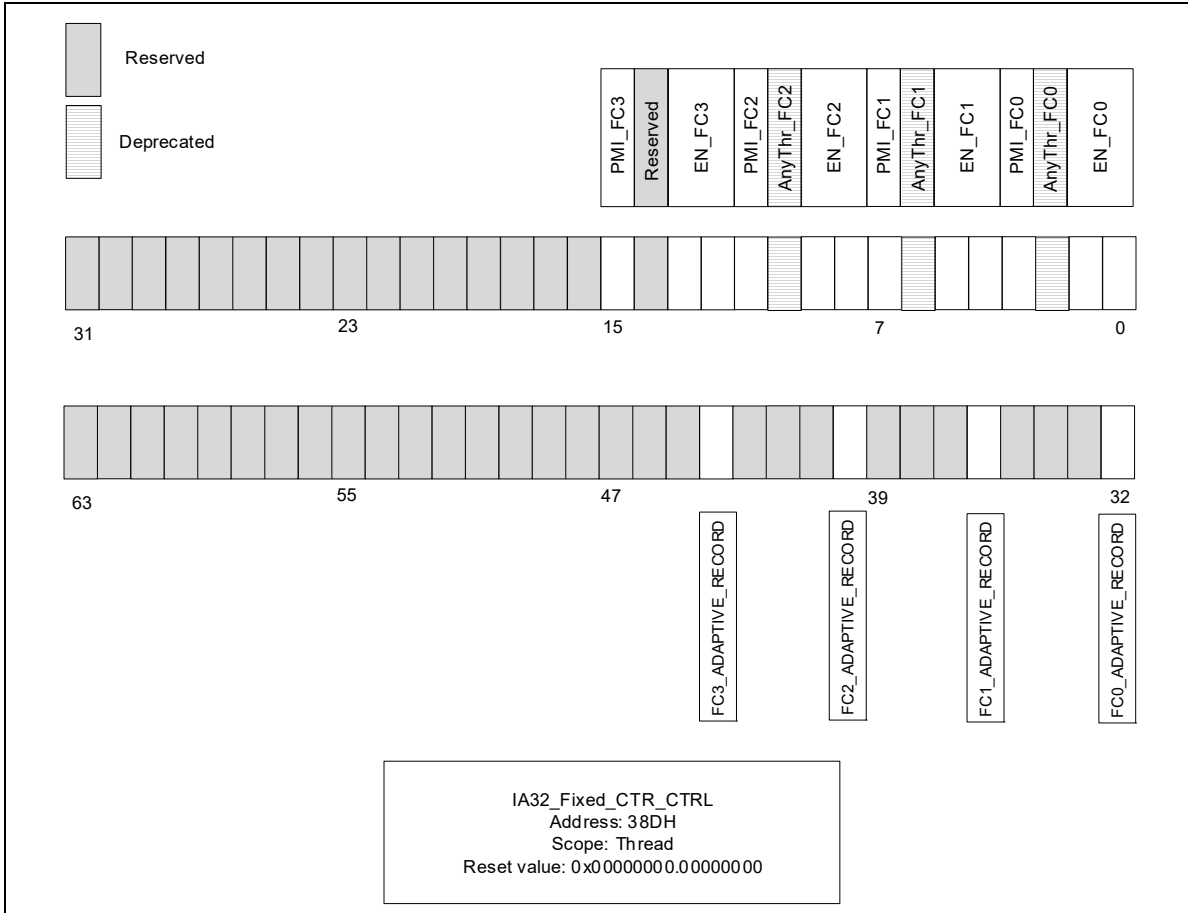


Figure 20-69. Layout of IA32_FIXED_CTR_CTRL MSR Supporting Adaptive PEBS

20.9.2.2 PEBS Record Format

The data fields in the PEBS record are aggregated into five groups which are described in the sub-sections below. Processors that support Adaptive PEBS implement a new MSR called MSR_PEBS_DATA_CFG which allows software to select the data groups to be captured. The data groups are not placed at fixed locations in the PEBS record, but are positioned immediately after one another, thus making the record format/size variable based on the groups selected.

20.9.2.2.1 Basic Info

The Basic group contains essential information for software to parse a record along with several critical fields. It is always collected.

Table 20-92. Basic Info Group

Field Name	Bit Width	Description
Record Format	[47:0]	This field indicates which data groups are included in the record. The field is zero if none of the counters that triggered the current PEBS record have their Adaptive_Record bit set. Otherwise it contains the value of MSR_PEBS_DATA_CFG.
	[63:48]	This field provides the size of the current record in bytes. Selected groups are packed back-to-back in the record without gaps or padding for unselected groups.

Table 20-92. Basic Info Group (Contd.)

Instruction Pointer	[63:0]	This field reports the Eventing Instruction Pointer (EventingIP) of the retired instruction that triggered the PEBS record generation. Note that this field is different than R/EIP which records the instruction pointer of the next instruction to be executed after record generation. The legacy R/EIP field has been removed.
Applicable Counters	[63:0]	The Applicable Counters field indicates which counters triggered the generation of the PEBS record, linking the record to specific events. This allows software to correlate the PEBS record entry properly with the instruction that caused the event, even when multiple counters are configured to generate PEBS records and multiple bits are set in the field.
TSC	[63:0]	This field provides the time stamp counter value when the PEBS record was generated.

20.9.2.2.2 Memory Access Info

This group contains the legacy PEBS memory-related fields; see Section 20.3.1.1.2.

Table 20-93. Memory Access Info Group

Field Name	Bit Width	Description
Memory Access Address	[63:0]	This field contains the linear address of the source of the load, or linear address of the destination (target) of the store. This value is written as a 64-bit address in canonical form.
Memory Auxiliary Info	[63:0]	When a MEM_TRANS_RETIRE.* event is configured in a General Purpose counter, this field contains an encoded value indicating the memory hierarchy source which satisfied the load. These encodings are detailed in Table 20-4 and Table 20-13. If the PEBS assist was triggered for a store uop, this field will contain information indicating the status of the store, as detailed in Table 20-14.
Memory Access Latency ¹	[63:0]	When a MEM_TRANS_RETIRE.* event is configured in a General Purpose counter, this field contains the latency to service the load in core clock cycles.
TSX Auxiliary Info	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	[63:32]	This field contains the abort details. Refer to Section 20.3.6.5.1.

NOTES:

1. In certain conditions, high latencies in fields under “Memory Access Latency” may be observed even when the Data Src of the “Memory Auxiliary Info” field indicates a close source.

Beginning with 12th generation Intel Core processors, the memory access information group has been updated. New fields added are shaded gray in Table 20-94.

Table 20-94. Updated Memory Access Info Group

Field Name	Sub-field Name	Bits	Description
Access Address (offset 0H)	DLA	[63:0]	This field reports the data linear address (DLA) of the memory access in canonical form. A zero value indicates the processor could not retrieve the address of the particular access.
Access Info (offset 8H)	Data Src	[3:0]	An encoded value indicating the memory hierarchy source which satisfied the access. These encodings are detailed in Table 20-4. A zero value indicates the processor could not retrieve the data source of the particular access.
	STLB-miss	[4]	A value of 1 indicates the access has missed the Second-level TLB (STLB).
	Is-Lock	[5]	A value of 1 indicates the access was part of a locked (atomic) memory transaction.
	Data-Blk	[6]	A value of 1 indicates the load was blocked since its data could not be forwarded from a preceding store.
	Address-Blk	[7]	A value of 1 indicates the load was blocked due to potential address conflict with a preceding store.
Access Latency (offset 10H)	Instruction Latency	[15:0]	Measured instruction latency in core cycles. For loads, the latency starts by the dispatch of the load operation for execution and lasts until completion of the instruction it belongs to. This field includes the entire latency including time for data-dependency resolution or TLB lookups.
	Cache Latency	[47:32]	Measured cache access latency in core cycles. For loads, the latency starts by the actual cache access until the data is returned by the memory subsystem. For stores, the latency starts when the demand write accesses the L1 data-cache and lasts until the cacheline write is completed in the memory subsystem. This field does not include non-data-cache latency such as memory ordering checks or TLB lookups.
TSX (offset 18H)	Transaction Latency	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	Abort Info	[63:32]	This field contains the abort details. Refer to Section 20.3.6.5.1.

To determine which fields are supported for certain performance monitoring events, consult the Memory Info attribute in the event lists at <https://download.01.org/perfmon/>.

NOTE

There may be additional block reasons, even if Data-Blk and Address-Blk are both clear, e.g., non-optimal instruction latency.

On P-core, the new Data-Blk and Address-Blk bits require the event LD_BLOCKS.STORE_FORWARD (r8203) to be configured in a programmable counter.

20.9.2.2.3 GPRs

This group is captured when the GPR bit is enabled in MSR_PEBS_DATA_CFG. GPRs are always 64 bits wide. If they are selected for non 64-bit mode, the upper 32-bit of the legacy RAX - RDI and all contents of R8-15 GPRs will be filled with 0s. In 64bit mode, the full 64 bit value of each register is written.

The order differs from legacy. The table below shows the order of the GPRs in Ice Lake microarchitecture.

Table 20-95. GPRs in Ice Lake Microarchitecture

Field Name	Bit Width
RFLAGS	[63:0]
RIP	[63:0]
RAX	[63:0]
RCX*	[63:0]
RDX*	[63:0]
RBX*	[63:0]
RSP*	[63:0]
RBP*	[63:0]
RSI*	[63:0]
RDI*	[63:0]
R8	[63:0]
...	...
R15	[63:0]

The machine state reported in the PEBS record is the committed machine state immediately after the instruction that triggers PEBS completes.

For instance, consider the following instruction sequence:

MOV eax, [eax]; triggers PEBS record generation

NOP

If the mov instruction triggers PEBS record generation, the EventingIP field in the PEBS record will report the address of the mov, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation. And the value of RIP will contain the linear address of the nop.

20.9.2.2.4 XMMs

This group is captured when the XMM bit is enabled in MSR_PEBS_DATA_CFG and SSE is enabled. If SSE is not enabled, the fields will contain zeroes. XMM8-XMM15 will also contain zeroes if not in 64-bit mode.

Table 20-96. XMMs

Field Name	Bit Width
XMM0	[127:0]
...	...
XMM15	[127:0]

20.9.2.2.5 LBRs

To capture LBR data in the PEBS record, the LBR bit in MSR_PEBS_DATA_CFG must be enabled. The number of LBR entries included in the record can be configured in the LBR_entries field of MSR_PEBS_DATA_CFG.

Table 20-97. LBRs

Field Name	Bit Width	Description
LBR[<i>i</i>].FROM	[63:0]	Branch from address.
LBR[<i>i</i>].TO	[63:0]	Branch to address.
LBR[<i>i</i>].INFO	[63:0]	Other LBR information, like timing. This field is described in more detail in Section 18.12.1, "MSR_LBR_INFO_x MSR".

LBR entries are recorded into the record starting at LBR[TOS] and proceeding to LBR[TOS-1] and following. Note that LBR index is modulo the number of LBRs supporting on the processor.

20.9.2.3 MSR_PEBS_DATA_CFG

Bits in MSR_PEBS_DATA_CFG can be set to include data field blocks/groups into adaptive records. The Basic Info group is always included in the record. Additionally, the number of LBR entries included in the record is configurable.

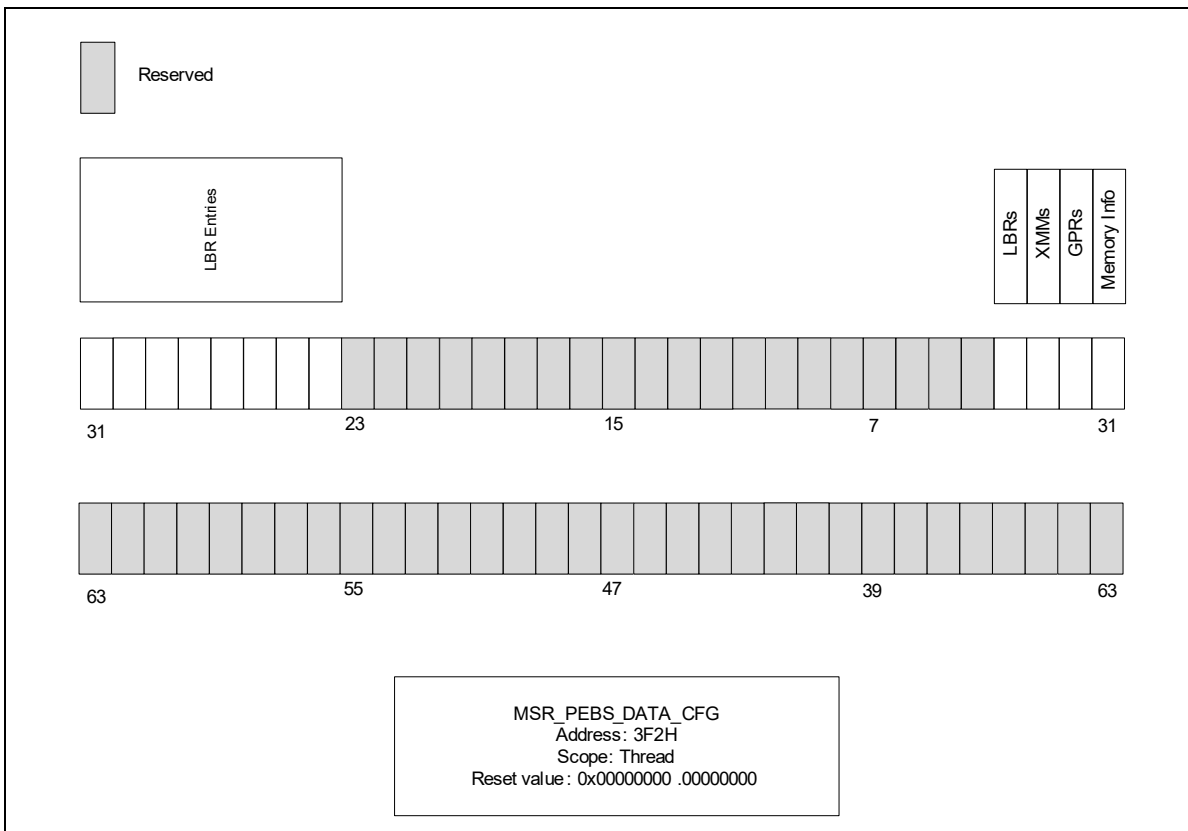


Figure 20-70. MSR_PEBS_DATA_CFG

Table 20-98. MSR_PEBS_CFG Programming¹

Bit	Bit Index	Access	Description
Memory Info	0	R/W	Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.
GPRs	1	R/W	Setting this bit will capture the contents of the General Purpose registers in the PEBS record.
XMMs	2	R/W	Setting this bit will capture the contents of the XMM registers in the PEBS record.
LBRs	3	R/W	Setting this bit will capture LBR TO, FROM, and INFO in the PEBS record.
Reserved ²	23:4	NA	Reserved
LBR Entries	31:24	R/W	Set the field to the desired number of entries minus 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, it is recommended to select an even number of LBR entries (programmed into LBR_entries as an odd number).

NOTES:

1. A write to the MSR will be ignored when IA32_MISC_ENABLE.PERFMON_AVAILABLE is zero (default).
2. Writing to the reserved bits will cause a GP fault.

20.9.2.4 PEBS Record Examples

The following example shows the layout of the PEBS record when all data groups are selected (all valid bits in MSR_PEBS_DATA_CFG are set) and maximum number of LBRs are selected. There are no gaps in the PEBS record when a subset of the groups are selected, thus keeping the layout compact. Implementations that do not support some features will have to pad zeroes in the corresponding fields.

Table 20-99. PEBS Record Example 1

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	
0x20	Memory Info	Memory Access Address	DLA
0x28		Memory Auxiliary Info	DATA_SRC
0x30		Memory Access Latency	Load Latency
0x38		TSX Auxiliary Info	HLE Information

Table 20-99. PEBS Record Example 1

0x40	GPRs	RFLAGS	
0x48		RIP	
0x50		RAX	
...		...	
0x88		RDI	
0x90		R8	
...		...	
0xC8		R15	
0xD0	XMMs	XMM0	New
...		...	
0x1C0		XMM15	
0x1D0	LBRs	LBR[TOS].FROM	New
0x1D8		LBR[TOS].TO	
0x1E0		LBR[TOS].INFO	
...		...	
0x4B8		LBR[TOS + 1].FROM	
0x4C0		LBR[TOS + 1].TO	
0x4C8		LBR[TOS + 1].INFO	

The following example shows the layout of the PEBS record when Basic, GPR, and LBR group with 3 LBR entries are selected.

Table 20-100. PEBS Record Example 2

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	

Table 20-100. PEBS Record Example 2

0x20	GPRs	RFLAGS	
0x28		RIP	
0x30		RAX	
...		...	
0x68		RDI	
0x70		R8	
...		...	
0xA8		R15	
0xB0	LBRs	LBR[TOS].FROM	New
0xB8		LBR[TOS].TO	
0xC0		LBR[TOS].INFO	
...		...	
0xE0		LBR[TOS + 1].FROM	
0xE8		LBR[TOS + 1].TO	
0xF0		LBR[TOS + 1].INFO	

20.9.3 Precise Distribution of Instructions Retired (PDIR) Facility

Precise Distribution of Instructions Retired Facility is available via PEBS on some microarchitectures. Refer to Section 20.3.4.4.4. Counters that support PDIR also vary. See the processor specific sections for availability.

20.9.4 Reduced Skid PEBS

For precise events, upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. The Reduced Skid mechanism mitigates the “skid” problem by providing an early indication of when the counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow thus greatly reducing skid.

This mechanism is a superset of the PDIR mechanism available in the Sandy Bridge microarchitecture. See Section 20.3.4.4.4

In the Goldmont microarchitecture, the mechanism applies to all precise events including, INST_RETIRE, except for UOPS_RETIRE. However, the Reduced Skid mechanism is disabled for any counter when the INV, ANY, E, or CMASK fields are set.

With Reduced Skid PEBS, the skid is precisely one event occurrence. Hence if counting INST_RETIRE, PEBS will indicate the instruction that follows that which caused the counter to overflow.

For the Reduced Skid mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via IA32_PERF_GLOBAL_CTRL MSR) before changes to the configuration (e.g., what event is specified in IA32_PERFVTSELx or whether PEBS is enabled for that counter via IA32_PEBS_ENABLE) or counter value (MSR write to IA32_PMCx and IA32_A_PMCx).

20.9.5 EPT-Friendly PEBS

The 3rd generation Intel Xeon Scalable Family of processors based on Ice Lake microarchitecture (and later processors) and the 12th generation Intel Core processor (and later processors) support VMX guest use of PEBS when the DS Area (including the PEBS Buffer and DS Management Area) is allocated from a paged pool of EPT pages. In such a configuration PEBS DS Area accesses may result in VM exits (e.g., EPT violations due to “lazy” EPT page-table entry propagation), and in such cases the PEBS record will not be lost but instead will “skid” to after the subsequent VM Entry back to the guest. For precise events the guest will observe that the record skid by one event occurrence, while for non-precise events the record will skid by one instruction.

20.9.6 PDist: Precise Distribution

PDist eliminates any skid or shadowing effects from PEBS. With PDist, the PEBS record will be generated precisely upon completion of the instruction or operation that causes the counter to overflow (there is no “wait for next occurrence” by default).

PDist is supported by selected counters, and is only supported when those counters are programmed to count select precise events¹. The legacy PEBS behavior applies to counters that do not support PDist, unless specified otherwise. PDist requires that the INV, ANY, E, and CMASK fields are cleared. Which counters support PDist, and which events are supported for PDist, is model-specific. Further, the counter reload value must not be lesser than 127 for PDist to operate.

For the PDist mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via IA32_PERF_GLOBAL_CTRL MSR) before changes to the configuration (e.g., what event is specified in IA32_PERFEVTSELx or whether PEBS is enabled for that counter via IA32_PEBS_ENABLE) or counter value (MSR write to IA32_PMCx and IA32_A_PMCx).

20.9.7 Load Latency Facility

The load latency facility provides software a means to characterize the latencies of memory load operations to different levels of cache/memory hierarchy. This facility requires a processor supporting the enhanced PEBS record format in the PEBS buffer.

Beginning with 12th generation Intel Core processors, the load latency facility supports all fields in Table 20-94, “Updated Memory Access Info Group”, in addition to the Memory Access Address field:

- The **Instruction Latency** field measures the load latency from the load's first dispatch until final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches and data dependencies).
- The **Cache Latency** field measures the subset of cache access latency in core cycles. It starts from the actual cache access until the data is returned by the memory subsystem. The latency is reported for retired demand load operations in core cycles (it does not account for memory ordering blocks).
- The **Data Source** field is an encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 20-101. In the descriptions, local memory refers to system memory physically attached to a processor package, and remote memory refers to system memory or cache physically attached to another processor package (in a server product).
- Through the **Access Info** field, load latency features binary indications on certain blocks that the load operation may have encountered. Refer to STLB-miss, Is-Lock, Data-Blk and Address-Blk fields in Table 20-94.

NOTE

For loads triggered by software prefetch instructions, the cache related fields including Data Source and Cache Latency, report values as if the load was an L1 cache hit (the prefetch completes without waiting for data return, for performance reasons).

1. To determine whether an event is precise or supports PDist, consult the relevant attribute in the event lists at <https://download.01.org/perfmon/>.

Table 20-101. Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)

Encoding	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request).
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. This request was merged into an outstanding cache miss to same cache-line address.
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. This request was satisfied by the L3 cache with no coherency actions performed (snooping).
05H	XCORE MISS. This request was satisfied by the L3 cache but involved a coherency check in some sibling core(s).
06H	XCORE HIT. This request was satisfied by the L3 cache but involved a coherency check that hit a non-modified copy in a sibling core.
07H	XCORE FWD. This request was satisfied by a sibling core where either a modified (cross-core HITM) or a non-modified (cross-core FWD) cache-line copy was found.
08H	Local Far Memory. This request has missed the L3 cache and was serviced by local far memory.
09H	Remote Far Memory. This request has missed the L3 cache and was serviced by remote far memory.
0AH	Local Near Memory. This request has missed the L3 cache and was serviced by local near memory.
0BH	Remote Near Memory. This request has missed the L3 cache and was serviced by remote near memory.
0CH	Remote FWD. This request has missed the L3 cache and a non-modified cache-line copy was forwarded from a remote cache.
0DH	Remote HITM. This request has missed the L3 cache and a modified cache-line was forwarded from a remote cache.
0EH	I/O. Request of input/output operation.
0FH	UC. The request was to uncacheable memory.

To use this feature, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- One of the relevant IA32_PERFEVTSELx MSRs is programmed to specify the event unit MEM_TRANS_RETIRED_LOAD_LATENCY (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter, IA32_PMCx, will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in an MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with instruction latency greater than this value are eligible for counting and PEBS data reporting. The minimum value that may be programmed in this register is 1.
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register.

Refer to Section 20.3.4.4.2 for further implementation details of Load Latency.

20.9.8 Store Latency Facility

Store latency support is available on the 12th generation Intel Core processor. Store latency is a PEBS extension that provides a means to profile store memory accesses in the system. It complements the load latency facility.

Store latency leverages the PEBS facility where it can provide additional information about sampled stores. The additional information includes the data address, memory auxiliary information, and the cache latency of the store access. Normal stores (those preceded with a read-for-ownership) as well as streaming stores are supported by the store latency facility.

Memory store operations typically do not limit performance since they update the memory with no operation that directly depends on them. Thus, data out of this facility should be carefully used once stores are suspected as a performance limiter; for example, once the TMA node of Backend_Bound.Memory_Bound.Store_Bound is flagged¹.

To enable the store latency facility, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- Program the MEM_TRANS_RETIRED.STORE_SAMPLE event on general-purpose performance-monitoring counter 0 (IA32_PERFVTSELO[15:0] = 2CDH).
- Setup the PEBS buffer to hold at least two records, setting both 'PEBS Absolute Maximum' and 'PEBS Interrupt Threshold', should any other counter be used by PEBS (that is whenever IA32_PEBS_ENABLE[x] ≠ 0 for x ≠ 0).
- Set IA32_PEBS_ENABLE[0].

The store latency information is written into a PEBS record as shown in Table 20-48.

The store latency relies on the PEBS facility, so the PEBS configuration must be completed first. Unlike load latency, there is no option to filter on a subset of stores that exceed a certain threshold.

1. For more details about the method, refer to Section B.1, "Top-Down Analysis Method" of the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

12. Updates to Chapter 25, Volume 3C

Change bars and green text show changes to Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Updates throughout the chapter for bus-lock detection and instruction timeout features.

25.1 OVERVIEW

A logical processor uses **virtual-machine control data structures (VMCSs)** while it is in VMX operation. These manage transitions into and out of VMX non-root operation (VM entries and VM exits) as well as processor behavior in VMX non-root operation. This structure is manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

A VMM can use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM can use a different VMCS for each virtual processor.

A logical processor associates a region in memory with each VMCS. This region is called the **VMCS region**.¹ Software references a specific VMCS using the 64-bit physical address of the region (a **VMCS pointer**). VMCS pointers must be aligned on a 4-KByte boundary (bits 11:0 must be zero). These pointers must not set bits beyond the processor's physical-address width.^{2,3}

A logical processor may maintain a number of VMCSs that are **active**. The processor may optimize VMX operation by maintaining the state of an active VMCS in memory, on the processor, or both. At any given time, at most one of the active VMCSs is the **current** VMCS. (This document frequently uses the term "the VMCS" to refer to the current VMCS.) The VMLAUNCH, VMREAD, VMRESUME, and VMWRITE instructions operate only on the current VMCS.

The following items describe how a logical processor determines which VMCSs are active and which is current:

- The memory operand of the VMPTRLD instruction is the address of a VMCS. After execution of the instruction, that VMCS is both active and current on the logical processor. Any other VMCS that had been active remains so, but no other VMCS is current.
- The VMCS link pointer field in the current VMCS (see Section 25.4.2) is itself the address of a VMCS. If VM entry is performed successfully with the 1-setting of the "VMCS shadowing" VM-execution control, the VMCS referenced by the VMCS link pointer field becomes active on the logical processor. The identity of the current VMCS does not change.
- The memory operand of the VMCLEAR instruction is also the address of a VMCS. After execution of the instruction, that VMCS is neither active nor current on the logical processor. If the VMCS had been current on the logical processor, the logical processor no longer has a current VMCS.

The VMPTRST instruction stores the address of the logical processor's current VMCS into a specified memory location (it stores the value FFFFFFFF_FFFFFFFFH if there is no current VMCS).

The **launch state** of a VMCS determines which VM-entry instruction should be used with that VMCS: the VMLAUNCH instruction requires a VMCS whose launch state is "clear"; the VMRESUME instruction requires a VMCS whose launch state is "launched". A logical processor maintains a VMCS's launch state in the corresponding VMCS region. The following items describe how a logical processor manages the launch state of a VMCS:

- If the launch state of the current VMCS is "clear", successful execution of the VMLAUNCH instruction changes the launch state to "launched".
- The memory operand of the VMCLEAR instruction is the address of a VMCS. After execution of the instruction, the launch state of that VMCS is "clear".
- There are no other ways to modify the launch state of a VMCS (it cannot be modified using VMWRITE) and there is no direct way to discover it (it cannot be read using VMREAD).

1. The amount of memory required for a VMCS region is at most 4 KBytes. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

3. If IA32_VMX_BASIC[48] is read as 1, these pointers must not set any bits in the range 63:32; see Appendix A.1.

Figure 25-1 illustrates the different states of a VMCS. It uses "X" to refer to the VMCS and "Y" to refer to any other VMCS. Thus: "VMPTRLD X" always makes X current and active; "VMPTRLD Y" always makes X not current (because it makes Y current); VMLAUNCH makes the launch state of X "launched" if X was current and its launch state was "clear"; and VMCLEAR X always makes X inactive and not current and makes its launch state "clear".

The figure does not illustrate operations that do not modify the VMCS state relative to these parameters (e.g., execution of VMPTRLD X when X is already current). Note that VMCLEAR X makes X "inactive, not current, and clear," even if X's current state is not defined (e.g., even if X has not yet been initialized). See Section 25.11.3.

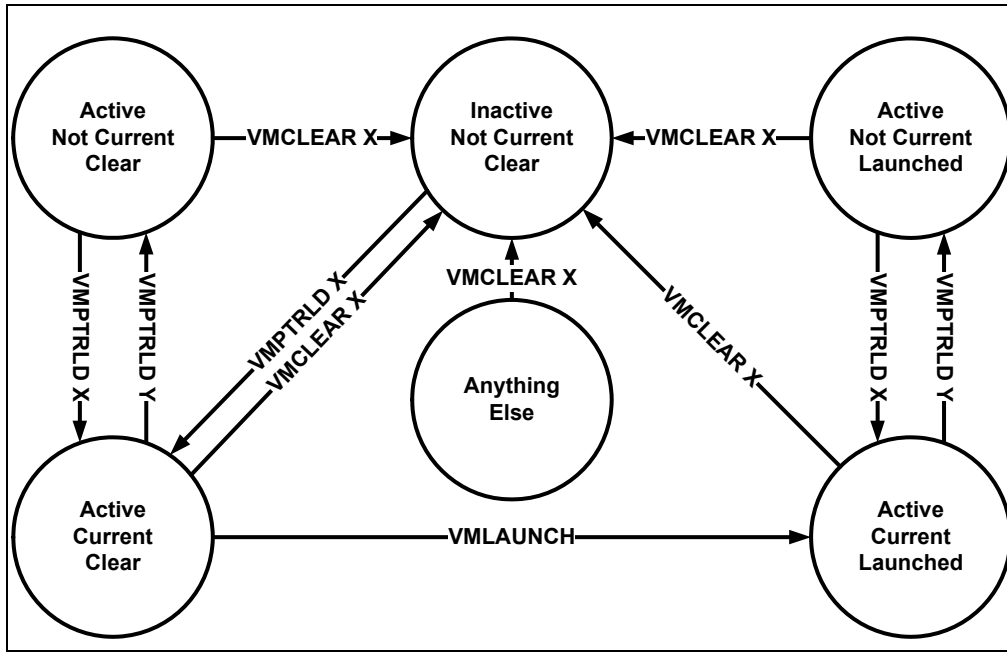


Figure 25-1. States of VMCS X

Because a shadow VMCS (see Section 25.10) cannot be used for VM entry, the launch state of a shadow VMCS is not meaningful. Figure 25-1 does not illustrate all the ways in which a shadow VMCS may be made active.

25.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.¹ The format of a VMCS region is given in Table 25-1.

Table 25-1. Format of the VMCS Region

Byte Offset	Contents
0	Bits 30:0: VMCS revision identifier Bit 31: shadow-VMCS indicator (see Section 25.10)
4	VMX-abort indicator
8	VMCS data (implementation-specific format)

1. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

The first 4 bytes of the VMCS region contain the **VMCS revision identifier** at bits 30:0.¹ Processors that maintain VMCS data in different formats (see below) use different VMCS revision identifiers. These identifiers enable software to avoid using a VMCS region formatted for one processor on a processor that uses a different format.² Bit 31 of this 4-byte region indicates whether the VMCS is a shadow VMCS (see Section 25.10).

Software should write the VMCS revision identifier to the VMCS region before using that region for a VMCS. The VMCS revision identifier is never written by the processor; VMPTRLD fails if its operand references a VMCS region whose VMCS revision identifier differs from that used by the processor. (VMPTRLD also fails if the shadow-VMCS indicator is 1 and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control; see Section 25.6.2) Software can discover the VMCS revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

Software should clear or set the shadow-VMCS indicator depending on whether the VMCS is to be an ordinary VMCS or a shadow VMCS (see Section 25.10). VMPTRLD fails if the shadow-VMCS indicator is set and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control. Software can discover support for this setting by reading the VMX capability MSR IA32_VMX_PROCBASED_CTL2 (see Appendix A.3.3).

The next 4 bytes of the VMCS region are used for the **VMX-abort indicator**. The contents of these bits do not control processor operation in any way. A logical processor writes a non-zero value into these bits if a VMX abort occurs (see Section 28.7). Software may also write into this field.

The remainder of the VMCS region is used for **VMCS data** (those parts of the VMCS that control VMX non-root operation and the VMX transitions). The format of these data is implementation-specific. VMCS data are discussed in Section 25.3 through Section 25.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 25.11.4) in writeback cacheable memory. Future implementations may allow or require a different memory type³. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

25.3 ORGANIZATION OF VMCS DATA

The VMCS data are organized into six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** These fields control processor behavior in VMX non-root operation. They determine in part the causes of VM exits.
- **VM-exit control fields.** These fields control VM exits.
- **VM-entry control fields.** These fields control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits and describe the cause and the nature of VM exits. On some processors, these fields are read-only.⁴

The VM-execution control fields, the VM-exit control fields, and the VM-entry control fields are sometimes referred to collectively as VMX controls.

-
1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.
 2. Logical processors that use the same VMCS revision identifier use the same size for VMCS regions.
 3. Alternatively, software may map any of these regions or structures with the UC memory type. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.
 4. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

25.4 GUEST-STATE AREA

This section describes fields contained in the guest-state area of the VMCS. VM entries load processor state from these fields and VM exits store processor state into these fields. See Section 27.3.2 and Section 28.3 for details.

25.4.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Debug register DR7 (64 bits; 32 bits on processors that do not support Intel 64 architecture).
- RSP, RIP, and RFLAGS (64 bits each; 32 bits on processors that do not support Intel 64 architecture).¹
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
 - Selector (16 bits).
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits on processors that support Intel 64 architecture.
 - Segment limit (32 bits). The limit field is always a measure in bytes.
 - Access rights (32 bits). The format of this field is given in Table 25-2 and detailed as follows:
 - The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.
 - Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.²
 - Bits 31:17 are reserved.

Table 25-2. Format of Access Rights

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level
7	P — Segment present
11:8	Reserved
12	AVL — Available for use by system software

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

2. There are a few exceptions to this statement. For example, a segment with a non-null selector may be unusable following a task switch that fails after its commit point; see “Interrupt 10—Invalid TSS Exception (#TS)” in Section 6.14, “Exception and Interrupt Handling in 64-bit Mode,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 11-1 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 25-2. Format of Access Rights (Contd.)

Bit Position(s)	Field
13	Reserved (except for CS) L — 64-bit mode active (for CS only)
14	D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G — Granularity
16	Segment unusable (0 = usable; 1 = unusable)
31:17	Reserved

The base address, segment limit, and access rights compose the “hidden” part (or “descriptor cache”) of each segment register. These data are included in the VMCS because it is possible for a segment register’s descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register’s selector.

The value of the DPL field for SS is always equal to the logical processor’s current privilege level (CPL).¹

On some processors, executions of VMWRITE ignore attempts to write non-zero values to any of bits 11:8 or bits 31:17. On such processors, VMREAD always returns 0 for those bits, and VM entry treats those bits as if they were all 0 (see Section 27.3.1.2).

- The following fields for each of the registers GDTR and IDTR:
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - Limit (32 bits). The limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.
- The following MSRs:
 - IA32_DEBUGCTL (64 bits)
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture)
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.
 - IA32_PAT (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_PAT” VM-entry control or that of the “save IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_EFER” VM-entry control or that of the “save IA32_EFER” VM-exit control.
 - IA32_BNDCFGS (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control.
 - IA32_RTIT_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control.
 - IA32_LBR_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load guest IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
 - IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

1. In protected mode, CPL is also associated with the RPL field in the CS selector. However, the RPL fields are not meaningful in real-address mode or in virtual-8086 mode.

- IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-entry control.
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
- The register SMBASE (32 bits). This register contains the base address of the logical processor’s SMRAM image.

25.4.2 Guest Non-Register State

In addition to the register state described in Section 25.4.1, the guest-state area includes the following fields that characterize guest state but which do not correspond to processor registers:

- **Activity state** (32 bits). This field identifies the logical processor’s activity state. When a logical processor is executing instructions normally, it is in the **active state**. Execution of certain instructions and the occurrence of certain events may cause a logical processor to transition to an **inactive state** in which it ceases to execute instructions.

The following activity states are defined:¹

- 0: **Active**. The logical processor is executing instructions normally.
- 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.
- 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**² or some other serious error.
- 3: **Wait-for-SIPI**. The logical processor is inactive because it is waiting for a startup-IPI (SIPI).

Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.

- **Interruptibility state** (32 bits). The IA-32 architecture includes features that permit certain events to be blocked for a period of time. This field contains information about such blocking. Details and the format of this field are given in Table 25-3.

Table 25-3. Format of Interruptibility State

Bit Position(s)	Bit Name	Notes
0	Blocking by STI	See the “STI—Set Interrupt Flag” section in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B. Execution of STI with RFLAGS.IF = 0 blocks maskable interrupts on the instruction boundary following its execution. ¹ Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See Section 6.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. Execution of a MOV to SS or a POP to SS blocks or suppresses certain debug exceptions as well as interrupts (maskable and nonmaskable) on the instruction boundary following its execution. Setting this bit indicates that this blocking is in effect. ² This document uses the term “blocking by MOV SS,” but it applies equally to POP SS.
2	Blocking by SMI	See Section 32.2, “System Management Interrupt (SMI).” System-management interrupts (SMIs) are disabled while the processor is in system-management mode (SMM). Setting this bit indicates that blocking of SMIs is in effect.

1. Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 28.1.

2. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 25-3. Format of Interruptibility State (Contd.)

Bit Position(s)	Bit Name	Notes
3	Blocking by NMI	See Section 6.7.1, “Handling Multiple NMIs,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A and Section 32.8, “NMI Handling While in SMM.” Delivery of a non-maskable interrupt (NMI) or a system-management interrupt (SMI) blocks subsequent NMIs until the next execution of IRET. See Section 26.3 for how this behavior of IRET may change in VMX non-root operation. Setting this bit indicates that blocking of NMIs is in effect. Clearing this bit does not imply that NMIs are not (temporarily) blocked for other reasons. If the “virtual NMIs” VM-execution control (see Section 25.6.1) is 1, this bit does not control the blocking of NMIs. Instead, it refers to “virtual-NMI blocking” (the fact that guest software is not ready for an NMI).
4	Enclave interruption	Set to 1 if the VM exit occurred while the logical processor was in enclave mode. Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode. A VM exit that is incident to delivery of an event injected by VM entry leaves this bit unmodified.
31:5	Reserved	VM entry will fail if these bits are not 0. See Section 27.3.1.5.

NOTES:

1. Nonmaskable interrupts and system-management interrupts may also be inhibited on the instruction boundary following such an execution of STI.
 2. System-management interrupts may also be inhibited on the instruction boundary following such an execution of MOV or POP.
- **Pending debug exceptions** (64 bits; 32 bits on processors that do not support Intel 64 architecture). IA-32 processors may recognize one or more debug exceptions without immediately delivering them.¹ This field contains information about such exceptions. This field is described in Table 25-4.

Table 25-4. Format of Pending-Debug-Exceptions

Bit Position(s)	Bit Name	Notes
3:0	B3 - B0	When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if the corresponding enabling bit in DR7 is not set.
10:4	Reserved	VM entry fails if these bits are not 0. See Section 27.3.1.5.
11	BLD	When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 18.3.1.6 (“OS Bus-Lock Detection”).) ¹
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7; the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled; or a bus lock was asserted while CPL > 0 and OS bus-lock detection had been enabled.
13	Reserved	VM entry fails if this bit is not 0. See Section 27.3.1.5.

1. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 6.8.3 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In addition, certain events incident to an instruction (for example, an INIT signal) may take priority over debug traps generated by that instruction. See Table 6-2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 25-4. Format of Pending-Debug-Exceptions (Contd.)

Bit Position(s)	Bit Name	Notes
14	BS	When set, this bit indicates that a debug exception would have been triggered by single-step execution mode.
15	Reserved	VM entry fails if this bit is not 0. See Section 27.3.1.5.
16	RTM	When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 16.3.7, "RTM-Enabled Debugger Support," of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). ²
63:17	Reserved	VM entry fails if these bits are not 0. See Section 27.3.1.5. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- **VMCS link pointer** (64 bits). If the "VMCS shadowing" VM-execution control is 1, the VMREAD and VMWRITE instructions access the VMCS referenced by this pointer (see Section 25.10). Otherwise, software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 27.3.1.5).
- **VMX-preemption timer value** (32 bits). This field is supported only on processors that support the 1-setting of the "activate VMX-preemption timer" VM-execution control. This field contains the value that the VMX-preemption timer will use following the next VM entry with that setting. See Section 26.5.1 and Section 27.7.4.
- **Page-directory-pointer-table entries** (PDPTes; 64 bits each). These four (4) fields (PDPTE0, PDPTE1, PDPTE2, and PDPTE3) are supported only on processors that support the 1-setting of the "enable EPT" VM-execution control. They correspond to the PDPTes referenced by CR3 when PAE paging is in use (see Section 4.4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A). They are used only if the "enable EPT" VM-execution control is 1.
- **Guest interrupt status** (16 bits). This field is supported only on processors that support the 1-setting of the "virtual-interrupt delivery" VM-execution control. It characterizes part of the guest's virtual-APIC state and does not correspond to any processor or APIC registers. It comprises two 8-bit subfields:
 - **Requesting virtual interrupt (RVI)**. This is the low byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is requesting service. (The value 0 implies that there is no such interrupt.)
 - **Servicing virtual interrupt (SVI)**. This is the high byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is in service. (The value 0 implies that there is no such interrupt.)

See Chapter 30 for more information on the use of this field.
- **PML index** (16 bits). This field is supported only on processors that support the 1-setting of the "enable PML" VM-execution control. It contains the logical index of the next entry in the page-modification log. Because the page-modification log comprises 512 entries, the PML index is typically a value in the range 0–511. Details of the page-modification log and use of the PML index are given in Section 29.3.6.

25.5 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM exit (see Section 28.5).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- RSP and RIP (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. There is no field in the host-state area for the LDTR selector.
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- The following MSRs:
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
 - IA32_PAT (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_EFER” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-exit control.
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

In addition to the state identified here, some processor state components are loaded with fixed values on every VM exit; there are no fields corresponding to these components in the host-state area. See Section 28.5 for details of how state is loaded on VM exits.

25.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 25.6.1 through Section 25.6.8.

25.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts).¹ Table 25-5 lists the controls. See Chapter 28 for how these controls affect processor behavior in VMX non-root operation.

1. Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 26.2).

Table 25-5. Definitions of Pin-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	External-interrupt exiting	If this control is 1, external interrupts cause VM exits. Otherwise, they are delivered normally through the guest interrupt-descriptor table (IDT). If this control is 1, the value of RFLAGS.IF does not affect interrupt blocking.
3	NMI exiting	If this control is 1, non-maskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using descriptor 2 of the IDT. This control also determines interactions between IRET and blocking by NMI (see Section 26.3).
5	Virtual NMIs	If this control is 1, NMIs are never blocked and the “blocking by NMI” bit (bit 3) in the interruptibility-state field indicates “virtual-NMI blocking” (see Table 25-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 25.6.2).
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 26.5.1. A VM exit occurs when the timer counts down to zero; see Section 26.2.
7	Process posted interrupts	If this control is 1, the processor treats interrupts with the posted-interrupt notification vector (see Section 25.6.8) specially, updating the virtual-APIC page with posted-interrupt requests (see Section 30.6).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PINBASED_CTLs and IA32_VMX_TRUE_PINBASED_CTLs (see Appendix A.3.1) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 27.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR IA32_VMX_PINBASED_CTLs will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PINBASED_CTLs MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

25.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute three vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions.¹ These are the **primary processor-based VM-execution controls** (32 bits), the **secondary processor-based VM-execution controls** (32 bits), and the tertiary **VM-execution controls** (64 bits).

Table 25-6 lists the primary processor-based VM-execution controls. See Chapter 26 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 25-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 25.4.2).
3	Use TSC offsetting	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC offset field (see Section 25.6.5 and Section 26.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPIC exiting	This control determines whether executions of RDPIC cause VM exits.

1. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 26.1.2), as do task switches (see Section 26.2).

Table 25-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 25.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 26.1.3. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
17	Activate tertiary controls	This control determines whether the tertiary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the tertiary processor-based VM-execution controls were also 0.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits.
21	Use TPR shadow	Setting this control to 1 enables TPR virtualization and other APIC-virtualization features. See Chapter 30.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 25.4.2).
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 25.6.4 and Section 26.1.3). For this control, "0" means "do not use I/O bitmaps" and "1" means "use I/O bitmaps." If the I/O bitmaps are used, the setting of the "unconditional I/O exiting" control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 26.5.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 25.6.9 and Section 26.1.3). For this control, "0" means "do not use MSR bitmaps" and "1" means "use MSR bitmaps." If the MSR bitmaps are not used, all executions of the RDMSR and WRMSR instructions cause VM exits.
29	MONITOR exiting	This control determines whether executions of MONITOR cause VM exits.
30	PAUSE exiting	This control determines whether executions of PAUSE cause VM exits.
31	Activate secondary controls	This control determines whether the secondary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the secondary processor-based VM-execution controls were also 0.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PROCBASED_CTLs and IA32_VMX_TRUE_PROCBASED_CTLs (see Appendix A.3.2) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 27.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 4–6, 8, 13–16, and 26. The VMX capability MSR IA32_VMX_PROCBASED_CTLs will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PROCBASED_CTLs MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-execution controls determines whether the secondary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 31 of

the primary processor-based VM-execution controls do not support the secondary processor-based VM-execution controls.

Table 25-7 lists the secondary processor-based VM-execution controls. See Chapter 26 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 25-7. Definitions of Secondary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	Virtualize APIC accesses	If this control is 1, the logical processor treats specially accesses to the page with the APIC-access address. See Section 30.4.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 29.3.
2	Descriptor-table exiting	This control determines whether executions of LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR cause VM exits.
3	Enable RDTSCP	If this control is 0, any execution of RDTSCP causes an invalid-opcode exception (#UD).
4	Virtualize x2APIC mode	If this control is 1, the logical processor treats specially RDMSR and WRMSR to APIC MSRs (in the range 800H-8FFH). See Section 30.5.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 29.1.
6	WBINVD exiting	This control determines whether executions of WBINVD and WBNOINVD cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpagged protected mode or in real-address mode.
8	APIC-register virtualization	If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 30.4 and Section 30.5.
9	Virtual-interrupt delivery	This controls enables the evaluation and delivery of pending virtual interrupts as well as the emulation of writes to the APIC registers that control interrupt prioritization.
10	PAUSE-loop exiting	This control determines whether a series of executions of PAUSE can cause a VM exit (see Section 25.6.13 and Section 26.1.3).
11	RDRAND exiting	This control determines whether executions of RDRAND cause VM exits.
12	Enable INVPCID	If this control is 0, any execution of INVPCID causes a #UD.
13	Enable VM functions	Setting this control to 1 enables use of the VMFUNC instruction in VMX non-root operation. See Section 26.5.6.
14	VMCS shadowing	If this control is 1, executions of VMREAD and VMWRITE in VMX non-root operation may access a shadow VMCS (instead of causing VM exits). See Section 25.10 and Section 31.3.
15	Enable ENCLS exiting	If this control is 1, executions of ENCLS consult the ENCLS-exiting bitmap to determine whether the instruction causes a VM exit. See Section 25.6.16 and Section 26.1.3.
16	RDSEED exiting	This control determines whether executions of RDSEED cause VM exits.
17	Enable PML	If this control is 1, an access to a guest-physical address that sets an EPT dirty bit first adds an entry to the page-modification log. See Section 29.3.6.
18	EPT-violation #VE	If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 26.5.7.
19	Conceal VMX from PT	If this control is 1, Intel Processor Trace suppresses from PIPs an indication that the processor was in VMX non-root operation and omits a VMCS packet from any PSB+ produced in VMX non-root operation (see Chapter 33).
20	Enable XSAVES/XRSTORS	If this control is 0, any execution of XSAVES or XRSTORS causes a #UD.
21	PASID translation	If this control is 1, PASID translation is performed for executions of ENQCMD and ENQCMDs. See Section 26.5.8.
22	Mode-based execute control for EPT	If this control is 1, EPT execute permissions are based on whether the linear address being accessed is supervisor mode or user mode. See Chapter 29.

Table 25-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
23	Sub-page write permissions for EPT	If this control is 1, EPT write permissions may be specified at the granularity of 128 bytes. See Section 29.3.4.
24	Intel PT uses guest physical addresses	If this control is 1, all output addresses used by Intel Processor Trace are treated as guest-physical addresses and translated using EPT. See Section 26.5.4.
25	Use TSC scaling	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC multiplier field (see Section 25.6.5 and Section 26.3).
26	Enable user wait and pause	If this control is 0, any execution of TPAUSE, UMONITOR, or UMWAIT causes a #UD.
27	Enable PCONFIG	If this control is 0, any execution of PCONFIG causes a #UD.
28	Enable ENCLV exiting	If this control is 1, executions of ENCLV consult the ENCLV-exiting bitmap to determine whether the instruction causes a VM exit. See Section 25.6.17 and Section 26.1.3.
30	VMM bus-lock detection	This control determines whether assertion of a bus lock causes a VM exit. See Section 26.2.
31	Instruction timeout	If this control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within a specified amount of time. See Section 25.6.25 and Section 26.2.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTLDS2 (see Appendix A.3.3) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 27.2.1.1).

Bit 17 of the primary processor-based VM-execution controls determines whether the tertiary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the tertiary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 17 of the primary processor-based VM-execution controls do not support the tertiary processor-based VM-execution controls.

Table 25-8 lists the tertiary processor-based VM-execution controls. See Chapter 26 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 25-8. Definitions of Tertiary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	LOADIWKEY exiting	This control determines whether executions of LOADIWKEY cause VM exits.
1	Enable HLAT	This control enables hypervisor-managed linear-address translation. See Section 4.5.1.
2	EPT paging-write control	If this control is 1, EPT permissions can be specified to allow writes only for paging-related updates. See Section 29.3.3.2.
3	Guest-paging verification	If this control is 1, EPT permissions can be specified to prevent accesses using linear addresses whose translation has certain properties. See Section 29.3.3.2.
4	IPI virtualization	If this control is 1, virtualization of interprocessor interrupts (IPIs) is enabled. See Section 30.1.6.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTLDS3 (see Appendix A.3.4) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 27.2.1.1).

25.6.3 Exception Bitmap

The **exception bitmap** is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally through the IDT, using the descriptor corresponding to the exception's vector.

Whether a page fault (exception with vector 14) causes a VM exit is determined by bit 14 in the exception bitmap as well as the error code produced by the page fault and two 32-bit fields in the VMCS (the **page-fault error-code mask** and **page-fault error-code match**). See Section 26.2 for details.

25.6.4 I/O-Bitmap Addresses

The VM-execution control fields include the 64-bit physical addresses of **I/O bitmaps** A and B (each of which are 4 KBytes in size). I/O bitmap A contains one bit for each I/O port in the range 0000H through 7FFFH; I/O bitmap B contains bits for ports in the range 8000H through FFFFH.

A logical processor uses these bitmaps if and only if the “use I/O bitmaps” control is 1. If the bitmaps are used, execution of an I/O instruction causes a VM exit if any bit in the I/O bitmaps corresponding to a port it accesses is 1. See Section 26.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

25.6.5 Time-Stamp Counter Offset and Multiplier

The VM-execution control fields include a 64-bit **TSC-offset** field. If the “RDTSC exiting” control is 0 and the “use TSC offsetting” control is 1, this field controls executions of the RDTSC and RDTSCP instructions. It also controls executions of the RDMSR instruction that read from the IA32_TIME_STAMP_COUNTER MSR. For all of these, the value of the TSC offset is added to the value of the time-stamp counter, and the sum is returned to guest software in EDX:EAX.

Processors that support the 1-setting of the “use TSC scaling” control also support a 64-bit **TSC-multiplier** field. If this control is 1 (and the “RDTSC exiting” control is 0 and the “use TSC offsetting” control is 1), this field also affects the executions of the RDTSC, RDTSCP, and RDMSR instructions identified above. Specifically, the contents of the time-stamp counter is first multiplied by the TSC multiplier before adding the TSC offset.

See Chapter 26 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation.

25.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4

VM-execution control fields include **guest/host masks** and **read shadows** for the CR0 and CR4 registers. These fields control executions of instructions that access those registers (including CLTS, LMSW, MOV CR, and SMSW). They are 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

In general, bits set to 1 in a guest/host mask correspond to bits “owned” by the host:

- Guest attempts to set them (using CLTS, LMSW, or MOV to CR) to values differing from the corresponding bits in the corresponding read shadow cause VM exits.
- Guest reads (using MOV from CR or SMSW) return values for these bits from the corresponding read shadow.

Bits cleared to 0 correspond to bits “owned” by the guest; guest attempts to modify them succeed and guest reads return values for these bits from the control register itself.

See Chapter 28 for details regarding how these fields affect VMX non-root operation.

25.6.7 CR3-Target Controls

The VM-execution control fields include a set of 4 **CR3-target values** and a **CR3-target count**. The CR3-target values each have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not. The CR3-target count has 32 bits on all processors.

An execution of MOV to CR3 in VMX non-root operation does not cause a VM exit if its source operand matches one of these values. If the CR3-target count is n , only the first n CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit.

There are no limitations on the values that can be written for the CR3-target values. VM entry fails (see Section 27.2) if the CR3-target count is greater than 4.

Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine the number of values supported.

25.6.8 Controls for APIC Virtualization

There are three mechanisms by which software accesses registers of the logical processor's local APIC:

- If the local APIC is in xAPIC mode, it can perform memory-mapped accesses to addresses in the 4-KByte page referenced by the physical address in the IA32_APIC_BASE MSR (see Section 11.4.4, "Local APIC Status and Location" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A and Intel® 64 Architecture Processor Topology Enumeration).¹
- If the local APIC is in x2APIC mode, it can access the local APIC's registers using the RDMSR and WRMSR instructions (see Intel® 64 Architecture Processor Topology Enumeration).
- In 64-bit mode, it can access the local APIC's task-priority register (TPR) using the MOV CR8 instruction.

Several processor-based VM-execution controls (see Section 25.6.2) control such accesses. These are "use TPR shadow", "virtualize APIC accesses", "virtualize x2APIC mode", "virtual-interrupt delivery", "APIC-register virtualization", and "IPI virtualization". These controls interact with the following fields:

- **APIC-access address** (64 bits). This field contains the physical address of the 4-KByte **APIC-access page**. If the "virtualize APIC accesses" VM-execution control is 1, access to this page may cause VM exits or be virtualized by the processor. See Section 30.4.

The APIC-access address exists only on processors that support the 1-setting of the "virtualize APIC accesses" VM-execution control.

- **Virtual-APIC address** (64 bits). This field contains the physical address of the 4-KByte **virtual-APIC page**. The processor uses the virtual-APIC page to virtualize certain accesses to APIC registers and to manage virtual interrupts; see Chapter 30.

Depending on the setting of the controls indicated earlier, the virtual-APIC page may be accessed by the following operations:

- The MOV CR8 instructions (see Section 30.3).
- Accesses to the APIC-access page if, in addition, the "virtualize APIC accesses" VM-execution control is 1 (see Section 30.4).
- The RDMSR and WRMSR instructions if, in addition, the value of ECX is in the range 800H–8FFH (indicating an APIC MSR) and the "virtualize x2APIC mode" VM-execution control is 1 (see Section 30.5).

If the "use TPR shadow" VM-execution control is 1, VM entry ensures that the virtual-APIC address is 4-KByte aligned. The virtual-APIC address exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.

- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which bits 7:4 of VTPR (see Section 30.1.1) cannot fall. If the "virtual-interrupt delivery" VM-execution control is 0, a VM exit occurs after an operation (e.g., an execution of MOV to CR8) that reduces the value of those bits below the TPR threshold. See Section 30.1.2.

The TPR threshold exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.

- **EOI-exit bitmap** (4 fields; 64 bits each). These fields are supported only on processors that support the 1-setting of the "virtual-interrupt delivery" VM-execution control. They are used to determine which virtualized writes to the APIC's EOI register cause VM exits:

- EOI_EXIT0 contains bits for vectors from 0 (bit 0) to 63 (bit 63).
- EOI_EXIT1 contains bits for vectors from 64 (bit 0) to 127 (bit 63).
- EOI_EXIT2 contains bits for vectors from 128 (bit 0) to 191 (bit 63).
- EOI_EXIT3 contains bits for vectors from 192 (bit 0) to 255 (bit 63).

See Section 30.1.4 for more information on the use of this field.

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

- **Posted-interrupt notification vector** (16 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. Its low 8 bits contain the interrupt vector that is used to notify a logical processor that virtual interrupts have been posted. See Section 30.6 for more information on the use of this field.
- **Posted-interrupt descriptor address** (64 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. It is the physical address of a 64-byte aligned posted interrupt descriptor. See Section 30.6 for more information on the use of this field.
- **PID-pointer table address** (64 bits). This field contains the physical address of the **PID-pointer table**. If the “IPI virtualization” VM-execution control is 1, the logical processor uses entries in this table to virtualize IPIs. See Section 30.1.6.
- **Last PID-pointer index** (16 bits). This field contains the index of the last entry in the PID-pointer table.

25.6.9 MSR-Bitmap Address

On processors that support the 1-setting of the “use MSR bitmaps” VM-execution control, the VM-execution control fields include the 64-bit physical address of four contiguous **MSR bitmaps**, which are each 1-KByte in size. This field does not exist on processors that do not support the 1-setting of that control. The four bitmaps are:

- **Read bitmap for low MSRs** (located at the MSR-bitmap address). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Read bitmap for high MSRs** (located at the MSR-bitmap address plus 1024). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Write bitmap for low MSRs** (located at the MSR-bitmap address plus 2048). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.
- **Write bitmap for high MSRs** (located at the MSR-bitmap address plus 3072). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

A logical processor uses these bitmaps if and only if the “use MSR bitmaps” control is 1. If the bitmaps are used, an execution of RDMSR or WRMSR causes a VM exit if the value of RCX is in neither of the ranges covered by the bitmaps or if the appropriate bit in the MSR bitmaps (corresponding to the instruction and the RCX value) is 1. See Section 26.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

25.6.10 Executive-VMCS Pointer

The executive-VMCS pointer is a 64-bit field used in the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). SMM VM exits save this field as described in Section 32.15.2. VM entries that return from SMM use this field as described in Section 32.15.4.

25.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of the base of EPT PML4 table (see Section 29.3.2), as well as other EPT configuration information. The format of this field is shown in Table 25-9.

Table 25-9. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 29.3.7): 0 = Uncacheable (UC) 6 = Write-back (WB) Other values are reserved. ¹
5:3	This value is 1 less than the EPT page-walk length (see Section 29.3.2)
6	Setting this control to 1 enables accessed and dirty flags for EPT (see Section 29.3.5) ²
7	Setting this control to 1 enables enforcement of access rights for supervisor shadow-stack pages (see Section 29.3.3.2) ³
11:8	Reserved
N-1:12	Bits N-1:12 of the physical address of the 4-KByte aligned EPT PML4 table ⁴
63:N	Reserved

NOTES:

1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT paging-structure memory types are supported.
2. Not all processors support accessed and dirty flags for EPT. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
3. Not all processors enforce access rights for shadow-stack pages. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
4. N is the physical-address width supported by the logical processor. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

The EPTP exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

25.6.12 Virtual-Processor Identifier (VPID)

The **virtual-processor identifier** (VPID) is a 16-bit field. It exists only on processors that support the 1-setting of the "enable VPID" VM-execution control. See Section 29.1 for details regarding the use of this field.

25.6.13 Controls for PAUSE-Loop Exiting

On processors that support the 1-setting of the "PAUSE-loop exiting" VM-execution control, the VM-execution control fields include the following 32-bit fields:

- **PLE_Gap.** Software can configure this field as an upper bound on the amount of time between two successive executions of PAUSE in a loop.
- **PLE_Window.** Software can configure this field as an upper bound on the amount of time a guest is allowed to execute in a PAUSE loop.

These fields measure time based on a counter that runs at the same rate as the timestamp counter (TSC). See Section 26.1.3 for more details regarding PAUSE-loop exiting.

25.6.14 VM-Function Controls

The **VM-function controls** constitute a 64-bit vector that governs use of the VMFUNC instruction in VMX non-root operation. This field is supported only on processors that support the 1-settings of both the “activate secondary controls” primary processor-based VM-execution control and the “enable VM functions” secondary processor-based VM-execution control.

Table 25-10 lists the VM-function controls. See Section 26.5.6 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 25-10. Definitions of VM-Function Controls

Bit Position(s)	Name	Description
0	EPTP switching	The EPTP-switching VM function changes the EPT pointer to a value chosen from the EPTP list. See Section 26.5.6.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_VMFUNC (see Appendix A.11) to determine which bits are reserved. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 27.2.1.1).

Processors that support the 1-setting of the “EPTP switching” VM-function control also support a 64-bit field called the **EPTP-list address**. This field contains the physical address of the 4-KByte **EPTP list**. The EPTP list comprises 512 8-Byte entries (each an EPTP value) and is used by the EPTP-switching VM function (see Section 26.5.6.3).

25.6.15 VMCS Shadowing Bitmap Addresses

On processors that support the 1-setting of the “VMCS shadowing” VM-execution control, the VM-execution control fields include the 64-bit physical addresses of the **VMREAD bitmap** and the **VMWRITE bitmap**. Each bitmap is 4 KBytes in size and thus contains 32 KBits. The addresses are the **VMREAD-bitmap address** and the **VMWRITE-bitmap address**.

If the “VMCS shadowing” VM-execution control is 1, executions of VMREAD and VMWRITE may consult these bitmaps (see Section 25.10 and Section 31.3).

25.6.16 ENCLS-Exiting Bitmap

The **ENCLS-exiting bitmap** is a 64-bit field. If the “enable ENCLS exiting” VM-execution control is 1, execution of ENCLS causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the bit is 0, the instruction executes normally. See Section 26.1.3 for more information.

25.6.17 ENCLV-Exiting Bitmap

The **ENCLV-exiting bitmap** is a 64-bit field. If the “enable ENCLV exiting” VM-execution control is 1, execution of ENCLV causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the bit is 0, the instruction executes normally. See Section 26.1.3 for more information.

25.6.18 PCONFIG-Exiting Bitmap

The **PCONFIG-exiting bitmap** is a 64-bit field. If the “enable PCONFIG” VM-execution control is 1, execution of PCONFIG causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the control is 0, any execution of PCONFIG causes a #UD. See Section 26.1.3 for more information.

25.6.19 Control Field for Page-Modification Logging

The **PML address** is a 64-bit field. It is the 4-KByte aligned address of the **page-modification log**. The page-modification log consists of 512 64-bit entries. It is used for the page-modification logging feature. Details of the page-modification logging are given in Section 29.3.6.

If the “enable PML” VM-execution control is 1, VM entry ensures that the PML address is 4-KByte aligned. The PML address exists only on processors that support the 1-setting of the “enable PML” VM-execution control.

25.6.20 Controls for Virtualization Exceptions

On processors that support the 1-setting of the “EPT-violation #VE” VM-execution control, the VM-execution control fields include the following:

- **Virtualization-exception information address** (64 bits). This field contains the physical address of the **virtualization-exception information area**. When a logical processor encounters a virtualization exception, it saves virtualization-exception information at the virtualization-exception information address; see Section 26.5.7.2.
- **EPTP index** (16 bits). When an EPT violation causes a virtualization exception, the processor writes the value of this field to the virtualization-exception information area. The EPTP-switching VM function updates this field (see Section 26.5.6.3).

25.6.21 XSS-Exiting Bitmap

On processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control, the VM-execution control fields include a 64-bit **XSS-exiting bitmap**. If the “enable XSAVES/XRSTORS” VM-execution control is 1, executions of XSAVES and XRSTORS may consult this bitmap (see Section 26.1.3 and Section 26.3).

25.6.22 Sub-Page-Permission-Table Pointer (SPPTP)

If the sub-page write-permission feature of EPT is enabled, EPT write permissions may be determined at a 128-byte granularity (see Section 29.3.4). These permissions are determined using a hierarchy of sub-page-permission structures in memory.

The root of this hierarchy is referenced by a VM-execution control field called the **sub-page-permission-table pointer** (SPPTP). The SPPTP contains the address of the base of the root SPP table (see Section 29.3.4.2). The format of this field is shown in Table 25-9.

Table 25-11. Format of Sub-Page-Permission-Table Pointer

Bit Position(s)	Field
11:0	Reserved
N-1:12	Bits N-1:12 of the physical address of the 4-KByte aligned root SPP table
63:N ¹	Reserved

NOTES:

1. N is the processor’s physical-address width. Software can determine this width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

The SPPTP exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.

25.6.23 Fields Related to Hypervisor-Managed Linear-Address Translation

Two fields are used when the “enable HLAT” VM-execution control is 1, enabling HLAT paging:

- The **hypervisor-managed linear-address translation pointer** (HLAT pointer or HLATP) is used by HLAT paging to locate and access the first paging structure used for linear-address translation (see Section 4.5). The format of this field is shown in Table 25-12.

Table 25-12. Format of Hypervisor-Managed Linear-Address Translation Pointer

Bit Position(s)	Field
2:0	Reserved
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
11:5	Reserved
N-1:12	Guest-physical address (4KB-aligned) of the first HLAT paging structure during linear-address translation. ¹
63:N	Reserved

NOTES:

1. N is the physical-address width supported by the logical processor. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

- The HLAT prefix size. The value of this field determines which linear address are subject to HLAT paging. See Section 4.5.1.

These fields exist only on processors that support the 1-setting of the “enable HLAT” VM-execution control.

25.6.24 Fields Related to PASID Translation

Two 64-bit VM-execution control fields are used when the “PASID translation” VM-execution control is 1, enabling translation of PASIDs for executions of ENQCMD and ENQCMDS: the **low PASID directory address** and the **high PASID directory address**. These are the physical addresses of the low PASID directory and the high PASID directory, respectively. These fields exist only on processors that support the 1-setting of the “PASID translation” VM-execution control.

See Section 26.5.8 for information on the PASID-translation process for ENQCMD and ENQCMDS.

25.6.25 Instruction-Timeout Control

On processors that support the 1-setting of the “instruction timeout” VM-execution control, the VM-execution control fields include a 32-bit **instruction-timeout control**. The processor interprets the value of this field as an amount of time as measured in units of crystal clock cycles.¹ If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within this amount of time.

1. CPUID.15H:ECX enumerates the nominal frequency of the core crystal clock in Hz.

25.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 25.7.1 and Section 25.7.2.

25.7.1 VM-Exit Controls

The VM-exit controls constitute two vectors that govern the basic operation of VM exits. These are the **primary VM-exit controls** (32 bits) and the **secondary VM-exits controls** (64 bits).

Table 25-13 lists the primary VM-exit controls. See Chapter 28 for complete details of how these controls affect VM exits.

Table 25-13. Definitions of Primary VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: <ul style="list-style-type: none"> ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt's vector. The vector is stored in the VM-exit interruption-information field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the VM-exit interruption-information field is marked invalid.
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.
22	Save VMX-preemption timer value	This control determines whether the value of the VMX-preemption timer is saved on VM exit.
23	Clear IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is cleared on VM exit.
24	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM exit or a VMCS packet on an SMM VM exit (see Chapter 33).
25	Clear IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is cleared on VM exit.
26	Clear IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is cleared on VM exit.
27	Clear UINV	This control determines whether UINV is cleared on VM exit.
28	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM exit.
29	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM exit.
30	Save IA32_PERF_GLOBAL_CTL	This control determines whether the IA32_PERF_GLOBAL_CTL MSR is saved on VM exit.

Table 25-13. Definitions of Primary VM-Exit Controls (Contd.)

Bit Position(s)	Name	Description
31	Activate secondary controls	This control determines whether the secondary VM-exit controls are used. If this control is 0, the logical processor operates as if all the secondary VM-exit controls were also 0.

NOTES:

1. Since the Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CRO.PG and IA32_EFER.LME, and since CRO.PG is always 1 in VMX root operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX root operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_EXIT_CTLS and IA32_VMX_TRUE_EXIT_CTLS (see Appendix A.4) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 27.2.1.2).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8, 10, 11, 13, 14, 16, and 17. The VMX capability MSR IA32_VMX_EXIT_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_EXIT_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-exit controls determines whether the secondary VM-exit controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary VM-exit controls were 0. Processors that support only the 0-setting of bit 31 of the primary VM-exit controls do not support the secondary VM-exit controls.

Currently, no secondary VM-exit controls are defined, and all bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_EXIT_CTLS2 (see Appendix A.4.2) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 27.2.1.2).

25.7.2 VM-Exit Controls for MSRs

A VMM may specify lists of MSRs to be stored and loaded on VM exits. The following VM-exit control fields determine how MSRs are stored on VM exits:

- **VM-exit MSR-store count** (32 bits). This field specifies the number of MSRs to be stored on VM exit. It is recommended that this count not exceed 512.¹ Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.
- **VM-exit MSR-store address** (64 bits). This field contains the physical address of the VM-exit MSR-store area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-store count. The format of each entry is given in Table 25-14. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 25-14. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 28.4 for how this area is used on VM exits.

The following VM-exit control fields determine how MSRs are loaded on VM exits:

1. Future implementations may allow more MSRs to be stored reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.¹
- **VM-exit MSR-load address** (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count (see Table 25-14). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 28.6 for how this area is used on VM exits.

25.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 25.8.1 through 25.8.3.

25.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 25-15 lists the controls supported. See Chapter 25 for how these controls affect VM entries.

Table 25-15. Definitions of VM-Entry Controls

Bit Position(s)	Name	Description
2	Load debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are loaded on VM entry. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	IA-32e mode guest	On processors that support Intel 64 architecture, this control determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA as part of VM entry. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
10	Entry to SMM	This control determines whether the logical processor is in system-management mode (SMM) after VM entry. This control must be 0 for any VM entry from outside SMM.
11	Deactivate dual-monitor treatment	If set to 1, the default treatment of SMIs and SMM is in effect after the VM entry (see Section 32.15.7). This control must be 0 for any VM entry from outside SMM.
13	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM entry.
14	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM entry.
15	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM entry.
16	Load IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is loaded on VM entry.
17	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM entry or a VMCS packet on a VM entry that returns from SMM (see Chapter 33).
18	Load IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is loaded on VM entry.
19	Load UINV	This control determines whether UINV is loaded on VM entry.

1. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

Table 25-15. Definitions of VM-Entry Controls (Contd.)

Bit Position(s)	Name	Description
20	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM entry.
21	Load guest IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is loaded on VM entry.
22	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM entry.

NOTES:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control. If it is read as 1, every VM exit stores the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control (see Section 28.2).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_ENTRY_CTLS and IA32_VMX_TRUE_ENTRY_CTLS (see Appendix A.5) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 27.2.1.3).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8 and 12. The VMX capability MSR IA32_VMX_ENTRY_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_ENTRY_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

25.8.2 VM-Entry Controls for MSRs

A VMM may specify a list of MSRs to be loaded on VM entries. The following VM-entry control fields manage this functionality:

- **VM-entry MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM entry. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM entry.¹
- **VM-entry MSR-load address** (64 bits). This field contains the physical address of the VM-entry MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-entry MSR-load count. The format of entries is described in Table 25-14. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 27.4 for details of how this area is used on VM entries.

25.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event through the IDT (after all guest state and MSRs have been loaded). This process is called **event injection** and is controlled by the following three VM-entry control fields:

- **VM-entry interruption-information field** (32 bits). This field provides details about the event to be injected. Table 25-16 describes the field.

1. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

Table 25-16. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g., #PF) 4: Software interrupt (INT <i>n</i>) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

- The **vector** (bits 7:0) determines which entry in the IDT is used or which other event is injected.
- The **interruption type** (bits 10:8) determines details of how the injection is performed. In general, a VMM should use the type hardware exception for all exceptions **other than** the following:
 - breakpoint exceptions (#BP; a VMM should use the type software exception);
 - overflow exceptions (#OF a VMM should use the use type software exception); and
 - those debug exceptions (#DB) that are generated by INT1 (a VMM should use the use type privileged software exception).¹

The type **other event** is used for injection of events that are not delivered through the IDT.²

- For exceptions, the **deliver-error-code bit** (bit 11) determines whether delivery pushes an error code on the guest stack.
- VM entry injects an event if and only if the **valid bit** (bit 31) is 1. The valid bit in this field is cleared on every VM exit (see Section 28.2).
- **VM-entry exception error code** (32 bits). This field is used if and only if the valid bit (bit 31) and the deliver-error-code bit (bit 11) are both set in the VM-entry interruption-information field.
- **VM-entry instruction length** (32 bits). For injection of events whose type is software interrupt, software exception, or privileged software exception, this field is used to determine the value of RIP that is pushed on the stack.

See Section 27.6 for details regarding the mechanics of event injection, including the use of the interruption type and the VM-entry instruction length.

VM exits clear the valid bit (bit 31) in the VM-entry interruption-information field.

25.9 VM-EXIT INFORMATION FIELDS

The VMCS contains a section of fields that contain information about the most recent VM exit.

On some processors, attempts to write to these fields with VMWRITE fail (see “VMWRITE—Write Field to Virtual-Machine Control Structure” in Chapter 31).³

-
1. The type hardware exception should be used for all other debug exceptions.
 2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with values 1 or 3 for *n*.
 3. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

25.9.1 Basic VM-Exit Information

The following VM-exit information fields provide basic information about a VM exit:

- **Exit reason** (32 bits). This field encodes the reason for the VM exit and has the structure given in Table 25-17.

Table 25-17. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason
16	Always cleared to 0
26:17	Not currently defined
27	A VM exit saves this bit as 1 to indicate that the VM exit was incident to enclave mode.
28	Pending MTF VM exit
29	VM exit from VMX root operation
30	Not currently defined
31	VM-entry failure (0 = true VM exit; 1 = VM-entry failure)

- Bits 15:0 provide basic information about the cause of the VM exit (if bit 31 is clear) or of the VM-entry failure (if bit 31 is set). Appendix C enumerates the basic exit reasons.
- Bit 16 is always cleared to 0.
- Bit 27 is set to 1 if the VM exit occurred while the logical processor was in enclave mode.
A VM exit also sets this bit if it is incident to delivery of an event injected by VM entry and the guest interruptibility-state field indicates an enclave interrupt (bit 4 of the field is 1). See Section 28.2.1 for details.
- Bit 28 is set only by an SMM VM exit (see Section 32.15.2) that took priority over an MTF VM exit (see Section 26.5.2) that would have occurred had the SMM VM exit not occurred. See Section 32.15.2.3.
- Bit 29 is set if and only if the processor was in VMX root operation at the time the VM exit occurred. This can happen only for SMM VM exits. See Section 32.15.2.
- Because some VM-entry failures load processor state from the host-state area (see Section 27.8), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose.
- **Exit qualification** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field contains additional information about the cause of VM exits due to the following: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); task switches; INVEPT; INVLPG; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; and MWAIT. The format of the field depends on the cause of the VM exit. See Section 28.2.1 for details.
- **Guest-linear address** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is used in the following cases:
 - VM exits due to attempts to execute LMSW with a memory operand.
 - VM exits due to attempts to execute INS or OUTS.
 - VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions.
 - Certain VM exits due to EPT violations
 See Section 28.2.1 and Section 32.15.2.3 for details of when and how this field is used.
- **Guest-physical address** (64 bits). This field is used by VM exits due to EPT violations and EPT misconfigurations. See Section 28.2.1 for details of when and how this field is used.

25.9.2 Information for VM Exits Due to Vectored Events

Event-specific information is provided for VM exits due to the following vectored events: exceptions (including those generated by the instructions INT3, INTO, INT1, BOUND, UD0, UD1, and UD2); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs). This information is provided in the following fields:

- **VM-exit interruption information** (32 bits). This field receives basic information associated with the event causing the VM exit. Table 25-18 describes this field.

Table 25-18. Format of the VM-Exit Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Not used 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	NMI unblocking due to IRET
30:13	Not currently defined
31	Valid

- **VM-exit interruption error code** (32 bits). For VM exits caused by hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

Section 28.2.2 provides details of how these fields are saved on VM exits.

25.9.3 Information for VM Exits That Occur During Event Delivery

Additional information is provided for VM exits that occur during event delivery in VMX non-root operation.¹ This information is provided in the following fields:

- **IDT-vectoring information** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 25-19 describes this field.

Table 25-19. Format of the IDT-Vectoring Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Not used

1. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 27.6.1.2.

Table 25-19. Format of the IDT-Vectoring Information Field (Contd.)

Bit Position(s)	Content
11	Error code valid (0 = invalid; 1 = valid)
30:12	Not currently defined
31	Valid

- **IDT-vectoring error code** (32 bits). For VM exits that occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

See Section 28.2.4 provides details of how these fields are saved on VM exits.

25.9.4 Information for VM Exits Due to Instruction Execution

The following fields are used for VM exits caused by attempts to execute certain instructions in VMX non-root operation:

- **VM-exit instruction length** (32 bits). For VM exits resulting from instruction execution, this field receives the length in bytes of the instruction whose execution led to the VM exit.¹ See Section 28.2.5 for details of when and how this field is used.
- **VM-exit instruction information** (32 bits). This field is used for VM exits due to attempts to execute `INS`, `INVEPT`, `INVVPID`, `LIDT`, `LGDT`, `LLDT`, `LTR`, `OUTS`, `SIDT`, `SGDT`, `SLDT`, `STR`, `VMCLEAR`, `VMPTRLD`, `VMPTRST`, `VMREAD`, `VMWRITE`, or `VMXON`.² The format of the field depends on the cause of the VM exit. See Section 28.2.5 for details.

The following fields (64 bits each; 32 bits on processors that do not support Intel 64 architecture) are used only for VM exits due to SMIs that arrive immediately after retirement of I/O instructions. They provide information about that I/O instruction:

- **I/O RCX**. The value of RCX before the I/O instruction started.
- **I/O RSI**. The value of RSI before the I/O instruction started.
- **I/O RDI**. The value of RDI before the I/O instruction started.
- **I/O RIP**. The value of RIP before the I/O instruction started (the RIP that addressed the I/O instruction).

25.9.5 VM-Instruction Error Field

The 32-bit **VM-instruction error field** does not provide information about the most recent VM exit. In fact, it is not modified on VM exits. Instead, it provides information about errors encountered by a non-faulting execution of one of the VMX instructions.

25.10 VMCS TYPES: ORDINARY AND SHADOW

Every VMCS is either an **ordinary VMCS** or a **shadow VMCS**. A VMCS's type is determined by the shadow-VMCS indicator in the VMCS region (this is the value of bit 31 of the first 4 bytes of the VMCS region; see Table 25-1): 0 indicates an ordinary VMCS, while 1 indicates a shadow VMCS. Shadow VMCSs are supported only on processors that support the 1-setting of the "VMCS shadowing" VM-execution control (see Section 25.6.2).

A shadow VMCS differs from an ordinary VMCS in two ways:

- An ordinary VMCS can be used for VM entry but a shadow VMCS cannot. Attempts to perform VM entry when the current VMCS is a shadow VMCS fail (see Section 27.1).

1. This field is also used for VM exits that occur during the delivery of a software interrupt or software exception.
 2. Whether the processor provides this information on VM exits due to attempts to execute `INS` or `OUTS` can be determined by consulting the VMX capability MSR `IA32_VMX_BASIC` (see Appendix A.1).

- The VMREAD and VMWRITE instructions can be used in VMX non-root operation to access a shadow VMCS but not an ordinary VMCS. This fact results from the following:
 - If the “VMCS shadowing” VM-execution control is 0, execution of the VMREAD and VMWRITE instructions in VMX non-root operation always cause VM exits (see Section 26.1.3).
 - If the “VMCS shadowing” VM-execution control is 1, execution of the VMREAD and VMWRITE instructions in VMX non-root operation can access the VMCS referenced by the VMCS link pointer (see Section 31.3).
 - If the “VMCS shadowing” VM-execution control is 1, VM entry ensures that any VMCS referenced by the VMCS link pointer is a shadow VMCS (see Section 27.3.1.5).

In VMX root operation, both types of VMCSs can be accessed with the VMREAD and VMWRITE instructions.

Software should not modify the shadow-VMCS indicator in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted (see Section 25.11.1). Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

25.11 SOFTWARE USE OF THE VMCS AND RELATED STRUCTURES

This section details guidelines that software should observe when using a VMCS and related structures. It also provides descriptions of consequences for failing to follow guidelines.

25.11.1 Software Use of Virtual-Machine Control Structures

To ensure proper processor behavior, software should observe certain guidelines when using an active VMCS.

No VMCS should ever be active on more than one logical processor. If a VMCS is to be “migrated” from one logical processor to another, the first logical processor should execute VMCLEAR for the VMCS (to make it inactive on that logical processor and to ensure that all VMCS data are in memory) before the other logical processor executes VMPTRLD for the VMCS (to make it active on the second logical processor).¹ A VMCS that is made active on more than one logical processor may become **corrupted** (see below).

Software should not modify the shadow-VMCS indicator (see Table 25-1) in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted. Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Section 25.11.2). Software should never access or modify the VMCS data of an active VMCS using ordinary memory operations, in part because the format used to store the VMCS data is implementation-specific and not architecturally defined, and also because a logical processor may maintain some VMCS data of an active VMCS on the processor and not in the VMCS region. The following items detail some of the hazards of accessing VMCS data using ordinary memory operations:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS. Results may vary from time to time or from logical processor to logical processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. Doing so may cause the VMCS to become corrupted (see below).

(Software can avoid these hazards by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.)

If a logical processor leaves VMX operation, any VMCSs active on that logical processor may be corrupted (see below). To prevent such corruption of a VMCS that may be used either after a return to VMX operation or on another logical processor, software should execute VMCLEAR for that VMCS before executing the VMXOFF instruction or removing power from the processor (e.g., as part of a transition to the S3 and S4 power states).

1. As noted in Section 25.1, execution of the VMPTRLD instruction makes a VMCS is active. In addition, VM entry makes active any shadow VMCS referenced by the VMCS link pointer in the current VMCS. If a shadow VMCS is made active by VM entry, it is necessary to execute VMCLEAR for that VMCS before allowing that VMCS to become active on another logical processor.

This section has identified operations that may cause a VMCS to become corrupted. These operations may cause the VMCS’s data to become undefined. Behavior may be unpredictable if that VMCS used subsequently on any logical processor. The following items detail some hazards of VMCS corruption:

- VM entries may fail for unexplained reasons or may load undesired processor state.
- The processor may not correctly support VMX non-root operation as documented in Chapter 26 and may generate unexpected VM exits.
- VM exits may load undesired processor state, save incorrect state into the VMCS, or cause the logical processor to transition to a shutdown state.

25.11.2 VMREAD, VMWRITE, and Encodings of VMCS Fields

Every field of the VMCS is associated with a 32-bit value that is its **encoding**. The encoding is provided in an operand to VMREAD and VMWRITE when software wishes to read or write that field. These instructions fail if given, in 64-bit mode, an operand that sets an encoding bit beyond bit 32. See Chapter 31 for a description of these instructions.

The structure of the 32-bit encodings of the VMCS components is determined principally by the width of the fields and their function in the VMCS. See Table 25-20.

Table 25-20. Structure of VMCS Component Encoding

Bit Position(s)	Contents
0	Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural-width fields
9:1	Index
11:10	Type: 0: control 1: VM-exit information 2: guest state 3: host state
12	Reserved (must be 0)
14:13	Width: 0: 16-bit 1: 64-bit 2: 32-bit 3: natural-width
31:15	Reserved (must be 0)

The following items detail the meaning of the bits in each encoding:

- **Field width.** Bits 14:13 encode the width of the field.
 - A value of 0 indicates a 16-bit field.
 - A value of 1 indicates a 64-bit field.
 - A value of 2 indicates a 32-bit field.
 - A value of 3 indicates a **natural-width** field. Such fields have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

Fields whose encodings use value 1 are specially treated to allow 32-bit software access to all 64 bits of the field. Such access is allowed by defining, for each such field, an encoding that allows direct access to the high 32 bits of the field. See below.
- **Field type.** Bits 11:10 encode the type of VMCS field: control, guest-state, host-state, or VM-exit information. (The last category also includes the VM-instruction error field.)

- **Index.** Bits 9:1 distinguish components with the same field width and type.
- **Access type.** Bit 0 must be 0 for all fields except for 64-bit fields (those with field-width 1; see above). A VMREAD or VMWRITE using an encoding with this bit cleared to 0 accesses the entire field. For a 64-bit field with field-width 1, a VMREAD or VMWRITE using an encoding with this bit set to 1 accesses only the high 32 bits of the field.

Appendix B gives the encodings of all fields in the VMCS.

The following describes the operation of VMREAD and VMWRITE based on processor mode, VMCS-field width, and access type:

- 16-bit fields:
 - A VMREAD returns the value of the field in bits 15:0 of the destination operand; other bits of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 15:0 of the source operand into the VMCS field; other bits of the source operand are not used.
- 32-bit fields:
 - A VMREAD returns the value of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 31:0 of the source operand into the VMCS field; in 64-bit mode, bits 63:32 of the source operand are not used.
- 64-bit fields and natural-width fields using the full access type outside IA-32e mode.
 - A VMREAD returns the value of bits 31:0 of the field in its destination operand; bits 63:32 of the field are ignored.
 - A VMWRITE writes the value of its source operand to bits 31:0 of the field and clears bits 63:32 of the field.
- 64-bit fields and natural-width fields using the full access type in 64-bit mode (only on processors that support Intel 64 architecture).
 - A VMREAD returns the value of the field in bits 63:0 of the destination operand
 - A VMWRITE writes the value of bits 63:0 of the source operand into the VMCS field.
- 64-bit fields using the high access type.
 - A VMREAD returns the value of bits 63:32 of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 31:0 of the source operand to bits 63:32 of the field; in 64-bit mode, bits 63:32 of the source operand are not used.

Software seeking to read a 64-bit field outside IA-32e mode can use VMREAD with the full access type (reading bits 31:0 of the field) and VMREAD with the high access type (reading bits 63:32 of the field); the order of the two VMREAD executions is not important. Software seeking to modify a 64-bit field outside IA-32e mode should first use VMWRITE with the full access type (establishing bits 31:0 of the field while clearing bits 63:32) and then use VMWRITE with the high access type (establishing bits 63:32 of the field).

25.11.3 Initializing a VMCS

Software should initialize fields in a VMCS (using VMWRITE) before using the VMCS for VM entry. Failure to do so may result in unpredictable behavior; for example, a VM entry may fail for unexplained reasons, or a successful transition (VM entry or VM exit) may load processor state with unexpected values.

It is not necessary to initialize fields that the logical processor will not use. (For example, it is not necessary to initialize the MSR-bitmap address if the “use MSR bitmaps” VM-execution control is 0.)

A processor maintains some VMCS information that cannot be modified with the VMWRITE instruction; this includes a VMCS’s launch state (see Section 25.1). Such information may be stored in the VMCS data portion of a VMCS region. Because the format of this information is implementation-specific, there is no way for software to know, when it first allocates a region of memory for use as a VMCS region, how the processor will determine this information from the contents of the memory region.

In addition to its other functions, the VMCLEAR instruction initializes any implementation-specific information in the VMCS region referenced by its operand. To avoid the uncertainties of implementation-specific behavior, software should execute VMCLEAR on a VMCS region before making the corresponding VMCS active with VMPTRLD for the first time. (Figure 25-1 illustrates how execution of VMCLEAR puts a VMCS into a well-defined state.)

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry for the first time.
- VMLAUNCH should be used for the first VM entry using a VMCS after VMCLEAR has been executed for that VMCS.
- VMRESUME should be used for any subsequent VM entry using a VMCS (until the next execution of VMCLEAR for the VMCS).

It is expected that, in general, VMRESUME will have lower latency than VMLAUNCH. Since “migrating” a VMCS from one logical processor to another requires use of VMCLEAR (see Section 25.11.1), which sets the launch state of the VMCS to “clear”, such migration requires the next VM entry to be performed using VMLAUNCH. Software developers can avoid the performance cost of increased VM-entry latency by avoiding unnecessary migration of a VMCS from one logical processor to another.

25.11.4 Software Access to Related Structures

In addition to data in the VMCS region itself, VMX non-root operation can be controlled by data structures that are referenced by pointers in a VMCS (for example, the I/O bitmaps). While the pointers to these data structures are parts of the VMCS, the data structures themselves are not. They are not accessible using VMREAD and VMWRITE but by ordinary memory writes.

Software should ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 25.11.1). Exceptions are made for the following data structures (subject to detailed discussion in the sections indicated): EPT paging structures and the data structures used to locate SPP vectors (Section 29.4.3); the virtual-APIC page (Section 30.1); the posted interrupt descriptor (Section 30.6); and the virtualization-exception information area (Section 26.5.7.2).

25.11.5 VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region)¹ that the logical processor uses to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. The VMXON pointer is subject to the limitations that apply to VMCS pointers:

- The VMXON pointer must be 4-KByte aligned (bits 11:0 must be zero).
- The VMXON pointer must not set any bits beyond the processor’s physical-address width.^{2,3}

Before executing VMXON, software should write the VMCS revision identifier (see Section 25.2) to the VMXON region. (Specifically, it should write the 31-bit VMCS revision identifier to bits 30:0 of the first 4 bytes of the VMXON region; bit 31 should be cleared to 0.) It need not initialize the VMXON region in any other way. Software should use a separate region for each logical processor and should not access or modify the VMXON region of a logical processor between execution of VMXON and VMXOFF on that logical processor. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 25.11.1).

1. The amount of memory required for the VMXON region is the same as that required for a VMCS region. This size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

2. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

3. If IA32_VMX_BASIC[48] is read as 1, the VMXON pointer must not set any bits in the range 63:32; see Appendix A.1.

13. Updates to Chapter 26, Volume 3C

Change bars and green text show changes to Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Updates throughout the chapter for bus-lock detection and instruction timeout features.

In a virtualized environment using VMX, the guest software stack typically runs on a logical processor in VMX non-root operation. This mode of operation is similar to that of ordinary processor operation outside of the virtualized environment. This chapter describes the differences between VMX non-root operation and ordinary processor operation with special attention to causes of VM exits (which bring a logical processor from VMX non-root operation to root operation). The differences between VMX non-root operation and ordinary processor operation are described in the following sections:

- Section 26.1, “Instructions That Cause VM Exits.”
- Section 26.2, “Other Causes of VM Exits.”
- Section 26.3, “Changes to Instruction Behavior in VMX Non-Root Operation.”
- Section 26.4, “Other Changes in VMX Non-Root Operation.”
- Section 26.5, “Features Specific to VMX Non-Root Operation.”
- Section 26.6, “Unrestricted Guests.”

Chapter 27, “VM Entries,” describes the data control structures that govern VMX non-root operation. Chapter 27, “VM Entries,” describes the operation of VM entries by which the processor transitions from VMX root operation to VMX non-root operation. Chapter 26, “VMX Non-Root Operation,” describes the operation of VM exits by which the processor transitions from VMX non-root operation to VMX root operation.

Chapter 29, “VMX Support for Address Translation,” describes two features that support address translation in VMX non-root operation. Chapter 30, “APIC Virtualization and Virtual Interrupts,” describes features that support virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC) in VMX non-root operation.

26.1 INSTRUCTIONS THAT CAUSE VM EXITS

Certain instructions may cause VM exits if executed in VMX non-root operation. Unless otherwise specified, such VM exits are “fault-like,” meaning that the instruction causing the VM exit does not execute and no processor state is updated by the instruction. Section 28.1 details architectural state in the context of a VM exit.

Section 26.1.1 defines the prioritization between faults and VM exits for instructions subject to both. Section 26.1.2 identifies instructions that cause VM exits whenever they are executed in VMX non-root operation (and thus can never be executed in VMX non-root operation). Section 26.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields (see Section 25.6).

26.1.1 Relative Priority of Faults and VM Exits

The following principles describe the ordering between existing faults and VM exits:

- Certain exceptions have priority over VM exits. These include invalid-opcode exceptions, faults based on privilege level,¹ and general-protection exceptions that are based on checking I/O permission bits in the task-state segment (TSS). For example, execution of RDMSR with CPL = 3 generates a general-protection exception and not a VM exit.²
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see LMSW in Section 26.1.3).
- VM exits caused by execution of the INS and OUTS instructions (resulting either because the “unconditional I/O exiting” VM-execution control is 1 or because the “use I/O bitmaps control is 1”) have priority over the following faults:

1. These include faults generated by attempts to execute, in virtual-8086 mode, privileged instructions that are not recognized in that mode.
2. MOV DR is an exception to this rule; see Section 26.1.3.

- A general-protection fault due to the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) being unusable
- A general-protection fault due to an offset beyond the limit of the relevant segment
- An alignment-check exception
- Fault-like VM exits have priority over exceptions other than those mentioned above. For example, RDMSR of a non-existent MSR with CPL = 0 generates a VM exit and not a general-protection exception.

When Section 26.1.2 or Section 26.1.3 (below) identify an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that takes priority over a VM exit.

26.1.2 Instructions That Cause VM Exits Unconditionally

The following instructions cause VM exits when they are executed in VMX non-root operation: CPUID, GETSEC,¹ INVD, and XSETBV. This is also true of instructions introduced with VMX, which include: INVEPT, INVVPID, VMCALL,² VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMRESUME, VMXOFF, and VMXON.

26.1.3 Instructions That Cause VM Exits Conditionally

Certain instructions cause VM exits in VMX non-root operation depending on the setting of the VM-execution controls. The following instructions can cause “fault-like” VM exits based on the conditions described:³

- **CLTS.** The CLTS instruction causes a VM exit if the bits in position 3 (corresponding to CR0.TS) are set in both the CR0 guest/host mask and the CR0 read shadow.
- **ENCLS.** The ENCLS instruction causes a VM exit if the “enable ENCLS exiting” VM-execution control is 1 and one of the following is true:
 - The value of EAX is less than 63 and the corresponding bit in the ENCLS-exiting bitmap is 1 (see Section 25.6.16).
 - The value of EAX is greater than or equal to 63 and bit 63 in the ENCLS-exiting bitmap is 1.
- **ENCLV.** The ENCLV instruction causes a VM exit if the “enable ENCLV exiting” VM-execution control is 1 and one of the following is true:
 - The value of EAX is less than 63 and the corresponding bit in the ENCLV-exiting bitmap is 1 (see Section 25.6.17).
 - The value of EAX is greater than or equal to 63 and bit 63 in the ENCLV-exiting bitmap is 1.
- **ENQCMD, ENQCMLS.** The behavior of each of these instructions is determined by the setting of the “PASID translation” VM-execution control. If that control is 0, the instruction executes normally. If the control is 1, instruction behavior is modified and may cause a VM exit. See Section 26.5.8.
- **HLT.** The HLT instruction causes a VM exit if the “HLT exiting” VM-execution control is 1.
- **IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD.** The behavior of each of these instructions is determined by the settings of the “unconditional I/O exiting” and “use I/O bitmaps” VM-execution controls:
 - If both controls are 0, the instruction executes normally.

1. An execution of GETSEC in VMX non-root operation causes a VM exit if CR4.SMXE[Bit 14] = 1 regardless of the value of CPL or RAX. An execution of GETSEC causes an invalid-opcode exception (#UD) if CR4.SMXE[Bit 14] = 0.

2. Under the dual-monitor treatment of SMIs and SMM, executions of VMCALL cause SMM VM exits in VMX root operation outside SMM. See Section 32.15.2.

3. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 25.6.2.

- If the “unconditional I/O exiting” VM-execution control is 1 and the “use I/O bitmaps” VM-execution control is 0, the instruction causes a VM exit.
- If the “use I/O bitmaps” VM-execution control is 1, the instruction causes a VM exit if it attempts to access an I/O port corresponding to a bit set to 1 in the appropriate I/O bitmap (see Section 25.6.4). If an I/O operation “wraps around” the 16-bit I/O-port space (accesses ports FFFFH and 0000H), the I/O instruction causes a VM exit (the “unconditional I/O exiting” VM-execution control is ignored if the “use I/O bitmaps” VM-execution control is 1).

See Section 26.1.1 for information regarding the priority of VM exits relative to faults that may be caused by the INS and OUTS instructions.

- **INVLPG.** The INVLPG instruction causes a VM exit if the “INVLPG exiting” VM-execution control is 1.
- **INVPCID.** The INVPCID instruction causes a VM exit if the “INVLPG exiting” and “enable INVPCID” VM-execution controls are both 1.
- **LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR.** These instructions cause VM exits if the “descriptor-table exiting” VM-execution control is 1.
- **LMSW.** In general, the LMSW instruction causes a VM exit if it would write, for any bit set in the low 4 bits of the CR0 guest/host mask, a value different than the corresponding bit in the CR0 read shadow. LMSW never clears bit 0 of CR0 (CR0.PE); thus, LMSW causes a VM exit if either of the following are true:
 - The bits in position 0 (corresponding to CR0.PE) are set in both the CR0 guest/host mask and the source operand, and the bit in position 0 is clear in the CR0 read shadow.
 - For any bit position in the range 3:1, the bit in that position is set in the CR0 guest/host mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.
- **LOADIWKEY.** The LOADIWKEY instruction causes a VM exit if the “LOADIWKEY exiting” VM-execution control is 1.
- **MONITOR.** The MONITOR instruction causes a VM exit if the “MONITOR exiting” VM-execution control is 1.
- **MOV from CR3.** The MOV from CR3 instruction causes a VM exit if the “CR3-store exiting” VM-execution control is 1. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
- **MOV from CR8.** The MOV from CR8 instruction causes a VM exit if the “CR8-store exiting” VM-execution control is 1.
- **MOV to CR0.** The MOV to CR0 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 cannot cause a VM exit.)
- **MOV to CR3.** The MOV to CR3 instruction causes a VM exit unless the “CR3-load exiting” VM-execution control is 0 or the value of its source operand is equal to one of the CR3-target values specified in the VMCS. Only the first n CR3-target values are considered, where n is the CR3-target count. If the “CR3-load exiting” VM-execution control is 1 and the CR3-target count is 0, MOV to CR3 always causes a VM exit.

The first processors to support the virtual-machine extensions supported only the 1-setting of the “CR3-load exiting” VM-execution control. These processors always consult the CR3-target controls to determine whether an execution of MOV to CR3 causes a VM exit.

- **MOV to CR4.** The MOV to CR4 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow.
- **MOV to CR8.** The MOV to CR8 instruction causes a VM exit if the “CR8-load exiting” VM-execution control is 1.
- **MOV DR.** The MOV DR instruction causes a VM exit if the “MOV-DR exiting” VM-execution control is 1. Such VM exits represent an exception to the principles identified in Section 26.1.1 in that they take priority over the following: general-protection exceptions based on privilege level; and invalid-opcode exceptions that occur because CR4.DE=1 and the instruction specified access to DR4 or DR5.
- **MWAIT.** The MWAIT instruction causes a VM exit if the “MWAIT exiting” VM-execution control is 1. If this control is 0, the behavior of the MWAIT instruction may be modified (see Section 26.3).
- **PAUSE.** The behavior of each of this instruction depends on CPL and the settings of the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls:
 - CPL = 0.

- If the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls are both 0, the PAUSE instruction executes normally.
- If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit (the “PAUSE-loop exiting” VM-execution control is ignored if CPL = 0 and the “PAUSE exiting” VM-execution control is 1).
- If the “PAUSE exiting” VM-execution control is 0 and the “PAUSE-loop exiting” VM-execution control is 1, the following treatment applies.

The processor determines the amount of time between this execution of PAUSE and the previous execution of PAUSE at CPL 0. If this amount of time exceeds the value of the VM-execution control field PLE_Gap, the processor considers this execution to be the first execution of PAUSE in a loop. (It also does so for the first execution of PAUSE at CPL 0 after VM entry.)

Otherwise, the processor determines the amount of time since the most recent execution of PAUSE that was considered to be the first in a loop. If this amount of time exceeds the value of the VM-execution control field PLE_Window, a VM exit occurs.

For purposes of these computations, time is measured based on a counter that runs at the same rate as the timestamp counter (TSC).

— CPL > 0.

- If the “PAUSE exiting” VM-execution control is 0, the PAUSE instruction executes normally.
- If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit.

The “PAUSE-loop exiting” VM-execution control is ignored if CPL > 0.

- **PCONFIG.** The PCONFIG instruction causes a VM exit if the “enable PCONFIG” VM-execution control is 1 and one of the following is true:

- The value of EAX is less than 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1 (see Section 25.6.18).
- The value of EAX is greater than or equal to 63 and bit 63 in the PCONFIG-exiting bitmap is 1.

If the “enable PCONFIG” VM-execution control is 1 and neither of the previous items hold, the PCONFIG instruction executes normally.

- **RDMSR.** The RDMSR instruction causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in read bitmap for low MSRs is 1, where *n* is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

See Section 25.6.9 for details regarding how these bitmaps are identified.

- **RDPMC.** The RDPMC instruction causes a VM exit if the “RDPMC exiting” VM-execution control is 1.
- **RDRAND.** The RDRAND instruction causes a VM exit if the “RDRAND exiting” VM-execution control is 1.
- **RDSEED.** The RDSEED instruction causes a VM exit if the “RDSEED exiting” VM-execution control is 1.
- **RDTSC.** The RDTSC instruction causes a VM exit if the “RDTSC exiting” VM-execution control is 1.
- **RDTSCP.** The RDTSCP instruction causes a VM exit if the “RDTSC exiting” and “enable RDTSCP” VM-execution controls are both 1.
- **RSM.** The RSM instruction causes a VM exit if executed in system-management mode (SMM).¹
- **TPAUSE.** The TPAUSE instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.

1. Execution of the RSM instruction outside SMM causes an invalid-opcode exception regardless of whether the processor is in VMX operation. It also does so in VMX root operation in SMM; see Section 32.15.3.

- **UMWAIT.** The UMWAIT instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.
- **VMREAD.** The VMREAD instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMREAD bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 25.6.15 for details regarding how the VMREAD bitmap is identified.

If the VMREAD instruction does not cause a VM exit, it reads from the VMCS referenced by the VMCS link pointer. See Chapter 31, “VMREAD—Read Field from Virtual-Machine Control Structure” for details of the operation of the VMREAD instruction.

- **VMWRITE.** The VMWRITE instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMWRITE bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 25.6.15 for details regarding how the VMWRITE bitmap is identified.

If the VMWRITE instruction does not cause a VM exit, it writes to the VMCS referenced by the VMCS link pointer. See Chapter 31, “VMWRITE—Write Field to Virtual-Machine Control Structure” for details of the operation of the VMWRITE instruction.

- **WBINVD.** The WBINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WBNOINVD.** The WBNOINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WRMSR.** The WRMSR instruction causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit n in write bitmap for low MSRs is 1, where n is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit n in write bitmap for high MSRs is 1, where n is the value of ECX & 00001FFFH.

See Section 25.6.9 for details regarding how these bitmaps are identified.

- **XRSTORS.** The XRSTORS instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 25.6.21).
- **XSAVES.** The XSAVES instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 25.6.21).

26.2 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:¹

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 25.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally through the guest IDT. This use of the exception bitmap applies also to exceptions generated by the instructions INT1, INT3, INTO, BOUND, UD0, UD1, and UD2.²

1. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 25.6.2.

Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.

- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.
- **External interrupts.** An external interrupt causes a VM exit if the “external-interrupt exiting” VM-execution control is 1. (See Section 26.6 for an exception.) Otherwise, the processor handles the interrupt normally.¹ (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The processor does handle the interrupt and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the “NMI exiting” VM-execution control is 1. Otherwise, it is delivered using descriptor 2 of the IDT. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered through the IDT and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)
- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)
- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 26.4.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 32.15.2.²
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 26.5.1 for details of operation of the VMX-preemption timer.

Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the “NMI-window exiting” VM-execution control and lower priority events.

These VM exits wake a logical processor from the same inactive states as would a non-maskable interrupt. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

- **Bus locks.** Assertion of a bus lock (see Section 9.1.2) causes a VM exit if the “VMM bus-lock detection” VM-execution control is 1. Such a VM exit is trap-like because it is generated after execution of an instruction that asserts a bus lock. The VM exit thus does not prevent assertion of the bus lock. These VM exits take priority over system-management interrupts (SMIs), INIT signals, and lower priority events.

2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

1. Normal handling usually means delivery through the IDT, but it could also mean treatment of the interrupt as a user-interrupt notification.

2. Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 32.14.1.

- **Instruction timeout.** If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within the amount of time specified by the instruction-timeout control VM-execution control field (see Section 25.6.25).

In addition, there are controls that cause VM exits based on the readiness of guest software to receive interrupts:

- If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if RFLAGS.IF = 1 and there is no blocking of events by STI or by MOV SS (see Table 25-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 27.7.5).

Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.

- If the “NMI-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no blocking of events by MOV SS and no blocking of events by STI (see Table 25-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 27.7.6).

VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

26.3 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation. Some of these changes are determined by the settings of certain VM-execution control fields. The following items detail such changes:¹

- **CLTS.** Behavior of the CLTS instruction is determined by the bits in position 3 (corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:
 - If bit 3 in the CR0 guest/host mask is 0, CLTS clears CR0.TS normally (the value of bit 3 in the CR0 read shadow is irrelevant in this case), unless CR0.TS is fixed to 1 in VMX operation (see Section 24.8), in which case CLTS causes a general-protection exception.
 - If bit 3 in the CR0 guest/host mask is 1 and bit 3 in the CR0 read shadow is 0, CLTS completes but does not change the contents of CR0.TS.
 - If the bits in position 3 in the CR0 guest/host mask and the CR0 read shadow are both 1, CLTS causes a VM exit.
- **ENQCMD, ENQCMDs.** Each of these instructions performs a 64-byte enqueue store that includes a PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMDs, the PASID is normally read from memory.

The behavior of each of these instructions (and in particular the PASID value used for the enqueue store) is determined by the setting of the “PASID translation” VM-execution control:

- If the “PASID translation” VM-execution control is 0, the instruction operates normally.
- If the “PASID translation” VM-execution control is 1, the PASID value used for the enqueue store is determined by the PASID-translation process described in Section 26.5.8. (Note the PASID translation may result in a VM exit, in which case the enqueue store is not performed.)

1. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 25.6.2.

An execution of ENQCMD or ENQCMLS performs PASID translation only after checking for conditions that may result in general-protection exception (the check of IA32_PASID.Valid for ENQCMD; the privilege-level check for ENQCMLS), after loading the instruction's source operand from memory, and thus after any faults or VM exits that the loading may cause (e.g., page faults or EPT violations). PASID translation occurs before the actual enqueue store and thus before any faults or VM exits that it may cause.

- **INVPCID.** Behavior of the INVPCID instruction is determined first by the setting of the “enable INVPCID” VM-execution control:
 - If the “enable INVPCID” VM-execution control is 0, INVPCID causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable INVPCID” VM-execution control is 1, treatment is based on the setting of the “INVLPG exiting” VM-execution control:
 - If the “INVLPG exiting” VM-execution control is 0, INVPCID operates normally.
 - If the “INVLPG exiting” VM-execution control is 1, INVPCID causes a VM exit.
- **IRET.** Behavior of IRET with regard to NMI blocking (see Table 25-3) is determined by the settings of the “NMI exiting” and “virtual NMIs” VM-execution controls:
 - If the “NMI exiting” VM-execution control is 0, IRET operates normally and unblocks NMIs. (If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 27.2.1.1.)
 - If the “NMI exiting” VM-execution control is 1, IRET does not affect blocking of NMIs. If, in addition, the “virtual NMIs” VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, IRET removes any virtual-NMI blocking.

The unblocking of NMIs or virtual NMIs specified above occurs even if IRET causes a fault.

- **LMSW.** Outside of VMX non-root operation, LMSW loads its source operand into CR0[3:0], but it does not clear CR0.PE if that bit is set. In VMX non-root operation, an execution of LMSW that does not cause a VM exit (see Section 26.1.3) leaves unmodified any bit in CR0[3:0] corresponding to a bit set in the CR0 guest/host mask. An attempt to set any other bit in CR0[3:0] to a value not supported in VMX operation (see Section 24.8) causes a general-protection exception. Attempts to clear CR0.PE are ignored without fault.
- **MOV from CR0.** The behavior of MOV from CR0 is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow. Depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.
- **MOV from CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV from CR3 does not cause a VM exit (see Section 26.1.3), the value loaded from CR3 is a guest-physical address; see Section 29.3.1.
- **MOV from CR4.** The behavior of MOV from CR4 is determined by the CR4 guest/host mask and the CR4 read shadow. For each position corresponding to a bit clear in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR4. For each position corresponding to a bit set in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR4 read shadow. Thus, if every bit is cleared in the CR4 guest/host mask, MOV from CR4 reads normally from CR4; if every bit is set in the CR4 guest/host mask, MOV from CR4 returns the value of the CR4 read shadow. Depending on the contents of the CR4 guest/host mask and the CR4 read shadow, bits may be set in the destination that would never be set when reading directly from CR4.
- **MOV from CR8.** If the MOV from CR8 instruction does not cause a VM exit (see Section 26.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 30.3.
- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 26.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. Treatment of attempts to modify other bits in CR0 depends on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 to a value not supported in VMX operation (see Section 24.8).

- If the control is 1, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 other than bit 0 (PE) or bit 31 (PG) to a value not supported in VMX operation. It remains the case, however, that MOV to CR0 causes a general-protection exception if it would result in CR0.PE = 0 and CR0.PG = 1 or if it would result in CR0.PG = 1, CR4.PAE = 0, and IA32_EFER.LME = 1.
- **MOV to CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV to CR3 does not cause a VM exit (see Section 26.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 29.3.1.
 - If PAE paging is not being used, the instruction does not use the guest-physical address to access memory and it does not cause it to be translated through EPT.¹
 - If PAE paging is being used, the instruction translates the guest-physical address through EPT and uses the result to load the four (4) page-directory-pointer-table entries (PDPTes). The instruction does not use the guest-physical addresses the PDPTes to access memory and it does not cause them to be translated through EPT.
- **MOV to CR4.** An execution of MOV to CR4 that does not cause a VM exit (see Section 26.1.3) leaves unmodified any bit in CR4 corresponding to a bit set in the CR4 guest/host mask. Such an execution causes a general-protection exception if it attempts to set any bit in CR4 (not corresponding to a bit set in the CR4 guest/host mask) to a value not supported in VMX operation (see Section 24.8).
- **MOV to CR8.** If the MOV to CR8 instruction does not cause a VM exit (see Section 26.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 30.3.
- **MWAIT.** Behavior of the MWAIT instruction (which always causes an invalid-opcode exception—#UD—if CPL > 0) is determined by the setting of the “MWAIT exiting” VM-execution control:
 - If the “MWAIT exiting” VM-execution control is 1, MWAIT causes a VM exit.
 - If the “MWAIT exiting” VM-execution control is 0, MWAIT operates normally if one of the following are true: (1) ECX[0] is 0; (2) RFLAGS.IF = 1; or both of the following are true: (a) the “interrupt-window exiting” VM-execution control is 0; and (b) the logical processor has not recognized a pending virtual interrupt (see Section 29.2.1).
 - If the “MWAIT exiting” VM-execution control is 0, ECX[0] = 1, and RFLAGS.IF = 0, MWAIT does not cause the processor to enter an implementation-dependent optimized state if either the “interrupt-window exiting” VM-execution control is 1 or the logical processor has recognized a pending virtual interrupt; instead, control passes to the instruction following the MWAIT instruction.
- **PCONFIG.** Behavior of the PCONFIG instruction is determined by the setting of the “enable PCONFIG” VM-execution control:
 - If the “enable PCONFIG” VM-execution control is 0, PCONFIG causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable PCONFIG” VM-execution control is 1, PCONFIG may cause a VM exit as specified in Section 26.1.3; if it does not cause such a VM exit, it operates normally.
- **RDMSR.** Section 26.1.3 identifies when executions of the RDMSR instruction cause VM exits. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
 - If ECX contains 10H (indicating the IA32_TIME_STAMP_COUNTER MSR), the value returned by the instruction is determined by the setting of the “use TSC offsetting” VM-execution control:
 - If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_TIME_STAMP_COUNTER MSR.
 - If the control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDMSR loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

- If the control is 1, RDMSR first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

The 1-setting of the “use TSC-offsetting” VM-execution control does not affect executions of RDMSR if ECX contains 6E0H (indicating the IA32_TSC_DEADLINE MSR). Such executions return the APIC-timer deadline relative to the actual timestamp counter without regard to the TSC offset.

- If ECX is in the range 800H–8FFH (indicating an APIC MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 30.5.
- **RDPID.** Behavior of the RDPID instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
 - If the “enable RDTSCP” VM-execution control is 0, RDPID causes an invalid-opcode exception (#UD).
 - If the “enable RDTSCP” VM-execution control is 1, RDPID operates normally.
- **RDTSC.** Behavior of the RDTSC instruction is determined by the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTSC operates normally.
 - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTSC loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTSC first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.
 - If the “RDTSC exiting” VM-execution control is 1, RDTSC causes a VM exit.
- **RDTSCP.** Behavior of the RDTSCP instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
 - If the “enable RDTSCP” VM-execution control is 0, RDTSCP causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable RDTSCP” VM-execution control is 1, treatment is based on the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTSCP operates normally.
 - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTSCP loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTSCP first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

In either case, RDTSCP also loads ECX with the value of bits 31:0 of the IA32_TSC_AUX MSR.

 - If the “RDTSC exiting” VM-execution control is 1, RDTSCP causes a VM exit.- **SMSW.** The behavior of SMSW is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, SMSW reads normally from CR0; if every bit is set in the CR0 guest/host mask, SMSW returns the value of the CR0 read shadow.

Note the following: (1) for any memory destination or for a 16-bit register destination, only the low 16 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:16 of a register destination are left

unchanged); (2) for a 32-bit register destination, only the low 32 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:32 of the destination are cleared); and (3) depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.

- **TPAUSE.** Behavior of the TPAUSE instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, TPAUSE causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:
 - If the “RDTSC exiting” VM-execution control is 0, the instruction delays for an amount of time called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).
If IA32_UWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UWAIT_CONTROL,FFFFFFFFCH).
The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:
 - If either control is 0, the physical delay is the virtual delay.
 - If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
 - If the “RDTSC exiting” VM-execution control is 1, TPAUSE causes a VM exit.
- **UMONITOR.** Behavior of the UMONITOR instruction is determined by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, UMONITOR causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, UMONITOR operates normally.
- **UWAIT.** Behavior of the UWAIT instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, UWAIT causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:
 - If the “RDTSC exiting” VM-execution control is 0, and if the instruction causes a delay, the amount of time delayed is called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).
If IA32_UWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UWAIT_CONTROL,FFFFFFFFCH).
The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:
 - If either control is 0, the physical delay is the virtual delay.
 - If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
 - If the “RDTSC exiting” VM-execution control is 1, UWAIT causes a VM exit.

- **WRMSR.** Section 26.1.3 identifies when executions of the WRMSR instruction cause VM exits. If such an execution neither a fault due to CPL > 0 nor a VM exit, the instruction's behavior may be modified for certain values of ECX:
 - If ECX contains 79H (indicating IA32_BIOS_UPDT_TRIG MSR), no microcode update is loaded, and control passes to the next instruction. This implies that microcode updates cannot be loaded in VMX non-root operation.
 - On processors that support Intel PT but which do not allow it to be used in VMX operation, if ECX contains 570H (indicating the IA32_RTIT_CTL MSR), the instruction causes a general-protection exception.¹
 - If ECX contains 808H (indicating the TPR MSR), 80BH (the EOI MSR), 830H (the ICR MSR), or 83FH (the self-IPI MSR), instruction behavior may be modified if the "virtualize x2APIC mode" VM-execution control is 1; see Section 30.5.
- **XRSTORS.** Behavior of the XRSTORS instruction is determined first by the setting of the "enable XSAVES/XRSTORS" VM-execution control:
 - If the "enable XSAVES/XRSTORS" VM-execution control is 0, XRSTORS causes an invalid-opcode exception (#UD).
 - If the "enable XSAVES/XRSTORS" VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 25.6.21):
 - XRSTORS causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XRSTORS operates normally.
- **XSAVES.** Behavior of the XSAVES instruction is determined first by the setting of the "enable XSAVES/XRSTORS" VM-execution control:
 - If the "enable XSAVES/XRSTORS" VM-execution control is 0, XSAVES causes an invalid-opcode exception (#UD).
 - If the "enable XSAVES/XRSTORS" VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 25.6.21):
 - XSAVES causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XSAVES operates normally.

26.4 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking, task switches, and user interrupts differ in VMX non-root operation as described in the following sections.

26.4.1 Event Blocking

Event blocking is modified in VMX non-root operation as follows:

- If the "external-interrupt exiting" VM-execution control is 1, RFLAGS.IF does not control the blocking of external interrupts. In this case, an external interrupt that is not blocked for other reasons causes a VM exit (even if RFLAGS.IF = 0).
- If the "external-interrupt exiting" VM-execution control is 1, external interrupts may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).
- If the "NMI exiting" VM-execution control is 1, non-maskable interrupts (NMIs) may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).

1. Software should read the VMX capability MSR IA32_VMX_MISC to determine whether the processor allows Intel PT to be used in VMX operation (see Appendix A.6).

26.4.2 Treatment of Task Switches

Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. However, the following checks are performed (in the order indicated), possibly resulting in a fault, before there is any possibility of a VM exit due to task switch:

1. If a task gate is being used, appropriate checks are made on its P bit and on the proper values of the relevant privilege fields. The following cases detail the privilege checks performed:
 - a. If CALL, INT n , INT1, INT3, INTO, or JMP accesses a task gate in IA-32e mode, a general-protection exception occurs.
 - b. If CALL, INT n , INT3, INTO, or JMP accesses a task gate outside IA-32e mode, privilege-levels checks are performed on the task gate but, if they pass, privilege levels are not checked on the referenced task-state segment (TSS) descriptor.
 - c. If CALL or JMP accesses a TSS descriptor directly in IA-32e mode, a general-protection exception occurs.
 - d. If CALL or JMP accesses a TSS descriptor directly outside IA-32e mode, privilege levels are checked on the TSS descriptor.
 - e. If a non-maskable interrupt (NMI), an exception, or an external interrupt accesses a task gate in the IDT in IA-32e mode, a general-protection exception occurs.
 - f. If a non-maskable interrupt (NMI), an exception other than breakpoint exceptions (#BP) and overflow exceptions (#OF), or an external interrupt accesses a task gate in the IDT outside IA-32e mode, no privilege checks are performed.
 - g. If IRET is executed with RFLAGS.NT = 1 in IA-32e mode, a general-protection exception occurs.
 - h. If IRET is executed with RFLAGS.NT = 1 outside IA-32e mode, a TSS descriptor is accessed directly and no privilege checks are made.
2. Checks are made on the new TSS selector (for example, that is within GDT limits).
3. The new TSS descriptor is read. (A page fault results if a relevant GDT page is not present).
4. The TSS descriptor is checked for proper values of type (depends on type of task switch), P bit, S bit, and limit.

Only if checks 1–4 all pass (do not generate faults) might a VM exit occur. However, the ordering between a VM exit due to a task switch and a page fault resulting from accessing the old TSS or the new TSS is implementation-specific. Some processors may generate a page fault (instead of a VM exit due to a task switch) if accessing either TSS would cause a page fault. Other processors may generate a VM exit due to a task switch even if accessing either TSS would cause a page fault.

If an attempt at a task switch through a task gate in the IDT causes an exception (before generating a VM exit due to the task switch) and that exception causes a VM exit, information about the event whose delivery that accessed the task gate is recorded in the IDT-vectoring information fields and information about the exception that caused the VM exit is recorded in the VM-exit interruption-information fields. See Section 28.2. The fact that a task gate was being accessed is not recorded in the VMCS.

If an attempt at a task switch through a task gate in the IDT causes VM exit due to the task switch, information about the event whose delivery accessed the task gate is recorded in the IDT-vectoring fields of the VMCS. Since the cause of such a VM exit is a task switch and not an interruption, the valid bit for the VM-exit interruption information field is 0. See Section 28.2.

26.5 FEATURES SPECIFIC TO VMX NON-ROOT OPERATION

Some VM-execution controls support features that are specific to VMX non-root operation. These are the VMX-preemption timer (Section 26.5.1) and the monitor trap flag (Section 26.5.2), translation of guest-physical addresses (Section 26.5.3 and Section 26.5.4), APIC virtualization (Section 26.5.5), VM functions (Section 26.5.6), and virtualization exceptions (Section 26.5.7).

26.5.1 VMX-Preemption Timer

If the last VM entry was performed with the 1-setting of “activate VMX-preemption timer” VM-execution control, the **VMX-preemption timer** counts down (from the value loaded by VM entry; see Section 27.7.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 26.2).

The VMX-preemption timer counts down at rate proportional to that of the timestamp counter (TSC). Specifically, the timer counts down by 1 every time bit X in the TSC changes due to a TSC increment. The value of X is in the range 0–31 and can be determined by consulting the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VMX-preemption timer operates in the C-states C0, C1, and C2; it also operates in the shutdown and wait-for-SIPI states. If the timer counts down to zero in any state other than the wait-for SIPI state, the logical processor transitions to the C0 C-state and causes a VM exit; the timer does not cause a VM exit if it counts down to zero in the wait-for-SIPI state. The timer is not decremented in C-states deeper than C2.

Treatment of the timer in the case of system management interrupts (SMIs) and system-management mode (SMM) depends on whether the treatment of SMIs and SMM:

- If the default treatment of SMIs and SMM (see Section 32.14) is active, the VMX-preemption timer counts across an SMI to VMX non-root operation, subsequent execution in SMM, and the return from SMM via the RSM instruction. However, the timer can cause a VM exit only from VMX non-root operation. If the timer expires during SMI, in SMM, or during RSM, a timer-induced VM exit occurs immediately after RSM with its normal priority unless it is blocked based on activity state (Section 26.2).
- If the dual-monitor treatment of SMIs and SMM (see Section 32.15) is active, transitions into and out of SMM are VM exits and VM entries, respectively. The treatment of the VMX-preemption timer by those transitions is mostly the same as for ordinary VM exits and VM entries; Section 32.15.2 and Section 32.15.4 detail some differences.

26.5.2 Monitor Trap Flag

The **monitor trap flag** is a debugging feature that causes VM exits to occur on certain instruction boundaries in VMX non-root operation. Such VM exits are called **MTF VM exits**. An MTF VM exit may occur on an instruction boundary in VMX non-root operation as follows:

- If the “monitor trap flag” VM-execution control is 1 and VM entry is injecting a vectored event (see Section 27.6.1), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry.
- If VM entry is injecting a pending MTF VM exit (see Section 27.6.2), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0.
- If the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and a pending event (e.g., debug exception or interrupt) is delivered before an instruction can execute, an MTF VM exit is pending on the instruction boundary following delivery of the event (or any nested exception).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is a REP-prefixed string instruction:
 - If the first iteration of the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).
 - If the first iteration of the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary after that iteration.
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is the XBEGIN instruction. In this case, an MTF VM exit is pending at the fallback instruction address of the XBEGIN instruction. This behavior applies regardless of whether advanced debugging of RTM transactional regions has been enabled (see Section 16.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is neither a REP-prefixed string instruction or the XBEGIN instruction:

- If the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).¹
- If the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary following execution of that instruction. If the instruction is INT1, INT3, or INTO, this boundary follows delivery of any software exception. If the instruction is INT *n*, this boundary follows delivery of a software interrupt. If the instruction is HLT, the MTF VM exit will be from the HLT activity state.

No MTF VM exit occurs if another VM exit occurs before reaching the instruction boundary on which an MTF VM exit would be pending (e.g., due to an exception or triple fault).

An MTF VM exit occurs on the instruction boundary on which it is pending unless a higher priority event takes precedence or the MTF VM exit is blocked due to the activity state:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over MTF VM exits. MTF VM exits take priority over debug-trap exceptions and lower priority events.
- No MTF VM exit occurs if the processor is in either the shutdown activity state or wait-for-SIPI activity state. If a non-maskable interrupt subsequently takes the logical processor out of the shutdown activity state without causing a VM exit, an MTF VM exit is pending after delivery of that interrupt.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 40.2 for details.

26.5.3 Translation of Guest-Physical Addresses Using EPT

The extended page-table mechanism (EPT) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain physical addresses are treated as guest-physical addresses and are not used to access memory directly. Instead, guest-physical addresses are translated by traversing a set of EPT paging structures to produce physical addresses that are used to access memory.

Details of the EPT mechanism are given in Section 29.3.

26.5.4 Translation of Guest-Physical Addresses Used by Intel Processor Trace

As described in Chapter 33, Intel® Processor Trace (Intel PT) captures information about software execution using dedicated hardware facilities.

Intel PT can be configured so that the trace output is written to memory using physical addresses. For example, when the ToPA (table of physical addresses) output mechanism is used, the IA32_RTIT_OUTPUT_BASE MSR contains the physical address of the base of the current ToPA. Each entry in that table contains the physical address of an output region in memory. When an output region becomes full, the ToPA output mechanism directs subsequent trace output to the next output region as indicated in the ToPA.

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the logical processor treats the addresses used by Intel PT (the output addresses as well as those used to discover the output addresses) as guest-physical addresses, translating to physical addresses using EPT before trace output is written to memory.

Translating these addresses through EPT implies that the trace-output mechanism may cause EPT violations and VM exits; details are provided in Section 26.5.4.1. Section 26.5.4.2 describes a mechanism that ensures that these VM exits do not cause loss of trace data.

26.5.4.1 Guest-Physical Address Translation for Intel PT: Details

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses and translated using EPT. These addresses include the addresses of the output regions as well as the addresses of the ToPA entries that contain the output-region addresses.

1. This item includes the cases of an invalid opcode exception—#UD—generated by the UD0, UD1, and UD2 instructions and a BOUND-range exceeded exception—#BR—generated by the BOUND instruction.

Translation of accesses by the trace-output process may result in EPT violations or EPT misconfigurations (Section 29.3.3), resulting in VM exits. EPT violations resulting for the trace-output process always cause VM exits and are never converted to virtualization exceptions (Section 26.5.7.1).

If no EPT violation or EPT misconfiguration occurs and if page-modification logging (Section 29.3.6) is enabled, the address of an output region may be added to the page-modification log. If the log is full, a page-modification log-full event occurs, resulting in a VM exit.

If the “virtualize APIC accesses” VM-execution control is 1, a guest-physical address used by the trace-output process may be translated to an address on the APIC-access page. In this case, the access by the trace-output process causes an APIC-access VM exit as discussed in Section 30.4.6.1.

26.5.4.2 Trace-Address Pre-Translation (TAPT)

Because it buffers trace data produced by Intel PT before it is written to memory, the processor ensures that buffered data is not lost when a VM exit disables Intel PT. Specifically, the processor ensures that there is sufficient space left in the current output page for the buffered data. If this were not done, buffered trace data could be lost and the resulting trace corrupted.

To prevent the loss of buffered trace data, the processor uses a mechanism called **trace-address pre-translation (TAPT)**. With TAPT, the processor translates using EPT the guest-physical address of the current output region before that address would be used to write buffered trace data to memory.

Because of TAPT, no translation (and thus no EPT violation) occurs at the time output is written to memory; the writes to memory use translations that were cached as part of TAPT. (The details given in Section 26.5.4.1 apply to TAPT.) TAPT ensures that, if a write to the output region would cause an EPT violation, the resulting VM exit is delivered at the time of TAPT, before the region would be used. This allows software to resolve the EPT violation at that time and ensures that, when it is necessary to write buffered trace data to memory, that data will not be lost due to an EPT violation.

TAPT (and resulting VM exits) may occur at any of the following times:

- When software in VMX non-root operation enables tracing by loading the IA32_RTIT_CTL MSR to set the TraceEn bit, using the WRMSR instruction or the XRSTORS instruction.
Any VM exit resulting from TAPT in this case is trap-like: the WRMSR or XRSTORS completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).
- At an instruction boundary when one output region becomes full and Intel PT transitions to the next output region.
VM exits resulting from TAPT in this case take priority over any pending debug exceptions. Such a VM exit will save information about such exceptions in the guest-state area of the VMCS.
- As part of a VM entry that enables Intel PT. See Section 27.5 for details.

TAPT may translate not only the guest-physical address of the current output region but those of subsequent output regions as well. (Doing so may provide better protection of trace data.) This implies that any VM exits resulting from TAPT may result from the translation of output-region addresses other than that of the current output region.

26.5.5 APIC Virtualization

APIC virtualization is a collection of features that can be used to support the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC). When APIC virtualization is enabled, the processor emulates many accesses to the APIC, tracks the state of the virtual APIC, and delivers virtual interrupts — all in VMX non-root operation without a VM exit.

Details of the APIC virtualization are given in Chapter 30.

26.5.6 VM Functions

A **VM function** is an operation provided by the processor that can be invoked from VMX non-root operation without a VM exit. VM functions are enabled and configured by the settings of different fields in the VMCS. Software in VMX non-root operation invokes a VM function with the **VMFUNC** instruction; the value of EAX selects the specific VM function being invoked.

Section 26.5.6.1 explains how VM functions are enabled. Section 26.5.6.2 specifies the behavior of the VMFUNC instruction. Section 26.5.6.3 describes a specific VM function called **EPTP switching**.

26.5.6.1 Enabling VM Functions

Software enables VM functions generally by setting the “enable VM functions” VM-execution control. A specific VM function is enabled by setting the corresponding VM-function control.

Suppose, for example, that software wants to enable EPTP switching (VM function 0; see Section 25.6.14). To do so, it must set the “activate secondary controls” VM-execution control (bit 31 of the primary processor-based VM-execution controls), the “enable VM functions” VM-execution control (bit 13 of the secondary processor-based VM-execution controls) and the “EPTP switching” VM-function control (bit 0 of the VM-function controls).

26.5.6.2 General Operation of the VMFUNC Instruction

The VMFUNC instruction causes an invalid-opcode exception (#UD) if the “enable VM functions” VM-execution controls is 0¹ or the value of EAX is greater than 63 (only VM functions 0–63 can be enable). Otherwise, the instruction causes a VM exit if the bit at position EAX is 0 in the VM-function controls (the selected VM function is not enabled). If such a VM exit occurs, the basic exit reason used is 59 (3BH), indicating “VMFUNC”, and the length of the VMFUNC instruction is saved into the VM-exit instruction-length field. If the instruction causes neither an invalid-opcode exception nor a VM exit due to a disabled VM function, it performs the functionality of the VM function specified by the value in EAX.

Individual VM functions may perform additional fault checking (e.g., one might cause a general-protection exception if CPL > 0). In addition, specific VM functions may include checks that might result in a VM exit. If such a VM exit occurs, VM-exit information is saved as described in the previous paragraph. The specification of a VM function may indicate that additional VM-exit information is provided.

The specific behavior of the EPTP-switching VM function (including checks that result in VM exits) is given in Section 26.5.6.3.

26.5.6.3 EPTP Switching

EPTP switching is VM function 0. This VM function allows software in VMX non-root operation to load a new value for the EPT pointer (EPTP), thereby establishing a different EPT paging-structure hierarchy (see Section 29.3 for details of the operation of EPT). Software is limited to selecting from a list of potential EPTP values configured in advance by software in VMX root operation.

Specifically, the value of ECX is used to select an entry from the EPTP list, the 4-KByte structure referenced by the EPTP-list address (see Section 25.6.14; because this structure contains 512 8-Byte entries, VMFUNC causes a VM exit if ECX ≥ 512). If the selected entry is a valid EPTP value (it would not cause VM entry to fail; see Section 27.2.1.1), it is stored in the EPTP field of the current VMCS and is used for subsequent accesses using guest-physical addresses. The following pseudocode provides details:

```
IF ECX ≥ 512
  THEN VM exit;
ELSE
  tent_EPTP := 8 bytes from EPTP-list address + 8 * ECX;
  IF tent_EPTP is not a valid EPTP value (would cause VM entry to fail if in EPTP)
    THEN VM exit;
```

1. “Enable VM functions” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable VM functions” VM-execution control were 0. See Section 25.6.2.

```

ELSE
    write tent_EPTP to the EPTP field in the current VMCS;
    use tent_EPTP as the new EPTP value for address translation;
    IF processor supports the 1-setting of the "EPT-violation #VE" VM-execution control
        THEN
            write ECX[15:0] to EPTP-index field in current VMCS;
            use ECX[15:0] as EPTP index for subsequent EPT-violation virtualization exceptions (see Section 26.5.7.2);
        FI;
    FI;
FI;

```

Execution of the EPTP-switching VM function does not modify the state of any registers; no flags are modified.

If the "Intel PT uses guest physical addresses" VM-execution control is 1 and IA32_RTIT_CTL.TraceEn = 1, any execution of the EPTP-switching VM function causes a VM exit.¹

As noted in Section 26.5.6.2, an execution of the EPTP-switching VM function that causes a VM exit (as specified above), uses the basic exit reason 59, indicating "VMFUNC". The length of the VMFUNC instruction is saved into the VM-exit instruction-length field. No additional VM-exit information is provided.

An execution of VMFUNC loads EPTP from the EPTP list (and thus does not cause a fault or VM exit) is called an **EPTP-switching VMFUNC**. After an EPTP-switching VMFUNC, control passes to the next instruction. The logical processor starts creating and using guest-physical and combined mappings associated with the new value of bits 51:12 of EPTP; the combined mappings created and used are associated with the current VPID and PCID (these are not changed by VMFUNC).² If the "enable VPID" VM-execution control is 0, an EPTP-switching VMFUNC invalidates combined mappings associated with VPID 0000H (for all PCIDs and for all EP4TA values, where EP4TA is the value of bits 51:12 of EPTP).

Because an EPTP-switching VMFUNC may change the translation of guest-physical addresses, it may affect use of the guest-physical address in CR3. The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration due to the translation of that guest-physical address through the new EPT paging structures. The following items provide details that apply if CR0.PG = 1:

- If 32-bit paging or 4-level paging³ is in use (either CR4.PAE = 0 or IA32_EFER.LMA = 1), the next memory access with a linear address uses the translation of the guest-physical address in CR3 through the new EPT paging structures. As a result, this access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.
- If PAE paging is in use (CR4.PAE = 1 and IA32_EFER.LMA = 0), an EPTP-switching VMFUNC **does not** load the four page-directory-pointer-table entries (PDPTes) from the guest-physical address in CR3. The logical processor continues to use the four guest-physical addresses already present in the PDPTes. The guest-physical address in CR3 is not translated through the new EPT paging structures (until some operation that would load the PDPTes).

The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during the translation of a guest-physical address in any of the PDPTes. A subsequent memory access with a linear address uses the translation of the guest-physical address in the appropriate PDPTE through the new EPT paging structures. As a result, such an access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.

If an EPTP-switching VMFUNC establishes an EPTP value that enables accessed and dirty flags for EPT (by setting bit 6), subsequent memory accesses may fail to set those flags as specified if there has been no appropriate execution of INVEPT since the last use of an EPTP value that does not enable accessed and dirty flags for EPT (because bit 6 is clear) and that is identical to the new value on bits 51:12.

If the processor supports the 1-setting of the "EPT-violation #VE" VM-execution control, an EPTP-switching VMFUNC loads the value in ECX[15:0] into to EPTP-index field in current VMCS. Subsequent EPT-violation virtualization exceptions will save this value into the virtualization-exception information area (see Section 26.5.7.2).

1. Such a VM exit ensures the proper recording of trace data that might otherwise be lost during the change of EPT paging-structure hierarchy. Software handling the VM exit can change emulate the VM function and then resume the guest.
2. If the "enable VPID" VM-execution control is 0, the current VPID is 0000H; if CR4.PCIDE = 0, the current PCID is 000H.
3. Earlier versions of this manual used the term "IA-32e paging" to identify 4-level paging.

26.5.7 Virtualization Exceptions

A **virtualization exception** is a new processor exception. It uses vector 20 and is abbreviated #VE.

A virtualization exception can occur only in VMX non-root operation. Virtualization exceptions occur only with certain settings of certain VM-execution controls. Generally, these settings imply that certain conditions that would normally cause VM exits instead cause virtualization exceptions

In particular, the 1-setting of the “EPT-violation #VE” VM-execution control causes some EPT violations to generate virtualization exceptions instead of VM exits. Section 26.5.7.1 provides the details of how the processor determines whether an EPT violation causes a virtualization exception or a VM exit.

When the processor encounters a virtualization exception, it saves information about the exception to the virtualization-exception information area; see Section 26.5.7.2.

After saving virtualization-exception information, the processor delivers a virtualization exception as it would any other exception; see Section 26.5.7.3 for details.

26.5.7.1 Convertible EPT Violations

If the “EPT-violation #VE” VM-execution control is 0 (e.g., on processors that do not support this feature), EPT violations always cause VM exits. If instead the control is 1, certain EPT violations may be converted to cause virtualization exceptions instead; such EPT violations are **convertible**.

The values of certain EPT paging-structure entries determine which EPT violations are convertible. Specifically, bit 63 of certain EPT paging-structure entries may be defined to mean **suppress #VE**:

- If bits 2:0 of an EPT paging-structure entry are all 0, the entry is not **present**.¹ If the processor encounters such an entry while translating a guest-physical address, it causes an EPT violation. The EPT violation is convertible if and only if bit 63 of the entry is 0.
- If an EPT paging-structure entry is present, the following cases apply:
 - If the value of the EPT paging-structure entry is not supported, the entry is **misconfigured**. If the processor encounters such an entry while translating a guest-physical address, it causes an EPT misconfiguration (not an EPT violation). EPT misconfigurations always cause VM exits.
 - If the value of the EPT paging-structure entry is supported, the following cases apply:
 - If bit 7 of the entry is 1, or if the entry is an EPT PTE, the entry maps a page. If the processor uses such an entry to translate a guest-physical address, and if an access to that address causes an EPT violation, the EPT violation is convertible if and only if bit 63 of the entry is 0.
 - If bit 7 of the entry is 0 and the entry is not an EPT PTE, the entry references another EPT paging structure. The processor does not use the value of bit 63 of the entry to determine whether any subsequent EPT violation is convertible.

If an access to a guest-physical address causes an EPT violation, bit 63 of exactly one of the EPT paging-structure entries used to translate that address is used to determine whether the EPT violation is convertible: either a entry that is not present (if the guest-physical address does not translate to a physical address) or an entry that maps a page (if it does).

A convertible EPT violation instead causes a virtualization exception if the following all hold:

- CR0.PE = 1;
- the logical processor is not in the process of delivering an event through the IDT;
- the EPT violation does not result from the output process of Intel Processor Trace (Section 26.5.4); and
- the 32 bits at offset 4 in the virtualization-exception information area are all 0.

Delivery of virtualization exceptions writes the value FFFFFFFFH to offset 4 in the virtualization-exception information area (see Section 26.5.7.2). Thus, once a virtualization exception occurs, another can occur only if software clears this field.

1. If the “mode-based execute control for EPT” VM-execution control is 1, an EPT paging-structure entry is present if any of bits 2:0 or bit 10 is 1.

26.5.7.2 Virtualization-Exception Information

Virtualization exceptions save data into the virtualization-exception information area (see Section 25.6.20). Table 26-1 enumerates the data saved and the format of the area.

Table 26-1. Format of the Virtualization-Exception Information Area

Byte Offset	Contents
0	The 32-bit value that would have been saved into the VMCS as an exit reason had a VM exit occurred instead of the virtualization exception. For EPT violations, this value is 48 (00000030H)
4	FFFFFFFFH
8	The 64-bit value that would have been saved into the VMCS as an exit qualification had a VM exit occurred instead of the virtualization exception
16	The 64-bit value that would have been saved into the VMCS as a guest-linear address had a VM exit occurred instead of the virtualization exception
24	The 64-bit value that would have been saved into the VMCS as a guest-physical address had a VM exit occurred instead of the virtualization exception
32	The current 16-bit value of the EPTP index VM-execution control (see Section 25.6.20 and Section 26.5.6.3)

A VMM may allow guest software to access the virtualization-exception information area. If it does, the guest software may modify that memory (e.g., to clear the 32-bit value at offset 4; see Section 26.5.7.1). (This is an exception to the general requirement given in Section 25.11.4.)

26.5.7.3 Delivery of Virtualization Exceptions

After saving virtualization-exception information, the processor treats a virtualization exception as it does other exceptions:

- If bit 20 (#VE) is 1 in the exception bitmap in the VMCS, a virtualization exception causes a VM exit (see below). If the bit is 0, the virtualization exception is delivered using gate descriptor 20 in the IDT.
- Virtualization exceptions produce no error code. Delivery of a virtualization exception pushes no error code on the stack.
- With respect to double faults, virtualization exceptions have the same severity as page faults. If delivery of a virtualization exception encounters a nested fault that is either contributory or a page fault, a double fault (#DF) is generated. See Chapter 6, "Interrupt 8—Double Fault Exception (#DF)" in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

It is not possible for a virtualization exception to be encountered while delivering another exception (see Section 26.5.7.1).

If a virtualization exception causes a VM exit directly (because bit 20 is 1 in the exception bitmap), information about the exception is saved normally in the VM-exit interruption information field in the VMCS (see Section 28.2.2). Specifically, the event is reported as a hardware exception with vector 20 and no error code. Bit 12 of the field (NMI unblocking due to IRET) is set normally.

If a virtualization exception causes a VM exit indirectly (because bit 20 is 0 in the exception bitmap and delivery of the exception generates an event that causes a VM exit), information about the exception is saved normally in the IDT-vectoring information field in the VMCS (see Section 28.2.4). Specifically, the event is reported as a hardware exception with vector 20 and no error code.

26.5.8 PASID Translation

The ENQCMD and ENQCMDs instructions each performs a 64-byte enqueue store that includes a 20-bit PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMDs, the PASID is normally read from memory.

If the “PASID translation” VM-execution control is 1, the PASID value identified in the previous paragraph is treated as a **guest PASID**. PASID translation converts this guest PASID to a 20-bit **host PASID**. After this translation, the enqueue store is performed, using the host PASID in place of the guest PASID.

PASID translation is implemented by two hierarchies of data structures (**PASID-translation hierarchies**) configured by a VMM. Guest PASIDs 00000H to 7FFFFH are translated through the low PASID-translation hierarchy, while guest PASIDs 80000 to FFFFFH are translated through the high PASID-translation hierarchy.

The root of each PASID-translation hierarchy is a 4-KByte **PASID directory**. The low PASID directory is located at the low PASID directory address, and the high PASID directory is located at the high PASID directory address (these physical addresses are VM-execution control fields in the VMCS). A PASID directory comprises 512 8-byte entries, each of which has the following format:

- Bit 0 is the entry’s present bit. The entry is used only if this bit is 1.
- Bits 11:1 are reserved and must be 0.
- Bits M–1:12 specify the 4-KByte aligned address of a PASID table (see below), where M is the processor’s physical-address width.
- Bits 63:M are reserved and must be 0.

A PASID-translation hierarchy also includes up to 512 4-KByte **PASID tables**; each of these is referenced by a PASID directory entry (see above). A PASID table comprises 1024 4-byte entries, each of which has the following format:

- Bits 19:0 are the host PASID specified by the entry.
- Bits 30:20 are reserved and must be 0.
- Bit 31 is the entry’s valid bit. The entry is used only if this bit is 1.

When PASID translation is enabled, the guest PASID determined by the instruction (see above) is converted to a host PASID using the following process:

- If bit 19 of guest PASID is clear, the low PASID directory is used; otherwise, the high PASID directory is used.
- Bits 18:10 of the guest PASID select an entry from the PASID directory. A VM exit occurs if the entry’s present bit is clear or if any reserved bit is set. Otherwise, bits M:0 of the entry (with bit 0 cleared) contain the physical address of a PASID table.
- Bits 9:0 of the guest PASID select an entry from the PASID table. A VM exit occurs if the entry’s valid bit is clear or if any reserved bit is set. Otherwise, bits 19:0 of the entry are the host PASID.

If PASID translation results in a VM exit (due to a present or valid bit being clear, or a reserved bit being set), the instruction does not complete and no enqueue store is performed.

26.6 UNRESTRICTED GUESTS

The first processors to support VMX operation require CR0.PE and CR0.PG to be 1 in VMX operation (see Section 24.8). This restriction implies that guest software cannot be run in unpagged protected mode or in real-address mode. Later processors support a VM-execution control called “unrestricted guest”.¹ If this control is 1, CR0.PE and CR0.PG may be 0 in VMX non-root operation. Such processors allow guest software to run in unpagged protected mode or in real-address mode. The following items describe the behavior of such software:

- The MOV CR0 instructions does not cause a general-protection exception simply because it would set either CR0.PE and CR0.PG to 0. See Section 26.3 for details.

1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “unrestricted guest” VM-execution control were 0. See Section 25.6.2.

VMX NON-ROOT OPERATION

- A logical processor treats the values of CR0.PE and CR0.PG in VMX non-root operation just as it does outside VMX operation. Thus, if CR0.PE = 0, the processor operates as it does normally in real-address mode (for example, it uses the 16-bit **interrupt table** to deliver interrupts and exceptions). If CR0.PG = 0, the processor operates as it does normally when paging is disabled.
- Processor operation is modified by the fact that the processor is in VMX non-root operation and by the settings of the VM-execution controls just as it is in protected mode or when paging is enabled. Instructions, interrupts, and exceptions that cause VM exits in protected mode or when paging is enabled also do so in real-address mode or when paging is disabled. The following examples should be noted:
 - If CR0.PG = 0, page faults do not occur and thus cannot cause VM exits.
 - If CR0.PE = 0, invalid-TSS exceptions do not occur and thus cannot cause VM exits.
 - If CR0.PE = 0, the following instructions cause invalid-opcode exceptions and do not cause VM exits: INVEPT, INVVPID, LLDT, LTR, SLDT, STR, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.
- If CR0.PG = 0, each linear address is passed directly to the EPT mechanism for translation to a physical address.¹ The guest memory type passed on to the EPT mechanism is WB (writeback).

1. As noted in Section 27.2.1.1, the “enable EPT” VM-execution control must be 1 if the “unrestricted guest” VM-execution control is 1.

14. Updates to Chapter 27, Volume 3C

Change bars and green text show changes to Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Updated cross-reference in a footnote in Section 27.2.1.1, "VM-Execution Control Fields."

Software can enter VMX non-root operation using either of the VM-entry instructions VMLAUNCH and VMRESUME. VMLAUNCH can be used only with a VMCS whose launch state is clear and VMRESUME can be used only with a VMCS whose the launch state is launched. VMLAUNCH should be used for the first VM entry after VMCLEAR; VMRESUME should be used for subsequent VM entries with the same VMCS.

Each VM entry performs the following steps in the order indicated:

1. Basic checks are performed to ensure that VM entry can commence (Section 27.1).
2. The control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation and that the VMCS is correctly configured to support the next VM exit (Section 27.2).
3. The following may be performed in parallel or in any order (Section 27.3):
 - The guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures.
 - Processor state is loaded from the guest-state area and based on controls in the VMCS.
 - Address-range monitoring is cleared.
4. MSR's are loaded from the VM-entry MSR-load area (Section 27.4).
5. If VMLAUNCH is being executed, the launch state of the VMCS is set to "launched."
6. If the "Intel PT uses guest physical addresses" VM-execution control is 1, trace-address pre-translation (TAPT) may occur (see Section 26.5.4 and Section 27.5).
7. An event may be injected in the guest context (Section 27.6).

Steps 1–4 above perform checks that may cause VM entry to fail. Such failures occur in one of the following three ways:

- Some of the checks in Section 27.1 may generate ordinary faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 27.1 and all the checks in Section 27.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF¹ (if there is a current VMCS) or RFLAGS.CF (if there is no current VMCS). If there is a current VMCS, an error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 31 for the error numbers.
- The checks in Section 27.3 and Section 27.4 cause processor state to be loaded from the host-state area of the VMCS (as would be done on a VM exit). Information about the failure is stored in the VM-exit information fields. See Section 27.8 for details.

EFLAGS.TF = 1 causes a VM-entry instruction to generate a single-step debug exception only if failure of one of the checks in Section 27.1 and Section 27.2 causes control to pass to the following instruction. A VM-entry does not generate a single-step debug exception in any of the following cases: (1) the instruction generates a fault; (2) failure of one of the checks in Section 27.3 or in loading MSR's causes processor state to be loaded from the host-state area of the VMCS; or (3) the instruction passes all checks in Section 27.1, Section 27.2, and Section 27.3 and there is no failure in loading MSR's.

Section 32.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, code running in SMM returns using VM entries instead of the RSM instruction. A VM entry **returns from SMM** if it is executed in SMM and the "entry to SMM" VM-entry control is 0. VM entries that return from SMM differ from ordinary VM entries in ways that are detailed in Section 32.15.4.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

27.1 BASIC VM-ENTRY CHECKS

Before a VM entry commences, the current state of the logical processor is checked in the following order:

1. If the logical processor is in virtual-8086 mode or compatibility mode, an invalid-opcode exception is generated.
2. If the current privilege level (CPL) is not zero, a general-protection exception is generated.
3. If there is no current VMCS, RFLAGS.CF is set to 1 and control passes to the next instruction.
4. If there is a current VMCS but the current VMCS is a shadow VMCS (see Section 25.10), RFLAGS.CF is set to 1 and control passes to the next instruction.
5. If there is a current VMCS that is not a shadow VMCS, the following conditions are evaluated in order; any of these cause VM entry to fail:
 - a. If there is MOV-SS blocking (see Table 25-3).
 - b. If the VM entry is invoked by VMLAUNCH and the VMCS launch state is not clear.
 - c. If the VM entry is invoked by VMRESUME and the VMCS launch state is not launched.

If any of these checks fail, RFLAGS.ZF is set to 1 and control passes to the next instruction. An error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 31 for the error numbers.

27.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 27.1 do not cause VM entry to fail, the control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation, that the VMCS is correctly configured to support the next VM exit, and that, after the next VM exit, the processor's state is consistent with the Intel 64 and IA-32 architectures.

VM entry fails if any of these checks fail. When such failures occur, control is passed to the next instruction, RFLAGS.ZF is set to 1 to indicate the failure, and the VM-instruction error field is loaded with an error number that indicates whether the failure was due to the controls or the host-state area (see Chapter 31).

These checks may be performed in any order. Thus, an indication by error number of one cause (for example, host state) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same VMCS. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The checks on the controls and the host-state area are presented in Section 27.2.1 through Section 27.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

27.2.1 Checks on VMX Controls

This section identifies VM-entry checks on the VMX control fields.

27.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:¹

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.1).
-
1. If the "activate secondary controls" primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0. Similarly, if the "activate tertiary controls" primary processor-based VM-execution control is 0, VM entry operates as if each tertiary processor-based VM-execution control were 0. See Section 25.6.2.

- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.2).
- If the “activate secondary controls” primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.3).
If the “activate secondary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.
- If the “activate tertiary controls” primary processor-based VM-execution control is 1, reserved bits in the tertiary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.4).
If the “activate tertiary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the tertiary processor-based VM-execution controls. The logical processor operates as if all the tertiary processor-based VM-execution controls were 0.
- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix A.6).
- If the “use I/O bitmaps” VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. Neither address should set any bits beyond the processor’s physical-address width.^{1,2}
- If the “use MSR bitmaps” VM-execution control is 1, bits 11:0 of the MSR-bitmap address must be 0. The address should not set any bits beyond the processor’s physical-address width.³
- If the “use TPR shadow” VM-execution control is 1, the virtual-APIC address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.⁴
 If all of the above checks are satisfied and the “use TPR shadow” VM-execution control is 1, bytes 3:1 of VTPR (see Section 30.1.1) may be cleared (behavior may be implementation-specific).
The clearing of these bytes may occur even if the VM entry fails. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it causes processor state to be loaded from the host-state area of the VMCS.
- If the “use TPR shadow” VM-execution control is 1 and the “virtual-interrupt delivery” VM-execution control is 0, bits 31:4 of the TPR threshold VM-execution control field must be 0.
- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” and “virtual-interrupt delivery” VM-execution controls are both 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 of VTPR (see Section 30.1.1).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.
- If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.⁵

1. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

2. If IA32_VMX_BASIC[48] is read as 1, these addresses must not set any bits in the range 63:32; see Appendix A.1.

3. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

5. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, “virtual-interrupt delivery”, and “IPI virtualization”.
- If the “virtualize x2APIC mode” VM-execution control is 1, the “virtualize APIC accesses” VM-execution control must be 0.
- If the “virtual-interrupt delivery” VM-execution control is 1, the “external-interrupt exiting” VM-execution control must be 1.
- If the “process posted interrupts” VM-execution control is 1, the following must be true:
 - The “virtual-interrupt delivery” VM-execution control is 1.
 - The “acknowledge interrupt on exit” VM-exit control is 1.
 - The posted-interrupt notification vector has a value in the range 0–255 (bits 15:8 are all 0).
 - Bits 5:0 of the posted-interrupt descriptor address are all 0.
 - The posted-interrupt descriptor address does not set any bits beyond the processor’s physical-address width.¹
- If the “IPI virtualization” VM-execution control is 1, the following must be true:
 - Bits 2:0 of the PID-pointer table address are all 0.
 - The PID-pointer table address does not set any bits beyond the processor’s physical-address width.
 - The address of the last entry in the PID-pointer table does not set any bits beyond the processor’s physical-address width. (This address is the PID-pointer table address plus 8 times the last PID-pointer index.)
- If the “enable VPID” VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.
- If the “enable EPT” VM-execution control is 1, the EPTP VM-execution control field (see Table 25-9 in Section 25.6.11) must satisfy the following checks:
 - The EPT memory type (bits 2:0) must be a value supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10).
 - Bits 5:3 (1 less than the EPT page-walk length) must be 3, indicating an EPT page-walk length of 4; see Section 29.3.2.
 - Bit 6 (enable bit for accessed and dirty flags for EPT) must be 0 if bit 21 of the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10) is read as 0, indicating that the processor does not support accessed and dirty flags for EPT.
 - Reserved bits 11:7 and 63:N (where N is the processor’s physical-address width) must all be 0.
- The “enable EPT” VM-execution control must be 1 if any of the following VM-execution controls is 1: “enable PML”, “unrestricted guest”, “mode-based execute control for EPT”, “sub-page write permissions for EPT”, “Intel PT uses guest physical addresses”, “enable HLAT”, “EPT paging-write control”, or “guest-paging verification”.
- If the “enable PML” VM-execution control is 1, the PML address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.
- If the “sub-page write permissions for EPT” VM-execution control is 1, the SPPTP VM-execution control field (see Table 25-11 in Section 25.6.22) must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.
- If the “enable VM functions” processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.11). In addition, the following check is performed based on the setting of bits in the VM-function controls (see Section 25.6.14):

1. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

- If “EPTP switching” VM-function control is 1, the “enable EPT” VM-execution control must also be 1. In addition, the EPTP-list address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address must not set any bits beyond the processor’s physical-address width.

If the “enable VM functions” processor-based VM-execution control is 0, no checks are performed on the VM-function controls.

- If the “VMCS shadowing” VM-execution control is 1, the VMREAD-bitmap and VMWRITE-bitmap addresses must each satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address must not set any bits beyond the processor’s physical-address width.
- If the “EPT-violation #VE” VM-execution control is 1, the virtualization-exception information address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address must not set any bits beyond the processor’s physical-address width.
- If the logical processor is operating with Intel PT enabled (if IA32_RTIT_CTL.TraceEn = 1) at the time of VM entry, the “load IA32_RTIT_CTL” VM-entry control must be 0.
- If the “Intel PT uses guest physical addresses” VM-execution control is 1, the “load IA32_RTIT_CTL” VM-entry control and the “clear IA32_RTIT_CTL” VM-exit control must both be 1.
- If the “use TSC scaling” VM-execution control is 1, the TSC-multiplier must not be zero.
- If the “enable HLAT” VM-execution control is 1, the following bits in the HLATP VM-execution control field (see Table 25-12 in Section 25.6.23) must be zero: bits 2:0, bits 11:5, and bits beyond the processor’s physical-address width.
- If the “PASID translation” VM-execution control is 1, the low PASID directory address and the high PASID directory address must each satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address must not set any bits beyond the processor’s physical-address width.

27.2.1.2 VM-Exit Control Fields

VM entries perform the following checks on the VM-exit control fields.

- Reserved bits in the primary VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.4.1).
- If the “activate secondary controls” primary VM-exit control is 1, reserved bits in the secondary VM-exit controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.4.2).
- If the “activate secondary controls” primary VM-exit control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary VM-exit controls. The logical processor operates as if all the secondary VM-exit controls were 0.
- If the “activate VMX-preemption timer” VM-execution control is 0, the “save VMX-preemption timer value” VM-exit control must also be 0.
- The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-store address must be 0. The address should not set any bits beyond the processor’s physical-address width.¹

1. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

- The address of the last byte in the VM-exit MSR-store area should not set any bits beyond the processor's physical-address width. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) - 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.

- The following checks are performed for the VM-exit MSR-load address if the VM-exit MSR-load count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-load address must be 0. The address should not set any bits beyond the processor's physical-address width.
 - The address of the last byte in the VM-exit MSR-load area should not set any bits beyond the processor's physical-address width. The address of this last byte is VM-exit MSR-load address + (MSR count * 16) - 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)
- If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.

27.2.1.3 VM-Entry Control Fields

VM entries perform the following checks on the VM-entry control fields.

- Reserved bits in the VM-entry controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.5).
- Fields relevant to VM-entry event injection must be set properly. These fields are the VM-entry interruption-information field (see Table 25-16 in Section 25.8.3), the VM-entry exception error code, and the VM-entry instruction length. If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the following must hold:
 - The field's interruption type (bits 10:8) is not set to a reserved value. Value 1 is reserved on all logical processors; value 7 (other event) is reserved on logical processors that do not support the 1-setting of the "monitor trap flag" VM-execution control.
 - The field's vector (bits 7:0) is consistent with the interruption type:
 - If the interruption type is non-maskable interrupt (NMI), the vector is 2.
 - If the interruption type is hardware exception, the vector is at most 31.
 - If the interruption type is other event, the vector is 0 (pending MTF VM exit).
 - The field's deliver-error-code bit (bit 11) is 1 if each of the following holds: (1) the interruption type is hardware exception; (2) bit 0 (corresponding to CR0.PE) is set in the CR0 field in the guest-state area; (3) IA32_VMX_BASIC[56] is read as 0 (see Appendix A.1); and (4) the vector indicates one of the following exceptions: #DF (vector 8), #TS (10), #NP (11), #SS (12), #GP (13), #PF (14), or #AC (17).
 - The field's deliver-error-code bit is 0 if any of the following holds: (1) the interruption type is not hardware exception; (2) bit 0 is clear in the CR0 field in the guest-state area; or (3) IA32_VMX_BASIC[56] is read as 0 and the vector is in one of the following ranges: 0-7, 9, 15, 16, or 18-31.
 - Reserved bits in the field (30:12) are 0.
 - If the deliver-error-code bit (bit 11) is 1, bits 31:16 of the VM-entry exception error-code field are 0.
 - If the interruption type is software interrupt, software exception, or privileged software exception, the VM-entry instruction-length field is in the range 0-15. A VM-entry instruction length of 0 is allowed only if IA32_VMX_MISC[30] is read as 1; see Appendix A.6.
- The following checks are performed for the VM-entry MSR-load address if the VM-entry MSR-load count field is non-zero:
 - The lower 4 bits of the VM-entry MSR-load address must be 0. The address should not set any bits beyond the processor's physical-address width.¹

1. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

- The address of the last byte in the VM-entry MSR-load area should not set any bits beyond the processor's physical-address width. The address of this last byte is VM-entry MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.

- If the processor is not in SMM, the "entry to SMM" and "deactivate dual-monitor treatment" VM-entry controls must be 0.
- The "entry to SMM" and "deactivate dual-monitor treatment" VM-entry controls cannot both be 1.

27.2.2 Checks on Host Control Registers, MSRs, and SSP

The following checks are performed on fields in the host-state area that correspond to control registers and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 24.8).¹
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 24.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- On processors that support Intel 64 architecture, the CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width must be 0.^{2,3}
- On processors that support Intel 64 architecture, the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 20-3).
- If the "load IA32_PAT" VM-exit control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the "load IA32_EFER" VM-exit control is 1, bits reserved in the IA32_EFER MSR must be 0 in the field for that register. In addition, the values of the LMA and LME bits in the field must each be that of the "host address-space size" VM-exit control.
- If the "load CET state" VM-exit control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) in the field cannot both be set.
- If the "load CET state" VM-exit control is 1, bits 1:0 must be 0 in the SSP field.
- If the "load PKRS" VM-exit control is 1, bits 63:32 must be 0 in the IA32_PKRS field.

27.2.3 Checks on Host Segment and Descriptor-Table Registers

The following checks are performed on fields in the host-state area that correspond to segment and descriptor-table registers:

- In the selector field for each of CS, SS, DS, ES, FS, GS, and TR, the RPL (bits 1:0) and the TI flag (bit 2) must be 0.
- The selector fields for CS and TR cannot be 0000H.
- The selector field for SS cannot be 0000H if the "host address-space size" VM-exit control is 0.
- On processors that support Intel 64 architecture, the base-address fields for FS, GS, GDTR, IDTR, and TR must contain canonical addresses.

1. The bits corresponding to CR0.NW (bit 29) and CR0.CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 28.5.1.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. Bit 63 of the CR3 field in the host-state area must be 0. This is true even though, if CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

27.2.4 Checks Related to Address-Space Size

On processors that support Intel 64 architecture, the following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If the logical processor is outside IA-32e mode (if IA32_EFER.LMA = 0) at the time of VM entry, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.
 - The “host address-space size” VM-exit control is 0.
- If the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1) at the time of VM entry, the “host address-space size” VM-exit control must be 1.
- If the “host address-space size” VM-exit control is 0, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.
 - Bit 17 of the CR4 field (corresponding to CR4.PCIDE) is 0.
 - Bits 63:32 in the RIP field are 0.
 - If the “load CET state” VM-exit control is 1, bits 63:32 in the IA32_S_CET field and in the SSP field are 0.
- If the “host address-space size” VM-exit control is 1, the following must hold:
 - Bit 5 of the CR4 field (corresponding to CR4.PAE) is 1.
 - The RIP field contains a canonical address.
 - If the “load CET state” VM-exit control is 1, the IA32_S_CET field and the SSP field contain canonical addresses.
- If the “load CET state” VM-exit control is 1, the IA32_INTERRUPT_SSP_TABLE_ADDR field contains a canonical address.

On processors that do not support Intel 64 architecture, checks are performed to ensure that the “IA-32e mode guest” VM-entry control and the “host address-space size” VM-exit control are both 0.

27.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 27.2), the following operations take place concurrently: (1) the guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures; (2) processor state is loaded from the guest-state area or as specified by the VM-entry control fields; and (3) address-range monitoring is cleared.

Because the checking and the loading occur concurrently, a failure may be discovered only after some state has been loaded. For this reason, the logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 27.8.

27.3.1 Checks on the Guest State Area

This section describes checks performed on fields in the guest-state area. These checks may be performed in any order. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The following subsections reference fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

27.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 24.8). The following are exceptions:
 - Bit 0 (corresponding to CR0.PE) and bit 31 (PG) are not checked if the “unrestricted guest” VM-execution control is 1.¹
 - Bit 29 (corresponding to CR0.NW) and bit 30 (CD) are never checked because the values of these bits are not changed by VM entry; see Section 27.3.2.1.
- If bit 31 in the CR0 field (corresponding to PG) is 1, bit 0 in that field (PE) must also be 1.²
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 24.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- If the “load debug controls” VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.
- The following checks are performed on processors that support Intel 64 architecture:
 - If the “IA-32e mode guest” VM-entry control is 1, bit 31 in the CR0 field (corresponding to CR0.PG) and bit 5 in the CR4 field (corresponding to CR4.PAE) must each be 1.³
 - If the “IA-32e mode guest” VM-entry control is 0, bit 17 in the CR4 field (corresponding to CR4.PCIDE) must be 0.
 - The CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor’s physical-address width are 0.^{4,5}
 - If the “load debug controls” VM-entry control is 1, bits 63:32 in the DR7 field must be 0. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
 - The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
 - If the “load CET state” VM-entry control is 1, the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field must contain canonical addresses.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 20-3).
- If the “load IA32_PAT” VM-entry control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the “load IA32_EFER” VM-entry control is 1, the following checks are performed on the field for the IA32_EFER MSR:
 - Bits reserved in the IA32_EFER MSR must be 0.
 - Bit 10 (corresponding to IA32_EFER.LMA) must equal the value of the “IA-32e mode guest” VM-entry control. It must also be identical to bit 8 (LME) if bit 31 in the CR0 field (corresponding to CR0.PG) is 1.⁶

-
1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 25.6.2.
 2. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
 3. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
 4. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
 5. Bit 63 of the CR3 field in the guest-state area must be 0. This is true even though, if CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.
 6. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- If the “load IA32_BNDCFGS” VM-entry control is 1, the following checks are performed on the field for the IA32_BNDCFGS MSR:
 - Bits reserved in the IA32_BNDCFGS MSR must be 0.
 - The linear address in bits 63:12 must be canonical.
- If the “load IA32_RTIT_CTL” VM-entry control is 1, bits reserved in the IA32_RTIT_CTL MSR must be 0 in the field for that register (see Table 33-6).
- If the “load CET state” VM-entry control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) of the field cannot both be set.
- If the “load guest IA32_LBR_CTL” VM-entry control is 1, bits reserved in the IA32_LBR_CTL MSR must be 0 in the field for that register.
- If the “load PKRS” VM-entry control is 1, bits 63:32 must be 0 in the IA32_PKRS field.
- If the “load UINV” VM-entry control is 1, bits 15:8 must be 0 in the guest UINV field.

27.3.1.2 Checks on Guest Segment Registers

This section specifies the checks on the fields for CS, SS, DS, ES, FS, GS, TR, and LDTR. The following terms are used in defining these checks:

- The guest will be **virtual-8086** if the VM flag (bit 17) is 1 in the RFLAGS field in the guest-state area.
- The guest will be **IA-32e mode** if the “IA-32e mode guest” VM-entry control is 1. (This is possible only on processors that support Intel 64 architecture.)
- Any one of these registers is said to be **usable** if the unusable bit (bit 16) is 0 in the access-rights field for that register.

The following are the checks on these fields:

- Selector fields.
 - TR. The TI flag (bit 2) must be 0.
 - LDTR. If LDTR is usable, the TI flag (bit 2) must be 0.
 - SS. If the guest will not be virtual-8086 and the “unrestricted guest” VM-execution control is 0, the RPL (bits 1:0) must equal the RPL of the selector field for CS.¹
- Base-address fields.
 - CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the address must be the selector field shifted left 4 bits (multiplied by 16).
 - The following checks are performed on processors that support Intel 64 architecture:
 - TR, FS, GS. The address must be canonical.
 - LDTR. If LDTR is usable, the address must be canonical.
 - CS. Bits 63:32 of the address must be zero.
 - SS, DS, ES. If the register is usable, bits 63:32 of the address must be zero.
- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields.
 - CS, SS, DS, ES, FS, GS.
 - If the guest will be virtual-8086, the field must be 000000F3H. This implies the following:
 - Bits 3:0 (Type) must be 3, indicating an expand-up read/write accessed data segment.
 - Bit 4 (S) must be 1.

1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 25.6.2.

- Bits 6:5 (DPL) must be 3.
- Bit 7 (P) must be 1.
- Bits 11:8 (reserved), bit 12 (software available), bit 13 (reserved/L), bit 14 (D/B), bit 15 (G), bit 16 (unusable), and bits 31:17 (reserved) must all be 0.
- If the guest will not be virtual-8086, the different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - CS. The values allowed depend on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, the Type must be 9, 11, 13, or 15 (accessed code segment).
 - If the control is 1, the Type must be either 3 (read/write accessed expand-up data segment) or one of 9, 11, 13, and 15 (accessed code segment).
 - SS. If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
 - DS, ES, FS, GS. The following checks apply if the register is usable:
 - Bit 0 of the Type must be 1 (accessed).
 - If bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
 - Bit 4 (S). If the register is CS or if the register is usable, S must be 1.
 - Bits 6:5 (DPL).
 - CS.
 - If the Type is 3 (read/write accessed expand-up data segment), the DPL must be 0. The Type can be 3 only if the “unrestricted guest” VM-execution control is 1.
 - If the Type is 9 or 11 (non-conforming code segment), the DPL must equal the DPL in the access-rights field for SS.
 - If the Type is 13 or 15 (conforming code segment), the DPL cannot be greater than the DPL in the access-rights field for SS.
 - SS.
 - If the “unrestricted guest” VM-execution control is 0, the DPL must equal the RPL from the selector field.
 - The DPL must be 0 either if the Type in the access-rights field for CS is 3 (read/write accessed expand-up data segment) or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.¹
 - DS, ES, FS, GS. The DPL cannot be less than the RPL in the selector field if (1) the “unrestricted guest” VM-execution control is 0; (2) the register is usable; and (3) the Type in the access-rights field is in the range 0 – 11 (data segment or non-conforming code segment).
 - Bit 7 (P). If the register is CS or if the register is usable, P must be 1.
 - Bits 11:8 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
 - Bit 14 (D/B). For CS, D/B must be 0 if the guest will be IA-32e mode and the L bit (bit 13) in the access-rights field is 1.
 - Bit 15 (G). The following checks apply if the register is CS or if the register is usable:
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
 - Bits 31:17 (reserved). If the register is CS or if the register is usable, these bits must all be 0.

1. The following apply if either the “unrestricted guest” VM-execution control or bit 31 of the primary processor-based VM-execution controls is 0: (1) bit 0 in the CR0 field must be 1 if the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation; and (2) the Type in the access-rights field for CS cannot be 3.

- TR. The different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - If the guest will not be IA-32e mode, the Type must be 3 (16-bit busy TSS) or 11 (32-bit busy TSS).
 - If the guest will be IA-32e mode, the Type must be 11 (64-bit busy TSS).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
 - Bit 16 (Unusable). The unusable bit must be 0.
 - Bits 31:17 (reserved). These bits must all be 0.
- LDTR. The following checks on the different sub-fields apply only if LDTR is usable:
 - Bits 3:0 (Type). The Type must be 2 (LDT).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
 - Bits 31:17 (reserved). These bits must all be 0.

27.3.1.3 Checks on Guest Descriptor-Table Registers

The following checks are performed on the fields for GDTR and IDTR:

- On processors that support Intel 64 architecture, the base-address fields must contain canonical addresses.
- Bits 31:16 of each limit field must be 0.

27.3.1.4 Checks on Guest RIP, RFLAGS, and SSP

The following checks are performed on fields in the guest-state area corresponding to RIP, RFLAGS, and SSP (shadow-stack pointer):

- RIP. The following checks are performed on processors that support Intel 64 architecture:
 - Bits 63:32 must be 0 if the “IA-32e mode guest” VM-entry control is 0 or if the L bit (bit 13) in the access-rights field for CS is 0.
 - If the processor supports $N < 64$ linear-address bits, bits 63:N must be identical if the “IA-32e mode guest” VM-entry control is 1 and the L bit in the access-rights field for CS is 1.¹ (No check applies if the processor supports 64 linear-address bits.) The guest RIP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.
- RFLAGS.
 - Reserved bits 63:22 (bits 31:22 on processors that do not support Intel 64 architecture), bit 15, bit 5 and bit 3 must be 0 in the field, and reserved bit 1 must be 1.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

- The VM flag (bit 17) must be 0 either if the “IA-32e mode guest” VM-entry control is 1 or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.¹
- The IF flag (RFLAGS[bit 9]) must be 1 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) is external interrupt.
- SSP. The following checks are performed if the “load CET state” VM-entry control is 1
 - Bits 1:0 must be 0.
 - If the processor supports the Intel 64 architecture, bits 63:N must be identical, where N is the CPU’s maximum linear-address width. (This check does not apply if the processor supports 64 linear-address bits.) The guest SSP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.

27.3.1.5 Checks on Guest Non-Register State

The following checks are performed on fields in the guest-state area corresponding to non-register state:

- Activity state.
 - The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section 25.4.2). Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.
 - The activity-state field must not indicate the HLT state if the DPL (bits 6:5) in the access-rights field for SS is not 0.²
 - The activity-state field must indicate the active state if the interruptibility-state field indicates blocking by either MOV-SS or by STI (if either bit 0 or bit 1 in that field is 1).
 - If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the interruption to be delivered (as defined by interruption type and vector) must not be one that would normally be blocked while a logical processor is in the activity state corresponding to the contents of the activity-state field. The following items enumerate the interruptions (as specified in the VM-entry interruption-information field) whose injection is allowed for the different activity states:
 - Active. Any interruption is allowed.
 - HLT. The only events allowed are the following:
 - Those with interruption type external interrupt or non-maskable interrupt (NMI).
 - Those with interruption type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
 - Those with interruption type other event and vector 0 (pending MTF VM exit).
 See Table 25-16 in Section 25.8.3 for details regarding the format of the VM-entry interruption-information field.
 - Shutdown. Only NMIs and machine-check exceptions are allowed.
 - Wait-for-SIPI. No interruptions are allowed.
 - The activity-state field must not indicate the wait-for-SIPI state if the “entry to SMM” VM-entry control is 1.
- Interruptibility state.
 - The reserved bits (bits 31:5) must be 0.
 - The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).
 - Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

2. As noted in Section 25.4.1, SS.DPL corresponds to the logical processor’s current privilege level (CPL).

- Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 0, indicating external interrupt, or value 2, indicating non-maskable interrupt (NMI).
- Bit 2 (blocking by SMI) must be 0 if the processor is not in SMM.
- Bit 2 (blocking by SMI) must be 1 if the “entry to SMM” VM-entry control is 1.
- Bit 3 (blocking by NMI) must be 0 if the “virtual NMIs” VM-execution control is 1, the valid bit (bit 31) in the VM-entry interruption-information field is 1, and the interruption type (bits 10:8) in that field has value 2 (indicating NMI).
- If bit 4 (enclave interruption) is 1, bit 1 (blocking by MOV-SS) must be 0 and the processor must support for SGX by enumerating CPUID.(EAX=07H,ECX=0):EBX.SGX[bit 2] as 1.

NOTE

If the “virtual NMIs” VM-execution control is 0, there is no requirement that bit 3 be 0 if the valid bit in the VM-entry interruption-information field is 1 and the interruption type in that field has value 2.

- Pending debug exceptions.
 - Bits 11:4, bit 13, bit 15, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0.
 - The following checks are performed if any of the following holds: (1) the interruptibility-state field indicates blocking by STI (bit 0 in that field is 1); (2) the interruptibility-state field indicates blocking by MOV SS (bit 1 in that field is 1); or (3) the activity-state field indicates HLT:
 - Bit 14 (BS) must be 1 if the TF flag (bit 8) in the RFLAGS field is 1 and the BTF flag (bit 1) in the IA32_DEBUGCTL field is 0.
 - Bit 14 (BS) must be 0 if the TF flag (bit 8) in the RFLAGS field is 0 or the BTF flag (bit 1) in the IA32_DEBUGCTL field is 1.
 - The following checks are performed if bit 16 (RTM) is 1:
 - Bits 11:0, bits 15:13, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0; bit 12 must be 1.
 - The processor must support for RTM by enumerating CPUID.(EAX=07H,ECX=0):EBX[bit 11] as 1.
 - The interruptibility-state field must not indicate blocking by MOV SS (bit 1 in that field must be 0).
- VMCS link pointer. The following checks apply if the field contains a value other than FFFFFFFF_FFFFFFFFH:
 - Bits 11:0 must be 0.
 - Bits beyond the processor’s physical-address width must be 0.^{1,2}
 - The 4 bytes located in memory referenced by the value of the field (as a physical address) must satisfy the following:
 - Bits 30:0 must contain the processor’s VMCS revision identifier (see Section 25.2).³
 - Bit 31 must contain the setting of the “VMCS shadowing” VM-execution control.⁴ This implies that the referenced VMCS is a shadow VMCS (see Section 25.10) if and only if the “VMCS shadowing” VM-execution control is 1.

1. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

2. If IA32_VMX_BASIC[48] is read as 1, this field must not set any bits in the range 63:32; see Appendix A.1.

3. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.

4. “VMCS shadowing” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “VMCS shadowing” VM-execution control were 0. See Section 25.6.2.

- If the processor is not in SMM or the “entry to SMM” VM-entry control is 1, the field must not contain the current VMCS pointer.
- If the processor is in SMM and the “entry to SMM” VM-entry control is 0, the field must differ from the executive-VMCS pointer.

27.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, the logical processor uses **PAE paging** (see Section 4.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).¹ When PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTes.

A VM entry is to a guest that uses PAE paging if (1) bit 31 (corresponding to CR0.PG) is set in the CR0 field in the guest-state area; (2) bit 5 (corresponding to CR4.PAE) is set in the CR4 field; and (3) the “IA-32e mode guest” VM-entry control is 0. Such a VM entry checks the validity of the PDPTes:

- If the “enable EPT” VM-execution control is 0, VM entry checks the validity of the PDPTes referenced by the CR3 field in the guest-state area if either (1) PAE paging was not in use before the VM entry; or (2) the value of CR3 is changing as a result of the VM entry. VM entry may check their validity even if neither (1) nor (2) hold.²
- If the “enable EPT” VM-execution control is 1, VM entry checks the validity of the PDPTe fields in the guest-state area (see Section 25.4.2).

A VM entry to a guest that does not use PAE paging does not check the validity of any PDPTes.

A VM entry that checks the validity of the PDPTes uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use.³ If MOV to CR3 would cause a general-protection exception due to the PDPTes that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

27.3.2 Loading Guest State

Processor state is updated on VM entries in the following ways:

- Some state is loaded from the guest-state area.
- Some state is determined by VM-entry controls.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order and in parallel with the checking of VMCS contents (see Section 27.3.1).

The loading of guest state is detailed in Section 27.3.2.1 to Section 27.3.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

In addition to the state loading described in this section, VM entries may load MSRs from the VM-entry MSR-load area (see Section 27.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 27.3.1.

27.3.2.1 Loading Guest Control Registers, Debug Registers, and MSRs

The following items describe how guest control registers, debug registers, and MSRs are loaded on VM entry:

-
1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine the number physical-address bits supported by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
 2. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “enable EPT” VM-execution control were 0. See Section 25.6.2.
 3. This implies that (1) bits 11:9 in each PDPTe are ignored; and (2) if bit 0 (present) is clear in one of the PDPTes, bits 63:1 of that PDPTe are ignored.

- CR0 is loaded from the CR0 field with the exception of the following bits, which are never modified on VM entry: ET (bit 4); reserved bits 15:6, 17, and 28:19; NW (bit 29) and CD (bit 30).¹ The values of these bits in the CR0 field are ignored.
- CR3 and CR4 are loaded from the CR3 field and the CR4 field, respectively.
- If the “load debug controls” VM-entry control is 1, DR7 is loaded from the DR7 field with the exception that bit 12 and bits 15:14 are always 0 and bit 10 is always 1. The values of these bits in the DR7 field are ignored. The first processors to support the virtual-machine extensions supported only the 1-setting of the “load debug controls” VM-entry control and thus always loaded DR7 from the DR7 field.
- The following describes how certain MSRs are loaded using fields in the guest-state area:
 - If the “load debug controls” VM-entry control is 1, the IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always loaded the IA32_DEBUGCTL MSR from the IA32_DEBUGCTL field.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since this field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
 - The following are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 27.3.2.2).
 - If the “load IA32_EFER” VM-entry control is 0, bits in the IA32_EFER MSR are modified as follows:
 - IA32_EFER.LMA is loaded with the setting of the “IA-32e mode guest” VM-entry control.
 - If CR0 is being loaded so that CR0.PG = 1, IA32_EFER.LME is also loaded with the setting of the “IA-32e mode guest” VM-entry control.² Otherwise, IA32_EFER.LME is unmodified.

See below for the case in which the “load IA32_EFER” VM-entry control is 1

 - If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
 - If the “load IA32_PAT” VM-entry control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
 - If the “load IA32_EFER” VM-entry control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.
 - If the “load IA32_BNDCFGS” VM-entry control is 1, the IA32_BNDCFGS MSR is loaded from the IA32_BNDCFGS field.
 - If the “load IA32_RTIT_CTL” VM-entry control is 1, the IA32_RTIT_CTL MSR is loaded from the IA32_RTIT_CTL field.
 - If the “load CET” VM-entry control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
 - If the “load guest IA32_LBR_CTL” VM-entry control is 1, the IA32_LBR_CTL MSR is loaded from the IA32_LBR_CTL guest state field.
 - If the “load PKRS” VM-entry control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 27.4.

-
1. Bits 15:6, bit 17, and bit 28:19 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. Bits 15:6, bit 17, and bit 28:19 of CR0 are always 0 and CR0.ET is always 1.
 2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, VM entry must be loading CR0 so that CR0.PG = 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- The SMBASE register is unmodified by all VM entries except those that return from SMM.

27.3.2.2 Loading Guest Segment Registers and Descriptor-Table Registers

For each of CS, SS, DS, ES, FS, GS, TR, and LDTR, fields are loaded from the guest-state area as follows:

- The unusable bit is loaded from the access-rights field. This bit can never be set for TR (see Section 27.3.1.2). If it is set for one of the other registers, the following apply:
 - For each of CS, SS, DS, ES, FS, and GS, uses of the segment cause faults (general-protection exception or stack-fault exception) outside 64-bit mode, just as they would had the segment been loaded using a null selector. This bit does not cause accesses to fault in 64-bit mode.
 - If this bit is set for LDTR, uses of LDTR cause general-protection exceptions in all modes, just as they would had LDTR been loaded using a null selector.

If this bit is clear for any of CS, SS, DS, ES, FS, GS, TR, and LDTR, a null selector value does not cause a fault (general-protection exception or stack-fault exception).
- TR. The selector, base, limit, and access-rights fields are loaded.
- CS.
 - The following fields are always loaded: selector, base address, limit, and (from the access-rights field) the L, D, and G bits.
 - For the other fields, the unusable bit of the access-rights field is consulted:
 - If the unusable bit is 0, all of the access-rights field is loaded.
 - If the unusable bit is 1, the remainder of CS access rights are undefined after VM entry.
- SS, DS, ES, FS, GS, and LDTR.
 - The selector fields are loaded.
 - For the other fields, the unusable bit of the corresponding access-rights field is consulted:
 - If the unusable bit is 0, the base-address, limit, and access-rights fields are loaded.
 - If the unusable bit is 1, the base address, the segment limit, and the remainder of the access rights are undefined after VM entry with the following exceptions:
 - Bits 3:0 of the base address for SS are cleared to 0.
 - SS.DPL is always loaded from the SS access-rights field. This will be the current privilege level (CPL) after the VM entry completes.
 - SS.B is always set to 1.
 - The base addresses for FS and GS are loaded from the corresponding fields in the VMCS. On processors that support Intel 64 architecture, the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
 - On processors that support Intel 64 architecture, the base address for LDTR is set to an undefined but canonical value.
 - On processors that support Intel 64 architecture, bits 63:32 of the base addresses for SS, DS, and ES are cleared to 0.

GDTR and IDTR are loaded using the base and limit fields.

27.3.2.3 Loading Guest RIP, RSP, RFLAGS, and SSP

RSP, RIP, and RFLAGS are loaded from the RSP field, the RIP field, and the RFLAGS field, respectively.

If the “load CET” VM-entry control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

The following items regard the upper 32 bits of these fields on VM entries that are not to 64-bit mode:

- Bits 63:32 of RSP are undefined outside 64-bit mode. Thus, a logical processor may ignore the contents of bits 63:32 of the RSP field on VM entries that are not to 64-bit mode.

- As noted in Section 27.3.1.4, bits 63:32 of the RIP and RFLAGS fields must be 0 on VM entries that are not to 64-bit mode. (The same is true for SSP for VM entries that are not to 64-bit mode when the “load CET” VM-entry control is 1.)

27.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 27.3.1.6, the logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. A VM entry to a guest that uses PAE paging loads the PDPTEs into internal, non-architectural registers based on the setting of the “enable EPT” VM-execution control:

- If the control is 0, the PDPTEs are loaded from the page-directory-pointer table referenced by the physical address in the value of CR3 being loaded by the VM entry (see Section 27.3.2.1). The values loaded are treated as physical addresses in VMX non-root operation.
- If the control is 1, the PDPTEs are loaded from corresponding fields in the guest-state area (see Section 25.4.2). The values loaded are treated as guest-physical addresses in VMX non-root operation.

27.3.2.5 Updating Non-Register State

Section 29.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM entries invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).
- VM entries are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

If the “virtual-interrupt delivery” VM-execution control is 1, VM entry loads the values of RVI and SVI from the guest interrupt-status field in the VMCS (see Section 25.4.2). After doing so, the logical processor first causes PPR virtualization (Section 30.1.3) and then evaluates pending virtual interrupts (Section 30.2.1).

If a virtual interrupt is recognized, it may be delivered in VMX non-root operation immediately after VM entry (including any specified event injection) completes; see Section 27.7.5. See Section 30.2.2 for details regarding the delivery of virtual interrupts.

27.3.3 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 9.10.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. VM entries clear any address-range monitoring that may be in effect.

27.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 25.8.2). Specifically each entry in that area (up to the number specified in the VM-entry MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.¹

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101 (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM entry did not commence in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)

1. Because attempts to modify the value of IA32_EFER.LMA by WRMSR are ignored, attempts to modify it using the VM-entry MSR-load area are also ignored.

- The value of bits 31:0 indicates an MSR that cannot be loaded on VM entries for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.¹

The VM entry fails if processing fails for any entry. The logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 27.8.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM entry, the logical processor will not use any translations that were cached before the transition.

27.5 TRACE-ADDRESS PRE-TRANSLATION (TAPT)

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses, and these are translated to physical addresses using EPT.

VM entry uses **trace-address pre-translation (TAPT)** to prevent buffered trace data from being lost due to an EPT violation; see Section 26.5.4.2. VM entry uses TAPT only if Intel PT will be enabled following VM entry (IA32_RTIT_CTL.TraceEn = 1) and only if the “Intel PT uses guest physical addresses” VM-execution control is 1

As noted in Section 26.5.4, TAPT may cause a VM exit due to an EPT violation, EPT misconfiguration, page-modification log-full event, or APIC access. If such a VM exit occurs as a result of TAPT during VM entry, the VM exit operates as if it had occurred in VMX non-root operation after the VM entry completed (in the guest context).

If TAPT during VM entry causes a VM exit, the VM entry does not perform event injection (Section 27.6), even if the valid bit in the VM-entry interruption-information field is 1. Such VM exits save the contents of VM-entry interruption-information and VM-entry exception error code fields into the IDT-vectoring information and IDT-vectoring error code fields, respectively.

27.6 EVENT INJECTION

If the valid bit in the VM-entry interruption-information field (see Section 25.8.3) is 1, VM entry causes an event to be delivered (or made pending) after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.

- If the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception), the event is delivered as described in Section 27.6.1.
- If the interruption type in the field is 7 (other event) and the vector field is 0, an MTF VM exit is pending after VM entry. See Section 27.6.2.

27.6.1 Vectored-Event Injection

VM entry delivers an injected vectored event within the guest context established by VM entry. This means that delivery occurs after all components of guest state have been loaded (including MSRs) and after the VM-execution

1. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. If VM entry has established CR0.PG = 1, the IA32_EFER MSR should not be included in the VM-entry MSR-load area for the purpose of modifying the LME bit.

control fields have been established.¹ The event is delivered using the vector in that field to select a descriptor in the IDT. Since event injection occurs after loading IDTR from the guest-state area, this is the guest IDT.

Section 27.6.1.1 provides details of vectored-event injection. In general, the event is delivered exactly as if it had been generated normally.

An exception is made if the following all hold: bit 25 (UINTR) is set to 1 in the guest CR4 field and the “IA-32e mode guest” VM-entry control is 1, and VM entry is modified if it is injecting an external interrupt whose vector is the value that UINV would have after VM entry. In this case, the logical processor then performs user-interrupt notification processing as specified in Section 7.5.2 instead of the process described in Section 27.6.1.1. (If the guest activity-state field indicated the HLT state, the logical processor enters the HLT state following user-interrupt notification processing.)

If event delivery (or user-interrupt notification processing; see above) encounters a nested exception (for example, a general-protection exception because the vector indicates a descriptor beyond the IDT limit), the exception bitmap is consulted using the vector of that exception:

- If the bit for the nested exception is 0, the nested exception is delivered normally. If the nested exception is benign, it is delivered through the IDT. If it is contributory or a page fault, a double fault may be generated, depending on the nature of the event whose delivery encountered the nested exception. See Chapter 6, “Interrupt 8—Double Fault Exception (#DF)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.²
- If the bit for the nested exception is 1, a VM exit occurs. Section 27.6.1.2 details cases in which event injection causes a VM exit.

27.6.1.1 Details of Vectored-Event Injection

The event-injection process is controlled by the contents of the VM-entry interruption information field (format given in Table 25-16), the VM-entry exception error-code field, and the VM-entry instruction-length field. The following items provide details of the process:

- The value pushed on the stack for RFLAGS is generally that which was loaded from the guest-state area. The value pushed for the RF flag is not modified based on the type of event being delivered. However, the pushed value of RFLAGS may be modified if a software interrupt is being injected into a guest that will be in virtual-8086 mode (see below). After RFLAGS is pushed on the stack, the value in the RFLAGS register is modified as is done normally when delivering an event through the IDT.
- The instruction pointer that is pushed on the stack depends on the type of event and whether nested exceptions occur during its delivery. The term **current guest RIP** refers to the value to be loaded from the guest-state area. The value pushed is determined as follows:³
 - If VM entry successfully injects (with no nested exception) an event with interruption type external interrupt, NMI, or hardware exception, the current guest RIP is pushed on the stack.
 - If VM entry successfully injects (with no nested exception) an event with interruption type software interrupt, privileged software exception, or software exception, the current guest RIP is incremented by the VM-entry instruction length before being pushed on the stack.
 - If VM entry encounters an exception while injecting an event and that exception does not cause a VM exit, the current guest RIP is pushed on the stack regardless of event type or VM-entry instruction length. If the encountered exception does cause a VM exit that saves RIP, the saved RIP is current guest RIP.
- If the deliver-error-code bit (bit 11) is set in the VM-entry interruption-information field, the contents of the VM-entry exception error-code field is pushed on the stack as an error code would be pushed during delivery of an exception.

1. This does not imply that injection of an exception or interrupt will cause a VM exit due to the settings of VM-execution control fields (such as the exception bitmap) that would cause a VM exit if the event had occurred in VMX non-root operation. In contrast, a nested exception encountered during event delivery may cause a VM exit; see Section 27.6.1.1.

2. Hardware exceptions with the following unused vectors are considered benign: 15 and 21–31. A hardware exception with vector 20 is considered benign unless the processor supports the 1-setting of the “EPT-violation #VE” VM-execution control; in that case, it has the same severity as page faults.

3. While these items refer to RIP, the width of the value pushed (16 bits, 32 bits, or 64 bits) is determined normally.

- DR6, DR7, and the IA32_DEBUGCTL MSR are not modified by event injection, even if the event has vector 1 (normal deliveries of debug exceptions, which have vector 1, do update these registers).
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode (RFLAGS.VM = 1), no general-protection exception can occur due to RFLAGS.IOPL < 3. A VM monitor should check RFLAGS.IOPL before injecting such an event and, if desired, inject a general-protection exception instead of a software interrupt.
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode with virtual-8086 mode extensions (RFLAGS.VM = CR4.VME = 1), event delivery is subject to VME-based interrupt redirection based on the software interrupt redirection bitmap in the task-state segment (TSS) as follows:
 - If bit n in the bitmap is clear (where n is the number of the software interrupt), the interrupt is directed to an 8086 program interrupt handler: the processor uses a 16-bit interrupt-vector table (IVT) located at linear address zero. If the value of RFLAGS.IOPL is less than 3, the following modifications are made to the value of RFLAGS that is pushed on the stack: IOPL is set to 3, and IF is set to the value of VIF.
 - If bit n in the bitmap is set (where n is the number of the software interrupt), the interrupt is directed to a protected-mode interrupt handler. (In other words, the injection is treated as described in the next item.) In this case, the software interrupt does not invoke such a handler if RFLAGS.IOPL < 3 (a general-protection exception occurs instead). However, as noted above, RFLAGS.IOPL cannot cause an injected software interrupt to cause such an exception. Thus, in this case, the injection invokes a protected-mode interrupt handler independent of the value of RFLAGS.IOPL.

Injection of events of other types are not subject to this redirection.

- If VM entry is injecting a software interrupt (not redirected as described above) or software exception, privilege checking is performed on the IDT descriptor being accessed as would be the case for executions of INT n , INT3, or INTO (the descriptor's DPL cannot be less than CPL). There is no checking of RFLAGS.IOPL, even if the guest will be in virtual-8086 mode. Failure of this check may lead to a nested exception. Injection of an event with interruption type external interrupt, NMI, hardware exception, and privileged software exception, or with interruption type software interrupt and being redirected as described above, do not perform these checks.
- If VM entry is injecting a non-maskable interrupt (NMI) and the "virtual NMIs" VM-execution control is 1, virtual-NMI blocking is in effect after VM entry.
- The transition causes a last-branch record to be logged if the LBR bit is set in the IA32_DEBUGCTL MSR. This is true even for events such as debug exceptions, which normally clear the LBR bit before delivery.
- The last-exception record MSRs (LERs) may be updated based on the setting of the LBR bit in the IA32_DEBUGCTL MSR. Events such as debug exceptions, which normally clear the LBR bit before they are delivered, and therefore do not normally update the LERs, may do so as part of VM-entry event injection.
- If injection of an event encounters a nested exception, the value of the EXT bit (bit 0) in any error code for that nested exception is determined as follows:
 - If event being injected has interruption type external interrupt, NMI, hardware exception, or privileged software exception and encounters a nested exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If event being injected is a software interrupt or a software exception and encounters a nested exception, the error code for that exception clears the EXT bit.
 - If event delivery encounters a nested exception and delivery of that exception encounters another exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If a double fault is produced, the error code for the double fault is 0000H (the EXT bit is clear).

27.6.1.2 VM Exits During Event Injection

An event being injected never causes a VM exit directly regardless of the settings of the VM-execution controls. For example, setting the "NMI exiting" VM-execution control to 1 does not cause a VM exit due to injection of an NMI.

However, the event-delivery process may lead to a VM exit:

- If the vector in the VM-entry interruption-information field identifies a task gate in the IDT, the attempted task switch may cause a VM exit just as it would had the injected event occurred during normal execution in VMX non-root operation (see Section 26.4.2).

- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 26.2).
- If event delivery generates a double-fault exception (due to a nested exception); the logical processor encounters another nested exception while attempting to call the double-fault handler; and that exception does not cause a VM exit due to the exception bitmap; then a VM exit occurs due to triple fault (see Section 26.2).
- If event delivery injects a double-fault exception and encounters a nested exception that does not cause a VM exit due to the exception bitmap, then a VM exit occurs due to triple fault (see Section 26.2).
- If the “virtualize APIC accesses” VM-execution control is 1 and event delivery generates an access to the APIC-access page, that access is treated as described in Section 30.4 and may cause a VM exit.¹

If the event-delivery process does cause a VM exit, the processor state before the VM exit is determined just as it would be had the injected event occurred during normal execution in VMX non-root operation. If the injected event directly accesses a task gate that cause a VM exit or if the first nested exception encountered causes a VM exit, information about the injected event is saved in the IDT-vectoring information field (see Section 28.2.4).

The material in this section applies also if injection of an external interrupt results in user-interrupt notification processing instead of event delivery (see Section 27.6.1 earlier).

27.6.1.3 Event Injection for VM Entries to Real-Address Mode

If VM entry is loading CR0.PE with 0, any injected vectored event is delivered as would normally be done in real-address mode.² Specifically, VM entry uses the vector provided in the VM-entry interruption-information field to select a 4-byte entry from an interrupt-vector table at the linear address in IDTR.base. Further details are provided in Section 15.1.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Because bit 11 (deliver error code) in the VM-entry interruption-information field must be 0 if CR0.PE will be 0 after VM entry (see Section 27.2.1.3), vectored events injected with CR0.PE = 0 do not push an error code on the stack. This is consistent with event delivery in real-address mode.

If event delivery encounters a fault (due to a violation of IDTR.limit or of SS.limit), the fault is treated as if it had occurred during event delivery in VMX non-root operation. Such a fault may lead to a VM exit as discussed in Section 27.6.1.2.

27.6.2 Injection of Pending MTF VM Exits

If the interruption type in the VM-entry interruption-information field is 7 (other event) and the vector field is 0, VM entry causes an MTF VM exit to be pending on the instruction boundary following VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0. See Section 26.5.2 for the treatment of pending MTF VM exits.

27.7 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **vectoring** if the valid bit (bit 31) of the VM-entry interruption information field is 1 and the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

1. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 25.6.2.

2. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PE must be 1 in VMX operation, VM entry must be loading CR0.PE with 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

27.7.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 25-3) contains bits that control blocking by STI, blocking by MOV SS, and blocking by NMI. This field impacts event blocking after VM entry as follows:

- If the VM entry is vectoring, there is no blocking by STI or by MOV SS following the VM entry, regardless of the contents of the interruptibility-state field.
- If the VM entry is not vectoring, the following apply:
 - Events are blocked by STI if and only if bit 0 in the interruptibility-state field is 1. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry; see Section 27.7.3).
 - Events are blocked by MOV SS if and only if bit 1 in the interruptibility-state field is 1. This may affect the treatment of pending debug exceptions; see Section 27.7.3. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry).
- The blocking of non-maskable interrupts (NMIs) is determined as follows:
 - If the “virtual NMIs” VM-execution control is 0, NMIs are blocked if and only if bit 3 (blocking by NMI) in the interruptibility-state field is 1. If the “NMI exiting” VM-execution control is 0, execution of the IRET instruction removes this blocking (even if the instruction generates a fault). If the “NMI exiting” control is 1, IRET does not affect this blocking.
 - The following items describe the use of bit 3 (blocking by NMI) in the interruptibility-state field if the “virtual NMIs” VM-execution control is 1:
 - The bit’s value does not affect the blocking of NMIs after VM entry. NMIs are not blocked in VMX non-root operation (except for ordinary blocking for other reasons, such as by the MOV SS instruction, the wait-for-SIPI state, etc.)
 - The bit’s value determines whether there is virtual-NMI blocking after VM entry. If the bit is 1, virtual-NMI blocking is in effect after VM entry. If the bit is 0, there is no virtual-NMI blocking after VM entry unless the VM entry is injecting an NMI (see Section 27.6.1.1). Execution of IRET removes virtual-NMI blocking (even if the instruction generates a fault).

If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 27.2.1.1.
- Blocking of system-management interrupts (SMIs) is determined as follows:
 - If the VM entry was not executed in system-management mode (SMM), SMI blocking is unchanged by VM entry.
 - If the VM entry was executed in SMM, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.

27.7.2 Activity State

The activity-state field in the guest-state area controls whether, after VM entry, the logical processor is active or in one of the inactive states identified in Section 25.4.2. The use of this field is determined as follows:

- If the VM entry is vectoring, the logical processor is in the active state after VM entry. While the consistency checks described in Section 27.3.1.5 on the activity-state field do apply in this case, the contents of the activity-state field do not determine the activity state after VM entry.
- If the VM entry is not vectoring, the logical processor ends VM entry in the activity state specified in the guest-state area. If VM entry ends with the logical processor in an inactive activity state, the VM entry generates any special bus cycle that is normally generated when that activity state is entered from the active state. If VM entry would end with the logical processor in the shutdown state and the logical processor is in SMX operation,¹ an Intel® TXT shutdown condition occurs. The error code used is 0000H, indicating “legacy shutdown.” See *Intel® Trusted Execution Technology Preliminary Architecture Specification*.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENER]. See Chapter 6, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.

- Some activity states unconditionally block certain events. The following blocking is in effect after any VM entry that puts the processor in the indicated state:
 - The active state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the active state and in VMX non-root operation are discarded and do not cause VM exits.
 - The HLT state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the HLT state and in VMX non-root operation are discarded and do not cause VM exits.
 - The shutdown state blocks external interrupts and SIPIs. External interrupts that arrive while a logical processor is in the shutdown state and in VMX non-root operation do not cause VM exits even if the “external-interrupt exiting” VM-execution control is 1. SIPIs that arrive while a logical processor is in the shutdown state and in VMX non-root operation are discarded and do not cause VM exits.
 - The wait-for-SIPI state blocks external interrupts, non-maskable interrupts (NMIs), INIT signals, and system-management interrupts (SMIs). Such events do not cause VM exits if they arrive while a logical processor is in the wait-for-SIPI state and in VMX non-root operation.

27.7.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area indicates whether there are debug exceptions that have not yet been delivered (see Section 25.4.2). This section describes how these are treated on VM entry.

There are no pending debug exceptions after VM entry if any of the following are true:

- The VM entry is vectoring with one of the following interruption types: external interrupt, non-maskable interrupt (NMI), hardware exception, or privileged software exception.
- The interruptibility-state field does not indicate blocking by MOV SS and the VM entry is vectoring with either of the following interruption type: software interrupt or software exception.
- The VM entry is not vectoring and the activity-state field indicates either shutdown or wait-for-SIPI.

If none of the above hold, the pending debug exceptions field specifies the debug exceptions that are pending for the guest. There are **valid pending debug exceptions** if either the BS bit (bit 14) or the enable-breakpoint bit (bit 12) is 1. If there are valid pending debug exceptions, they are handled as follows:

- If the VM entry is not vectoring, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
 - If the logical processor is not blocking such exceptions (the interruptibility-state field indicates no blocking by MOV SS), a debug exception is delivered after VM entry (see below).
 - If the logical processor is blocking such exceptions (due to blocking by MOV SS), the pending debug exceptions are held pending or lost as would normally be the case.
- If the VM entry is vectoring (with interruption type software interrupt or software exception and with blocking by MOV SS), the following items apply:
 - For injection of a software interrupt or of a software exception with vector 3 (#BP) or vector 4 (#OF) — or a privileged software exception with vector 1 (#DB) — the pending debug exceptions are treated as they would had they been encountered normally in guest execution if the corresponding instruction (INT1, INT3, or INTO) were executed after a MOV SS that encountered a debug trap.
 - For injection of a software exception with a vector other than 3 and 4, the pending debug exceptions may be lost or they may be delivered after injection (see below).

If there are no valid pending debug exceptions (as defined above), no pending debug exceptions are delivered after VM entry.

If a pending debug exception is delivered after VM entry, it has the priority of “traps on the previous instruction” (see Section 6.9 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). Thus, INIT signals and system-management interrupts (SMIs) take priority of such an exception, as do VM exits induced by the TPR threshold (see Section 27.7.7) and pending MTF VM exits (see Section 27.7.8). The exception takes priority over any pending non-maskable interrupt (NMI) or external interrupt and also over VM exits due to the 1-settings of the “interrupt-window exiting” and “NMI-window exiting” VM-execution controls.

A pending debug exception delivered after VM entry causes a VM exit if the bit 1 (#DB) is 1 in the exception bitmap. If it does not cause a VM exit, it updates DR6 normally.

27.7.4 VMX-Preemption Timer

If the “activate VMX-preemption timer” VM-execution control is 1, VM entry starts the VMX-preemption timer with the unsigned value in the VMX-preemption timer-value field.

It is possible for the VMX-preemption timer to expire during VM entry (e.g., if the value in the VMX-preemption timer-value field is zero). If this happens (and if the VM entry was not to the wait-for-SIPI state), a VM exit occurs with its normal priority after any event injection and before execution of any instruction following VM entry. For example, any pending debug exceptions established by VM entry (see Section 27.7.3) take priority over a timer-induced VM exit. (The timer-induced VM exit will occur after delivery of the debug exception, unless that exception or its delivery causes a different VM exit.)

See Section 26.5.1 for details of the operation of the VMX-preemption timer in VMX non-root operation, including the blocking and priority of the VM exits that it causes.

27.7.5 Interrupt-Window Exiting and Virtual-Interrupt Delivery

If “interrupt-window exiting” VM-execution control is 1, an open interrupt window may cause a VM exit immediately after VM entry (see Section 26.2 for details). If the “interrupt-window exiting” VM-execution control is 0 but the “virtual-interrupt delivery” VM-execution control is 1, a virtual interrupt may be delivered immediately after VM entry (see Section 27.3.2.5 and Section 30.2.1).

The following items detail the treatment of these events:

- These events occur after any event injection specified for VM entry.
- Non-maskable interrupts (NMIs) and higher priority events take priority over these events. These events take priority over external interrupts and lower priority events.
- These events wake the logical processor if it just entered the HLT state because of a VM entry (see Section 27.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

27.7.6 NMI-Window Exiting

The “NMI-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 26.2 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.
- Debug-trap exceptions (see Section 27.7.3) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered either the HLT state or the shutdown state because of a VM entry (see Section 27.7.2). They do not occur if the logical processor just entered the wait-for-SIPI state.

27.7.7 VM Exits Induced by the TPR Threshold

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1 and the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs immediately after VM entry if the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 of VTPR (see Section 30.1.1).¹

1. “Virtualize APIC accesses” and “virtual-interrupt delivery” are secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if these controls were 0. See Section 25.6.2.

The following items detail the treatment of these VM exits:

- The VM exits are not blocked if RFLAGS.IF = 0 or by the setting of bits in the interruptibility-state field in guest-state area.
- The VM exits follow event injection if such injection is specified for VM entry.
- VM exits caused by this control take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. They thus have priority over the VM exits described in Section 27.7.5, Section 27.7.6, and Section 27.7.8, as well as any interrupts or debug exceptions that may be pending at the time of VM entry.
- These VM exits wake the logical processor if it just entered the HLT state as part of a VM entry (see Section 27.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state. If such a VM exit is suppressed because the processor just entered the shutdown state, it occurs after the delivery of any event that cause the logical processor to leave the shutdown state while remaining in VMX non-root operation (e.g., due to an NMI that occurs while the “NMI-exiting” VM-execution control is 0).
- The basic exit reason is “TPR below threshold.”

27.7.8 Pending MTF VM Exits

As noted in Section 27.6.2, VM entry may cause an MTF VM exit to be pending immediately after VM entry. The following items detail the treatment of these VM exits:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these VM exits. These VM exits take priority over debug-trap exceptions and lower priority events.
- These VM exits wake the logical processor if it just entered the HLT state because of a VM entry (see Section 27.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

27.7.9 VM Entries and Advanced Debugging Features

VM entries are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

27.7.10 User-Interrupt Recognition After VM Entry

A VM entry results in recognition of a pending user interrupt if it completes with CR4.UINTR = IA32_EFER.LMA = 1 and with UIRR ≠ 0; otherwise, no pending user interrupt is recognized.

27.8 VM-ENTRY FAILURES DURING OR AFTER LOADING GUEST STATE

VM-entry failures due to the checks identified in Section 27.3.1 and failures during the MSR loading identified in Section 27.4 are treated differently from those that occur earlier in VM entry. In these cases, the following steps take place:

1. Information about the VM-entry failure is recorded in the VM-exit information fields:
 - Exit reason.
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM-entry failure. The following numbers are used:
 33. VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 27.3.1.
 34. VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 27.4).
 41. VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 27.9).

- Bit 31 is set to 1 to indicate a VM-entry failure.
 - The remainder of the field (bits 30:16) is cleared.
- Exit qualification. This field is set based on the exit reason.
- VM-entry failure due to invalid guest state. In most cases, the exit qualification is cleared to 0. The following non-zero values are used in the cases indicated:
 1. Not used.
 2. Failure was due to a problem loading the PDPTes (see Section 27.3.1.6).
 3. Failure was due to an attempt to inject a non-maskable interrupt (NMI) into a guest that is blocking events through the STI blocking bit in the interruptibility-state field.
 4. Failure was due to an invalid VMCS link pointer (see Section 27.3.1.5).

VM-entry checks on guest-state fields may be performed in any order. Thus, an indication by exit qualification of one cause does not imply that there are not also other errors. Different processors may give different exit qualifications for the same VMCS.
 - VM-entry failure due to MSR loading. The exit qualification is loaded to indicate which entry in the VM-entry MSR-load area caused the problem (1 for the first entry, 2 for the second, etc.).
- All other VM-exit information fields are unmodified.
2. Processor state is loaded as would be done on a VM exit (see Section 28.5). If this results in $[CR4.PAE \& CR0.PG \& \sim IA32_EFER.LMA] = 1$, page-directory-pointer-table entries (PDPTes) may be checked and loaded (see Section 28.5.4).
 3. The state of blocking by NMI is what it was before VM entry.
 4. MSRs are loaded as specified in the VM-exit MSR-load area (see Section 28.6).

Although this process resembles that of a VM exit, many steps taken during a VM exit do not occur for these VM-entry failures:

- Most VM-exit information fields are not updated (see step 1 above).
- The valid bit in the VM-entry interruption-information field is not cleared.
- The guest-state area is not modified.
- No MSRs are saved into the VM-exit MSR-store area.

27.9 MACHINE-CHECK EVENTS DURING VM ENTRY

If a machine-check event occurs during a VM entry, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM entry:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, an Intel[®] TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If CR4.MCE = 1, a machine-check exception (#MC) is delivered through the IDT.
- The machine-check event is handled after VM entry completes:
 - If the VM entry ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel[®] TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM entry ends with CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- A VM-entry failure occurs as described in Section 27.8. The basic exit reason is 41, for “VM-entry failure due to machine-check event.”

The first option is not used if the machine-check event occurs after any guest state has been loaded. The second option is used only if VM entry is able to load all guest state.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in the Intel[®] 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.

15. Updates to Chapter 28, Volume 3C

Change bars and green text show changes to Chapter 28 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Updates throughout the chapter for bus-lock detection and instruction timeout features.

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 26.1 through Section 26.2. VM exits perform the following operations:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and VM-entry control fields are modified as described in Section 28.2.
2. Processor state is saved in the guest-state area (Section 28.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 28.4). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.
4. The following may be performed in parallel and in any order (Section 28.5):
 - Processor state is loaded based in part on the host-state area and some VM-exit controls. This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM. See Section 32.15.6 for information on how processor state is loaded by such VM exits.
 - Address-range monitoring is cleared.
5. MSRs may be loaded from the VM-exit MSR-load area (Section 28.6). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.

VM exits are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

Section 28.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 28.2 through Section 28.6.

Section 32.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, ordinary transitions to SMM are replaced by VM exits to a separate SMM monitor. Called **SMM VM exits**, these are caused by the arrival of an SMI or the execution of VMCALL in VMX root operation. SMM VM exits differ from other VM exits in ways that are detailed in Section 32.15.2.

28.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section describes the architectural state that exists before a VM exit, especially for VM exits caused by events that would normally be delivered through the IDT. Note the following:

- An exception causes a VM exit **directly** if the bit corresponding to that exception is set in the exception bitmap. A non-maskable interrupt (NMI) causes a VM exit directly if the “NMI exiting” VM-execution control is 1. An external interrupt causes a VM exit directly if the “external-interrupt exiting” VM-execution control is 1. A start-up IPI (SIPI) that arrives while a logical processor is in the wait-for-SIPI activity state causes a VM exit directly. INIT signals that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.
- An exception, NMI, external interrupt, or software interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, task switch, APIC access (see Section 30.4), EPT violation, EPT misconfiguration, page-modification log-full event (see Section 29.3.6), or SPP-related event (see Section 29.3.4) that causes a VM exit.
- An event **results** in a VM exit if it causes a VM exit (directly or indirectly).

The following bullets detail when architectural state is and is not updated in response to VM exits:

- If an event causes a VM exit directly, it does not update architectural state as it would have if it had it not caused the VM exit:
 - A debug exception does not update DR6, DR7, or IA32_DEBUGCTL. (Information about the nature of the debug exception is saved in the exit qualification field.)
 - A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)

- An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
 - An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the “acknowledge interrupt on exit” VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
 - The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
 - If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT¹; or the TR register.
 - No last-exception record is made if the event that would do so directly causes a VM exit.
 - If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine-check event itself and not the resulting machine-check exception.
 - If the logical processor is in an inactive state (see Section 25.4.2) and not executing instructions, some events may be blocked but others may return the logical processor to the active state. Unblocked events may cause VM exits.² If an unblocked event causes a VM exit directly, a return to the active state occurs only after the VM exit completes.³ The VM exit generates any special bus cycle that is normally generated when the active state is entered from that activity state.
- MTF VM exits (see Section 26.5.2 and Section 27.7.8) are not blocked in the HLT activity state. If an MTF VM exit occurs in the HLT activity state, the logical processor returns to the active state only after the VM exit completes. MTF VM exits are blocked the shutdown state and the wait-for-SIPI state.
- If an event causes a VM exit indirectly, the event does update architectural state:
 - A debug exception updates DR6, DR7, and the IA32_DEBUGCTL MSR. No debug exceptions are considered pending.
 - A page fault updates CR2.
 - An NMI causes subsequent NMIs to be blocked before the VM exit commences.
 - An external interrupt acknowledges the interrupt controller and the interrupt is no longer pending.
 - If the logical processor had been in an inactive state, it enters the active state and, before the VM exit commences, generates any special bus cycle that is normally generated when the active state is entered from that activity state.
 - There is no blocking by STI or by MOV SS when the VM exit commences.
 - Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.
 - The treatment of last-exception records is implementation dependent:
 - Some processors make a last-exception record when beginning the delivery of an event through the IDT (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).
 - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-exception record if a VM exit or triple fault occurs before an event handler is reached.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

2. If a VM exit takes the processor from an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

3. An exception is made if the logical processor had been inactive due to execution of MWAIT; in this case, it is considered to have become active before the VM exit.

- If the “virtual NMIs” VM-execution control is 1, VM entry injects an NMI, and delivery of the NMI causes a nested exception, double fault, task switch, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event, or APIC access that causes a VM exit, virtual-NMI blocking is in effect before the VM exit commences.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “NMI exiting” VM-execution control is 0, any blocking by NMI is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the VM-exit interruption-information field; see Section 28.2.3.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “virtual NMIs” VM-execution control is 1, virtual-NMI blocking is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the VM-exit interruption-information field; see Section 28.2.3.
- Suppose that a VM exit is caused directly by an x87 FPU Floating-Point Error (#MF) or by any of the following events if the event was unblocked due to (and given priority over) an x87 FPU Floating-Point Error: an INIT signal, an external interrupt, an NMI, an SMI; or a machine-check exception. In these cases, there is no blocking by STI or by MOV SS when the VM exit commences.
- Normally, a last-branch record may be made when an event is delivered through the IDT. However, if such an event results in a VM exit before delivery is complete, no last-branch record is made.
- If machine-check exception results in a VM exit, processor state is suspect and may result in suspect state being saved to the guest-state area. A VM monitor should consult the RIPV and EIPV bits in the IA32_MC_G_STATUS MSR before resuming a guest that caused a VM exit resulting from a machine-check exception.
- If a VM exit results from a fault, APIC access (see Section 30.4), EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered while executing an instruction, data breakpoints due to that instruction may have been recognized and information about them may be saved in the pending debug exceptions field (unless the VM exit clears that field; see Section 28.3.4).
- The following VM exits are considered to happen after an instruction is executed:
 - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
 - VM exits resulting from debug exceptions (data breakpoints) whose recognition was delayed by blocking by MOV SS.
 - VM exits resulting from some machine-check exceptions.
 - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1 (see Section 30.3). (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)
 - Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1; the value of ECX is in the range 800H–8FFH; and the bit corresponding to the ECX value in write bitmap for low MSRs is 0; and the “virtualize x2APIC mode” VM-execution control is 1. See Section 30.5.
 - VM exits caused by APIC-write emulation (see Section 30.4.3.2) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor’s interruptibility state (see Table 25-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

A VM exit that occurs in enclave mode sets bit 27 of the exit-reason field and bit 4 of the guest interruptibility-state field. Before such a VM exit is delivered, an Asynchronous Enclave Exit (AEX) occurs (see Chapter 37, “Enclave Exiting Events”). An AEX modifies architectural state (Section 37.3). In particular, the processor establishes the following architectural state as indicated:

- The following bits in RFLAGS are cleared: CF, PF, AF, ZF, SF, OF, and RF.
- FS and GS are restored to the values they had prior to the most recent enclave entry.
- RIP is loaded with the AEP of interrupted enclave thread.
- RSP is loaded from the URSP field in the enclave’s state-save area (SSA).

28.2 RECORDING VM-EXIT INFORMATION AND UPDATING VM-ENTRY CONTROL FIELDS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 28.2.1 to Section 28.2.5 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the VM-entry interruption-information field. If bit 5 of the IA32_VMX_MISC MSR (index 485H) is read as 1 (see Appendix A.6), the value of IA32_EFER.LMA is stored into the “IA-32e mode guest” VM-entry control.¹

28.2.1 Basic VM-Exit Information

Section 25.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix C lists the numbers used and their meaning.
 - Bit 26 of this field is set to 1 if the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1. Such VM exits include those that occur due to the 1-setting of that control as well as others that might occur during execution of an instruction that asserted a bus lock.
 - Bit 27 of this field is set to 1 if the VM exit occurred while the logical processor was in enclave mode. Such VM exits include those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode. A VM exit also sets this bit if it is incident to delivery of an event injected by VM entry and the guest interruptibility-state field indicates an enclave interruption (bit 4 of the field is 1).
 - The remainder of the field (bits 31:28 and bits 26:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 32.15.2.3).²
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); system-management interrupts (SMIs) that arrive immediately after the execution of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; WBINVD; WBNOINVD; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 30.4); EPT violations (see Section 29.3.3.2); EOI virtualization (see Section 30.1.4); APIC-write emulation (see Section 30.4.3.3); page-modification log full (see Section 29.3.6); and SPP-related events (see Section 29.3.4). For all other VM exits, this field is cleared. The following items provide details:
 - For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 28-1.

Table 28-1. Exit Qualification for Debug Exceptions

Bit Position(s)	Contents
3:0	B3 – B0. When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if its corresponding enabling bit in DR7 is not set.
10:4	Not currently defined.
11	BLD. When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 18.3.1.6 (“OS Bus-Lock Detection”)). ¹

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.
2. Bit 31 of this field is set on certain VM-entry failures; see Section 27.8.

Table 28-1. Exit Qualification for Debug Exceptions (Contd.)

Bit Position(s)	Contents
12	Not currently defined.
13	BD. When set, this bit indicates that the cause of the debug exception is “debug register access detected.”
14	BS. When set, this bit indicates that the cause of the debug exception is either the execution of a single instruction (if RFLAGS.TF = 1 and IA32_DEBUGCTL.BTF = 0) or a taken branch (if RFLAGS.TF = DEBUGCTL.BTF = 1).
15	Not currently defined.
16	RTM. When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 16.3.7, “RTM-Enabled Debugger Support,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). ²
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- For a page-fault exception, the exit qualification contains the linear address that caused the page fault. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

- For a start-up IPI (SIPI), the exit qualification contains the SIPI vector information in bits 7:0. Bits 63:8 of the exit qualification are cleared to 0.
- For a task switch, the exit qualification contains details about the task switch, encoded as shown in Table 28-2.
- For INVLPG, the exit qualification contains the linear-address operand of the instruction.
 - On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
 - If the INVLPG source operand specifies an unusable segment, the linear address specified in the exit qualification will match the linear address that the INVLPG would have used if no VM exit occurred. This address is not architecturally defined and may be implementation-specific.

Table 28-2. Exit Qualification for Task Switches

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Not currently defined
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT

Table 28-2. Exit Qualification for Task Switches (Contd.)

Bit Position(s)	Contents
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

- For INVEPT, INVPCID, INVVPID, LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, and XSAVES, the exit qualification receives the value of the instruction’s displacement field, which is sign-extended to 64 bits if necessary (32 bits on processors that do not support Intel 64 architecture). If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

On processors that support Intel 64 architecture, an exception is made for RIP-relative addressing (used only in 64-bit mode). Such addressing causes an instruction to use an address that is the sum of the displacement field and the value of RIP that references the following instruction. In this case, the exit qualification is loaded with the sum of the displacement field and the appropriate RIP value.

In all cases, bits of this field beyond the instruction’s address size are undefined. For example, suppose that the address-size field in the VM-exit instruction-information field (see Section 25.9.4 and Section 28.2.5) reports an *n*-bit address size. Then bits 63:*n* (bits 31:*n* on processors that do not support Intel 64 architecture) of the instruction displacement are undefined.

- For a control-register access, the exit qualification contains information about the access and has the format given in Table 28-3.
- For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 28-4.
- For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 28-5.
- For MWAIT, the exit qualification contains a value that indicates whether address-range monitoring hardware was armed. The exit qualification is set either to 0 (if address-range monitoring hardware is not armed) or to 1 (if address-range monitoring hardware is armed).
- WBINVD and WBNOINVD use the same basic exit reason (see Appendix C). For WBINVD, the exit qualification is 0, while for WBNOINVD it is 1.
- For an APIC-access VM exit resulting from a linear access or a guest-physical access to the APIC-access page (see Section 30.4), the exit qualification contains information about the access and has the format given in Table 28-6.¹

If the access to the APIC-access page occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

Such a VM exit that set bits 15:12 of the exit qualification to 0000b (data read during instruction execution) or 0001b (data write during instruction execution) set bit 12—which distinguishes data read from data write—to that which would have been stored in bit 1—W/R—of the page-fault error code had the access caused a page fault instead of an APIC-access VM exit. This implies the following:

- For an APIC-access VM exit caused by the CLFLUSH and CLFLUSHOPT instructions, the access type is “data read during instruction execution.”
- For an APIC-access VM exit caused by the ENTER instruction, the access type is “data write during instruction execution.”
- For an APIC-access VM exit caused by the MASKMOVQ instruction or the MASKMOVDQU instruction, the access type is “data write during instruction execution.”
- For an APIC-access VM exit caused by the MONITOR instruction, the access type is “data read during instruction execution.”

1. The exit qualification is undefined if the access was part of the logging of a branch record or a processor-event-based-sampling (PEBS) record to the DS save area. It is recommended that software configure the paging structures so that no address in the DS save area translates to an address on the APIC-access page.

Table 28-3. Exit Qualification for Control-Register Accesses

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory For CLTS and MOV CR, cleared to 0
7	Not currently defined
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) For CLTS and LMSW, cleared to 0
15:12	Not currently defined
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

- For an APIC-access VM exit caused directly by an access to a linear address in the DS save area (BTS or PEBS), the access type is “linear access for monitoring.”
- For an APIC-access VM exit caused by a guest-physical access performed for an access to the DS save area (e.g., to access a paging structure to translate a linear address), the access type is “guest-physical access for monitoring or trace.”
- For an APIC-access VM exit caused by trace-address pre-translation (TAPT) when the “Intel PT uses guest physical addresses” VM-execution control is 1, the access type is “guest-physical access for monitoring or trace.”

Such a VM exit stores 1 for bit 31 for IDT-vectoring information field (see Section 28.2.4) if and only if it sets bits 15:12 of the exit qualification to 0011b (linear access during event delivery) or 1010b (guest-physical access during event delivery).

See Section 30.4.4 for further discussion of these instructions and APIC-access VM exits.

For APIC-access VM exits resulting from physical accesses to the APIC-access page (see Section 30.4.6), the exit qualification is undefined.

- For an EPT violation, the exit qualification contains information about the access causing the EPT violation and has the format given in Table 28-7.

As noted in that table, the format and meaning of the exit qualification depends on the setting of the “mode-based execute control for EPT” VM-execution control and whether the processor supports advanced VM-exit information for EPT violations.¹

An EPT violation that occurs during as a result of execution of a read-modify-write operation sets bit 1 (data write). Whether it also sets bit 0 (data read) is implementation-specific and, for a given implementation, may differ for different kinds of read-modify-write operations.

Table 28-4. Exit Qualification for MOV DR

Bit Position(s)	Contents
2:0	Number of debug register
3	Not currently defined
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Not currently defined
11:8	General-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8 - 15 = R8 - R15, respectively
63:12	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

Table 28-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)

1. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

Table 28-5. Exit Qualification for I/O Instructions (Contd.)

Bit Position(s)	Contents
15:7	Not currently defined
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

Table 28-6. Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	<ul style="list-style-type: none"> ▪ If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. ▪ Undefined if the APIC-access VM exit is due a guest-physical access
15:12	<p>Access type:</p> <ul style="list-style-type: none"> 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 4 = linear access for monitoring 10 = guest-physical access during event delivery 11 = guest-physical access for monitoring or trace 15 = guest-physical access for an instruction fetch or during instruction execution <p>Other values not used</p>
16	This bit is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These includes guest-physical accesses related to trace output by Intel PT (see Section 26.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 20.9.5), and accesses that occur during user-interrupt delivery (see Section 7.4.2).
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

Bit 12 reports “NMI unblocking due to IRET”; see Section 28.2.3.

Bit 16 is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These include trace-address pre-translation (TAPT) for Intel PT (see Section 26.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 20.9.5), and accesses as part of user-interrupt delivery (see Section 7.4.2).

- For VM exits caused as part of EOI virtualization (Section 30.1.4), bits 7:0 of the exit qualification are set to vector of the virtual interrupt that was dismissed by the EOI virtualization. Bits above bit 7 are cleared.
- For APIC-write VM exits (Section 30.4.3.3), bits 11:0 of the exit qualification are set to the page offset of the write access that caused the VM exit.¹ Bits above bit 11 are cleared.
- For a VM exit due to a page-modification log-full event (Section 29.3.6), bit 12 of the exit qualification reports “NMI unblocking due to IRET.” Bit 16 is set if the VM exit occurs during TAPT, EPT-friendly PEBS, or user-interrupt delivery. All other bits of the exit qualification are undefined.
- For a VM exit due to an SPP-related event (Section 29.3.4), bit 11 of the exit qualification indicates the type of event: 0 indicates an SPP misconfiguration and 1 indicates an SPP miss. Bit 12 of the exit qualification reports “NMI unblocking due to IRET.” Bit 16 is set if the VM exit occurs during TAPT EPT-friendly PEBS, or user-interrupt delivery. All other bits of the exit qualification are undefined.

1. Execution of WRMSR with ECX = 83FH (self-IPI MSR) can lead to an APIC-write VM exit; the exit qualification for such an APIC-write VM exit is 3F0H.

- If the “PASID translation” VM-execution control, PASID translation is performed for executions of the ENQCMD and ENQCMLS instructions (see Section 26.5.8). PASID translation may fail, resulting in a VM exit. Such a VM exit saves an exit qualification specified in the following items:
 - For ENQCMD, the exit qualification is IA32_PASID[19:0].
 - For ENQCMLS, the exit qualification contains the low 32 bits of the instruction’s source operand (which had been read from memory prior to PASID translation).
- **Guest linear address.** For some VM exits, this field receives a linear address that pertains to the VM exit. The field is set for different VM exits as follows:
 - VM exits due to attempts to execute LMSW with a memory operand. In these cases, this field receives the linear address of that operand. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
 - VM exits due to attempts to execute INS or OUTS for which the relevant segment is usable (if the relevant segment is not usable, the value is undefined). (ES is always the relevant segment for INS; for OUTS, the relevant segment is DS unless overridden by an instruction prefix.) The linear address is the base address of relevant segment plus (E)DI (for INS) or (E)SI (for OUTS). Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

Table 28-7. Exit Qualification for EPT Violations

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read. ¹
1	Set if the access causing the EPT violation was a data write. ¹
2	Set if the access causing the EPT violation was an instruction fetch.
3	The logical-AND of bit 0 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was readable). ²
4	The logical-AND of bit 1 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was writeable).
5	The logical-AND of bit 2 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. If the “mode-based execute control for EPT” VM-execution control is 0, this indicates whether the guest-physical address was executable. If that control is 1, this indicates whether the guest-physical address was executable for supervisor-mode linear addresses.
6	If the “mode-based execute control” VM-execution control is 0, the value of this bit is undefined. If that control is 1, this bit is the logical-AND of bit 10 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. In this case, it indicates whether the guest-physical address was executable for user-mode linear addresses.
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTes as part of the execution of the MOV CR instruction and those due to trace-address pre-translation (TAPT; Section 26.5.4).
8	If bit 7 is 1: <ul style="list-style-type: none"> ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).

Table 28-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
9	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ³ this bit is 0 if the linear address is a supervisor-mode linear address and 1 if it is a user-mode linear address. (If CRO.PG = 0, the translation of every linear address is a user-mode linear address and thus this bit will be 1.) Otherwise, this bit is undefined.
10	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ³ this bit is 0 if paging translates the linear address to a read-only page and 1 if it translates to a read/write page. (If CRO.PG = 0, every linear address is read/write and thus this bit will be 1.) Otherwise, this bit is undefined.
11	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ³ this bit is 0 if paging translates the linear address to an executable page and 1 if it translates to an execute-disable page. (If CRO.PG = 0, CR4.PAE = 0, or IA32_EFER.NXE = 0, every linear address is executable and thus this bit will be 0.) Otherwise, this bit is undefined.
12	NMI unblocking due to IRET (see Section 28.2.3).
13	Set if the access causing the EPT violation was a shadow-stack access.
14	If supervisor shadow-stack control is enabled (by setting bit 7 of EPTP), this bit is the same as bit 60 in the EPT paging-structure entry that maps the page of the guest-physical address of the access causing the EPT violation. Otherwise (or if translation of the guest-physical address terminates before reaching an EPT paging-structure entry that maps a page), this bit is undefined.
15	This bit is set if the EPT violation was caused as a result of guest-paging verification. See Section 29.3.3.2.
16	This bit is set if the access was asynchronous to instruction execution not the result of event delivery. The bit is set if the access is related to trace output by Intel PT (see Section 26.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 20.9.5), or to user-interrupt delivery (see Section 7.4.2). Otherwise, this bit is cleared.
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. If accessed and dirty flags for EPT are enabled, processor accesses to guest paging-structure entries are treated as writes with regard to EPT violations (see Section 29.3.3.2). If such an access causes an EPT violation, the processor sets both bit 0 and bit 1 of the exit qualification.
2. Bits 5:3 are cleared to 0 if any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present (see Section 29.3.2).
3. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

- VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 28-7; these are all EPT violations except those resulting from an attempt to load the PDPTes as of execution of the MOV CR instruction and those due to TAPT). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the EPT violation occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

- VM exits due to SPP-related events.
- For all other VM exits, the field is undefined.
- **Guest-physical address.** For a VM exit due to an EPT violation, an EPT misconfiguration, or an SPP-related event, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

If the EPT violation or EPT misconfiguration occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

28.2.2 Information for VM Exits Due to Vectored Events

Section 25.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT1, INT3, INTO, BOUND, UD0, UD1, and UD2); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs).¹ Such VM exits include those that occur on an attempt at a task switch that causes an exception before generating the VM exit due to the task switch that causes the VM exit.

The following items detail the use of these fields:

- **VM-exit interruption information** (format given in Table 25-18). The following items detail how this field is established for VM exits due to these events:
 - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 5 (privileged software exception), or 6 (software exception). Hardware exceptions comprise all exceptions except the following:
 - Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
 - Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)
- BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, and UD2 are hardware exceptions.
- Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).² If bit 11 is set to 1, the error code is placed in the VM-exit interruption error code (see below).
 - Bit 12 reports “NMI unblocking due to IRET”; see Section 28.2.3. The value of this bit is undefined if the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).
 - Bits 30:13 are always set to 0.
 - Bit 31 is always set to 1.

For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- VM-exit interruption error code.
 - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally through the IDT. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the IDT-vectoring information field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).
 - For other VM exits, the value of this field is undefined.

1. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

28.2.3 Information About NMI Unblocking Due to IRET

A VM exit may occur during execution of the IRET instruction for reasons including the following: faults, EPT violations, page-modification log-full events, or SPP-related events.

An execution of IRET that commences while non-maskable interrupts (NMIs) are blocked will unblock NMIs even if a fault or VM exit occurs; the state saved by such a VM exit will indicate that NMIs were not blocked.

VM exits for the reasons enumerated above provide more information to software by saving a bit called “NMI unblocking due to IRET.” This bit is defined if (1) either the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1; (2) the VM exit does not set the valid bit in the IDT-vectoring information field (see Section 28.2.4); and (3) the VM exit is not due to a double fault. In these cases, the bit is defined as follows:

- The bit is 1 if the VM exit resulted from a memory access as part of execution of the IRET instruction and one of the following holds:
 - The “virtual NMIs” VM-execution control is 0 and blocking by NMI (see Table 25-3) was in effect before execution of IRET.
 - The “virtual NMIs” VM-execution control is 1 and virtual-NMI blocking was in effect before execution of IRET.
- The bit is 0 for all other relevant VM exits.

For VM exits due to faults, NMI unblocking due to IRET is saved in bit 12 of the VM-exit interruption-information field (Section 28.2.2). For VM exits due to EPT violations, page-modification log-full events, and SPP-related events, NMI unblocking due to IRET is saved in bit 12 of the exit qualification (Section 28.2.1).

(Executions of IRET may also incur VM exits due to APIC accesses and EPT misconfigurations. These VM exits do not report information about NMI unblocking due to IRET.)

28.2.4 Information for VM Exits During Event Delivery

Section 25.9.3 defined fields containing information for VM exits that occur while delivering an event through the IDT and as a result of any of the following cases:¹

- A fault occurs during event delivery and causes a VM exit (because the bit associated with the fault is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. The VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 26.4.2).
- Event delivery causes an APIC-access VM exit (see Section 30.4).
- An EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that occurs during event delivery.
- Any of the above VM exits that occur during user-interrupt notification processing (see Section 7.5.2). Such VM exits will be treated as if they occurred during delivery of an external interrupt with the vector UINV.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 27.6.1.2).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the “NMI exiting” VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.

The following items detail the use of these fields:

1. This includes the case in which a VM exit occurs while delivering a software interrupt (INT n) through the 16-bit IVT (interrupt vector table) that is used in virtual-8086 mode with virtual-machine extensions (if RFLAGS.VM = CR4.VME = 1).

- IDT-vectoring information (format given in Table 25-19). The following items detail how this field is established for VM exits that occur during event delivery:
 - If the VM exit occurred during delivery of an exception, bits 7:0 receive the exception vector (at most 31). If the VM exit occurred during delivery of an NMI, bits 7:0 are set to 2. If the VM exit occurred during delivery of an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to indicate the type of event that was being delivered when the VM exit occurred: 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 4 (software interrupt), 5 (privileged software interrupt), or 6 (software exception).

Hardware exceptions comprise all exceptions except the following:¹

- Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
- Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)

BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, and UD2 are hardware exceptions.

- Bit 11 is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).² If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).
- Bit 12 is undefined.
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- IDT-vectoring error code.
 - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
 - For other VM exits, the value of this field is undefined.

28.2.5 Information for VM Exits Due to Instruction Execution

Section 25.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
 - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 26.1.2) or based on the settings of VM-execution controls (see Section 26.1.3): CLTS, CPUID, ENCLS, GETSEC, HLT, IN, INS, INVLD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LOADIWKEY, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, PCONFIG, RDMR, RDPIC, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, TPAUSE, UMWAIT,

1. In the following items, INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WBNOINVD, WRMSR, XRSTORS, XSETBV, and XSAVES.¹

- For VM exits due to software exceptions (those generated by executions of INT3 or INTO) or privileged software exceptions (those generated by executions of INT1).
- For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, or software exception.
- For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:
 - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
 - An exit qualification indicating a task gate in the IDT and an IDT-vectoring information field indicating that the task gate was encountered during delivery of a software interrupt, privileged software exception, or software exception.
- For APIC-access VM exits and for VM exits caused by EPT violations, page-modification log-full events, and SPP-related events encountered during delivery of a software interrupt, privileged software exception, or software exception.²
- For VM exits due to executions of VMFUNC that fail because one of the following is true:
 - EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 26.5.6.2).
 - EAX = 0 and either ECX ≥ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 26.5.6.3).

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 27.6.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

If the VM exit occurred in enclave mode, this field is cleared (none of the previous items apply).

Table 28-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS

Bit Position(s)	Content
6:0	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
14:10	Undefined.

1. This item applies only to fault-like VM exits. It does not apply to trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1 or to those following executions of the WRMSR instruction when the “virtualize x2APIC mode” VM-execution control is 1.

2. The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 30.4.6) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.

Table 28-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS (Contd.)

Bit Position(s)	Content
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for VM exits due to execution of INS.
31:18	Undefined.

- VM-exit instruction information.** For VM exits due to attempts to execute INS, INVEPT, INVPCID, INVVPID, LIDT, LGDT, LLDT, LOADIWKEY, LTR, OUTS, RDRAND, RDSEED, SIDT, SGDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, or XSAVES, this field receives information about the instruction that caused the VM exit. The format of the field depends on the identity of the instruction causing the VM exit:
 - For VM exits due to attempts to execute INS or OUTS, the field has the format is given in Table 28-8.¹
 - For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 28-9.
 - For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 28-10.
 - For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 28-11.
 - For VM exits due to attempts to execute RDRAND, RDSEED, TPAUSE, or UMWAIT, the field has the format is given in Table 28-12.
 - For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 28-13.
 - For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 28-14.
 - For VM exits due to attempts to execute LOADIWKEY, the field has the format is given in Table 28-15.

For all other VM exits, the field is undefined, unless the VM exit occurred in enclave mode, in which case the field is cleared.
- I/O RCX, I/O RSI, I/O RDI, I/O RIP.** These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 32.15.2.3. Note that, if the VM exit occurred in enclave mode, these fields are all cleared.

Table 28-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.

1. The format of the field was undefined for these VM exits on the first processors to support the virtual-machine extensions. Software can determine whether the format specified in Table 28-8 is used by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

Table 28-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVPID (Contd.)

Bit Position(s)	Content
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for memory instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Reg2 (same encoding as IndexReg above)

Table 28-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.

Table 28-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)

Bit Position(s)	Content
10	Cleared to 0.
11	Operand size: 0: 16-bit 1: 32-bit Undefined for VM exits from 64-bit mode.
14:12	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
29:28	Instruction identity: 0: SGDT 1: SIDT 2: LGDT 3: LIDT
31:30	Undefined.

Table 28-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.

Table 28-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR (Contd.)

Bit Position(s)	Content
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
29:28	Instruction identity: 0: SLDT 1: STR 2: LLDT 3: LTR
31:30	Undefined.

Table 28-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND, RDSEED, TPAUSE, and UMWAIT

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (destination for RDRAND and RDSEED; source for TPAUSE and UMWAIT): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture)
10:7	Undefined.
12:11	Operand size: 0: 16-bit 1: 32-bit 2: 64-bit The value 3 is not used.
31:13	Undefined.

Table 28-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.

Table 28-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES (Contd.)

Bit Position(s)	Content
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Undefined.

Table 28-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.

Table 28-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE (Contd.)

Bit Position(s)	Content
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above)

Table 28-15. Format of the VM-Exit Instruction-Information Field as Used for LOADIWKEY

Bit Position(s)	Content
2:0	Undefined.
6:3	Reg1: identifies the first XMM register operand (XMM0–XMM15; values 8–15 are used only on processors that support Intel 64 architecture).
30:7	Undefined.
31:28	Reg2: identifies the second XMM register operand (see above).

28.3 SAVING GUEST STATE

VM exits save certain components of processor state into corresponding fields in the guest-state area of the VMCS (see Section 25.4). On processors that support Intel 64 architecture, the full value of each natural-width field (see Section 25.11.2) is saved regardless of the mode of the logical processor before and after the VM exit.

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 28.1 for a discussion of which architectural updates occur at that time.

Section 28.3.1 through Section 28.3.4 provide details for how various components of processor state are saved. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

28.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs are saved as follows:

- The contents of CR0, CR3, CR4, and the IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP MSRs are saved into the corresponding fields. Bits 63:32 of the IA32_SYSENTER_CS MSR are not saved. On

processors that do not support Intel 64 architecture, bits 63:32 of the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are not saved.

- If the “save debug controls” VM-exit control is 1, the contents of DR7 and the IA32_DEBUGCTL MSR are saved into the corresponding fields. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always saved data into these fields.
- If the “save IA32_PAT” VM-exit control is 1, the contents of the IA32_PAT MSR are saved into the corresponding field.
- If the “save IA32_EFER” VM-exit control is 1, the contents of the IA32_EFER MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control, the contents of the IA32_BNDCFGS MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control, the contents of the IA32_RTIT_CTL MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are saved into the corresponding fields. On processors that do not support Intel 64 architecture, bits 63:32 of these MSRs are not saved.
- If the processor supports either the 1-setting of the “load guest IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control, the contents of the IA32_LBR_CTL MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load PKRS” VM-entry control, the contents of the IA32_PKRS MSR are saved into the corresponding field.
- If a processor supports user interrupts, every VM exit saves UINV into the guest UINV field in the VMCS (bits 15:8 of the field are cleared).
- If the “save IA32_PERF_GLOBAL_CTL” VM-exit control is 1, the contents of the IA32_PERF_GLOBAL_CTL MSR are saved into the corresponding field.
- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 32.15.2.

28.3.2 Saving Segment Registers and Descriptor-Table Registers

For each segment register (CS, SS, DS, ES, FS, GS, LDTR, or TR), the values saved for the base-address, segment-limit, and access rights are based on whether the register was unusable (see Section 25.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
 - CS.
 - The base-address and segment-limit fields are saved.
 - The L, D, and G bits are saved in the access-rights field.
 - SS.
 - DPL is saved in the access-rights field.
 - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
 - DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
 - FS and GS. The base-address field is saved.
 - LDTR. The value saved for the base address is always canonical.
- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.

- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

28.3.3 Saving RIP, RSP, RFLAGS, and SSP

The contents of the RIP, RSP, RFLAGS, and SSP (shadow-stack pointer) registers are saved as follows:

- The value saved in the RIP field is determined by the nature and cause of the VM exit:
 - If the VM exit occurred in enclave mode, the value saved is the AEP of interrupted enclave thread (the remaining items do not apply).
 - If the VM exit occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.
 - If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
 - If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
 - If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 28.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,¹ or into the old task-state segment had the event been delivered through a task gate).
 - If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.
 - If the VM exit is due to a software exception (due to an execution of INT3 or INTO) or a privileged software exception (due to an execution of INT1), the value saved references the INT3, INTO, or INT1 instruction that caused that exception.
 - Suppose that the VM exit is due to a task switch that was caused by execution of CALL, IRET, or JMP or by execution of a software interrupt (INT *n*), software exception (due to execution of INT3 or INTO), or privileged software exception (due to execution of INT1) that encountered a task gate in the IDT. The value saved references the instruction that caused the task switch (CALL, IRET, JMP, INT *n*, INT3, INTO, INT1).
 - Suppose that the VM exit is due to a task switch that was caused by a task gate in the IDT that was encountered for any reason except the direct access by a software interrupt or software exception. The value saved is that which would have been saved in the old task-state segment had the task switch completed normally.
 - If the VM exit is due to an execution of MOV to CR8 or WRMSR that reduced the value of bits 7:4 of VTPR (see Section 30.1.1) below that of TPR threshold VM-execution control field (see Section 30.1.2), the value saved references the instruction following the MOV to CR8 or WRMSR.
 - If the VM exit was caused by APIC-write emulation (see Section 30.4.3.2) that results from an APIC access as part of instruction execution, the value saved references the instruction following the one whose execution caused the APIC-write emulation.
- The contents of the RSP register are saved into the RSP field.
- With the exception of the resume flag (RF; bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. RFLAGS.RF is saved as follows:
 - If the VM exit occurred in enclave mode, the value saved is 0 (the remaining items do not apply).

1. The reference here is to the full value of RIP before any truncation that would occur had the stack width been only 32 bits or 16 bits.

- If the VM exit is caused directly by an event that would normally be delivered through the IDT, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate¹ or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.
- If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
- If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
- If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.²
- For APIC-access VM exits and for VM exits caused by EPT violations, EPT misconfigurations, page-modification log-full events, or SPP-related events, the value saved depends on whether the VM exit occurred during delivery of an event through the IDT:
 - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 28.2.4), the value saved is 1.
 - If the VM exit stored 1 for bit 31 for IDT-vectoring information field (because the VM exit did occur during delivery of an event through the IDT), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered through the IDT (see above).
- For all other VM exits, the value saved is the value RFLAGS.RF had before the VM exit occurred.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the SSP register are saved into the SSP field.

28.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

- The activity-state field is saved with the logical processor’s activity state before the VM exit.³ See Section 28.1 for details of how events leading to a VM exit may affect the activity state. If the VM exit occurred during user-interrupt notification processing (see Section 7.5.2) and the logical processor would have entered the HLT state following user-interrupt notification processing, the saved activity state is “HLT”.
- The interruptibility-state field is saved to reflect the logical processor’s interruptibility before the VM exit.
 - See Section 28.1 for details of how events leading to a VM exit may affect this state.
 - VM exits that end outside system-management mode (SMM) save bit 2 (blocking by SMI) as 0 regardless of the state of such blocking before the VM exit.
 - Bit 3 (blocking by NMI) is treated specially if the “virtual NMIs” VM-execution control is 1. In this case, the value saved for this field does not indicate the blocking of NMIs but rather the state of virtual-NMI blocking.
 - Bit 4 (enclave interruption) is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

-
1. The reference here is to the full value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.
 2. This is true even if RFLAGS.RF was 1 before the instruction was executed. If, in response to such a VM exit, a VM monitor re-enters the guest to re-execute the instruction that caused the VM exit (for example, after clearing the VM-execution control that caused the VM exit), the instruction may encounter a code breakpoint that has already been processed. A VM monitor can avoid this by setting the guest value of RFLAGS.RF to 1 before resuming guest software.
 3. If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit that is incident to delivery of an event injected by VM entry leaves this bit unmodified.

- The pending debug exceptions field is saved as clear for all VM exits except the following:
 - A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
 - A VM exit with basic exit reason “TPR below threshold”,¹ “virtualized EOI”, “APIC write”, “monitor trap flag,” or “bus-lock detected.”
 - A VM exit due to trace-address pre-translation (TAPT; see Section 26.5.4) or due to accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 20.9.5). Such VM exits can have basic exit reason “APIC access,” “EPT violation,” “EPT misconfiguration,” “page-modification log full,” or “SPP-related event.” When due to TAPT or PEBS, these VM exits (with the exception of those due to EPT misconfigurations) set bit 16 of the exit qualification, indicating that they are asynchronous to instruction execution and not part of event delivery.
 - VM exits that are not caused by debug exceptions and that occur while there is MOV-SS blocking of debug exceptions.

For VM exits that do not clear the field, the value saved is determined as follows:

- Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.
- Suppose that a VM exit is due to an INIT signal, a machine-check exception, or an SMI; or that a VM exit has basic exit reason “TPR below threshold” or “monitor trap flag.” In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit.

If the VM exit occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 27.7.3). Otherwise, the following items apply:

- Bit 12 (enabled breakpoint) is set to 1 in any of the following cases:
 - If there was at least one matched data or I/O breakpoint that was enabled in DR7.
 - If it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 27.7.3) and the VM exit occurred before those exceptions were either delivered or lost.
 - If the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled (see Section 16.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). (This does not apply to VM exits with basic exit reason “monitor trap flag.”)
 - If a bus lock was asserted while CPL > 0 and OS bus-lock detection was enabled.

In other cases, bit 12 is cleared to 0.

- Bit 14 (BS) is set if RFLAGS.TF = 1 in either of the following cases:
 - IA32_DEBUGCTL.BTF = 0 and the cause of a pending debug exception was the execution of a single instruction.
 - IA32_DEBUGCTL.BTF = 1 and the cause of a pending debug exception was a taken branch.
- Bit 16 (RTM) is set if a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions had been enabled. (This does not apply to VM exits with basic exit reason “monitor trap flag.”)
- Suppose that a VM exit is due to another reason (but not a debug exception) and occurs while there is MOV-SS blocking of debug exceptions. In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit. If the VM exit occurs immediately after VM entry (no instructions were executed in VMX non-root operation), the value saved may match that which was loaded on VM entry (see Section 27.7.3). Otherwise, the following items apply:

1. This item includes VM exits that occur as a result of certain VM entries (Section 27.7.7).

- Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 27.7.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
 - The setting of bit 14 (BS) is implementation-specific. However, it is not set if RFLAGS.TF = 0 or IA32_DEBUGCTL.BTF = 1.
- The reserved bits in the field are cleared.
- If the “save VMX-preemption timer value” VM-exit control is 1, the value of timer is saved into the VMX-preemption timer-value field. This is the value loaded from this field on VM entry as subsequently decremented (see Section 26.5.1). VM exits due to timer expiration save the value 0. Other VM exits may also save the value 0 if the timer expired during VM exit. (If the “save VMX-preemption timer value” VM-exit control is 0, VM exit does not modify the value of the VMX-preemption timer-value field.)
 - If the logical processor supports the 1-setting of the “enable EPT” VM-execution control, values are saved into the four (4) PDPTE fields as follows:
 - If the “enable EPT” VM-execution control is 1 and the logical processor was using PAE paging at the time of the VM exit, the PDPTE values currently in use are saved:¹
 - The values saved into bits 11:9 of each of the fields is undefined.
 - If the value saved into one of the fields has bit 0 (present) clear, the value saved into bits 63:1 of that field is undefined. That value need not correspond to the value that was loaded by VM entry or to any value that might have been loaded in VMX non-root operation.
 - If the value saved into one of the fields has bit 0 (present) set, the value saved into bits 63:12 of the field is a guest-physical address.
 - If the “enable EPT” VM-execution control is 0 or the logical processor was not using PAE paging at the time of the VM exit, the values saved are undefined.

28.4 SAVING MSRS

After processor state is saved to the guest-state area, values of MSRs may be stored into the VM-exit MSR-store area (see Section 25.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMBASE is an MSR that can be read only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if they can normally be read by RDMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 of the entry are not all 0.
- An attempt to read the MSR indexed by bits 31:0 would cause a general-protection exception if executed via RDMSR with CPL = 0.

A VMX abort occurs if processing fails for any entry. See Section 28.7.

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the “enable EPT” VM-execution control were 0. See Section 25.6.2.

28.5 LOADING HOST STATE

Processor state is updated on VM exits in the following ways:

- Some state is loaded from or otherwise determined by the contents of the host-state area.
- Some state is determined by VM-exit controls.
- Some state is established in the same way on every VM exit.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order.

On processors that support Intel 64 architecture, the full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of the logical processor before and after the VM exit.

The loading of host state is detailed in Section 28.5.1 to Section 28.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

A logical processor is in IA-32e mode after a VM exit only if the “host address-space size” VM-exit control is 1. If the logical processor was in IA-32e mode before the VM exit and this control is 0, a VMX abort occurs. See Section 28.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 28.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 28.6). This loading occurs only after the state loading described in this section.

28.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
 - The following bits are not modified:
 - For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 24.8).¹
 - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor’s physical-address width (they are cleared to 0).² (This item applies only to processors that support Intel 64 architecture.)
 - For CR4, any bits that are fixed in VMX operation (see Section 24.8).
 - CR4.PAE is set to 1 if the “host address-space size” VM-exit control is 1.
 - CR4.PCIDE is set to 0 if the “host address-space size” VM-exit control is 0.
- DR7 is set to 400H.
- If the “clear UINV” VM-exit control is 1, VM exit clears UINV.
- The following MSRs are established as follows:
 - The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since that field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP fields, respectively.

If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

1. Bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.

2. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit N-1.¹

- The following steps are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 28.5.2).
 - The LMA and LME bits in the IA32_EFER MSR are each loaded with the setting of the “host address-space size” VM-exit control.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-exit control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field. Bits that are reserved in that MSR are maintained with their reserved values.
- If the “load IA32_PAT” VM-exit control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field. Bits that are reserved in that MSR are maintained with their reserved values.
- If the “load IA32_EFER” VM-exit control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field. Bits that are reserved in that MSR are maintained with their reserved values.
- If the “clear IA32_BNDCFGS” VM-exit control is 1, the IA32_BNDCFGS MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
- If the “clear IA32_RTIT_CTL” VM-exit control is 1, the IA32_RTIT_CTL MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
- If the “load CET” VM-exit control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR fields, respectively.

If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit N-1.

- If the “load PKRS” VM-exit control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field. Bits 63:32 of that MSR are maintained with zeroes.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 28.6.

28.5.2 Loading Host Segment and Descriptor-Table Registers

Each of the registers CS, SS, DS, ES, FS, GS, and TR is loaded as follows (see below for the treatment of LDTR):

- The selector is loaded from the selector field. The segment is unusable if its selector is loaded with zero. The checks specified in Section 27.2.3 limit the selector values that may be loaded. In particular, CS and TR are never loaded with zero and are thus never unusable. SS can be loaded with zero only on processors that support Intel 64 architecture and only if the VM exit is to 64-bit mode (64-bit mode allows use of segments marked unusable).
- The base address is set as follows:
 - CS. Cleared to zero.
 - SS, DS, and ES. Undefined if the segment is unusable; otherwise, cleared to zero.
 - FS and GS. Undefined (but, on processors that support Intel 64 architecture, canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit N-1.¹ The values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSR.

- TR. Loaded from the host-state area. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit N-1.
- The segment limit is set as follows:
 - CS. Set to FFFFFFFFH (corresponding to a descriptor limit of FFFFFFFH and a G-bit setting of 1).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.
 - TR. Set to 00000067H.
- The type field and S bit are set as follows:
 - CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
 - TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:
 - CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
 - CS, TR. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32_EFER.LMA (see Section 28.5.1), no VM exit is ever to compatibility mode (which requires IA32_EFER.LMA = 1 and CS.L = 0).
- D/B.
 - CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
 - SS. Set to 1.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.
- G.
 - CS. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.

The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined (although the base address is always canonical).

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N of each base address is set to the value of bit N-1 of that base address. The GDTR and IDTR limits are each set to FFFFH.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

28.5.3 Loading Host RIP, RSP, RFLAGS, and SSP

RIP and RSP are loaded from the RIP field and the RSP field, respectively. RFLAGS is cleared, except bit 1, which is always set.

If the “load CET” VM-exit control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

28.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 4.4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.¹ When in PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs and, if they are valid, loads them into the processor (into internal, non-architectural registers).

A VM exit is to a VMM that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the host-state area of the VMCS; and (2) the “host address-space size” VM-exit control is 0. Such a VM exit may check the validity of the PDPTEs referenced by the CR3 field in the host-state area of the VMCS. Such a VM exit must check their validity if either (1) PAE paging was not in use before the VM exit; or (2) the value of CR3 is changing as a result of the VM exit. A VM exit to a VMM that does not use PAE paging must not check the validity of the PDPTEs.

A VM exit that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use. If MOV to CR3 would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs (see Section 28.7). If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTEs are loaded into the processor as would MOV to CR3, using the value of CR3 being load by the VM exit.

28.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.
- Event blocking is affected as follows:
 - There is no blocking by STI or by MOV SS after a VM exit.
 - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 25-3). Other VM exits do not affect blocking by NMI. (See Section 28.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 29.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM exits invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).
- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

28.5.6 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 9.10.4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A. VM exits clear any address-range monitoring that may be in effect.

28.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 25.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101H (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM exits for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, "Model-Specific Registers (MSRs)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.¹

If processing fails for any entry, a VMX abort occurs. See Section 28.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM exit, the logical processor does not use any translations that were cached before the transition.

28.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shut-down state as described below.

A VMX abort does not modify the VMCS data in the VMCS region of any active VMCS. The contents of these data are thus suspect after the VMX abort.

On a VMX abort, a logical processor saves a nonzero 32-bit VMX-abort indicator field at byte offset 4 in the VMCS region of the VMCS whose misconfiguration caused the failure (see Section 25.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 28.4).
2. Host checking of the page-directory-pointer-table entries (PDPTs) failed (see Section 28.5.4).
3. The current VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the logical processor cannot complete the VM exit properly.
4. There was a failure on loading host MSRs (see Section 28.6).
5. There was a machine-check event during VM exit (see Section 28.8).

1. Note the following about processors that support Intel 64 architecture. If CRO.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CRO.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

6. The logical processor was in IA-32e mode before the VM exit and the “host address-space size” VM-exit control was 0 (see Section 28.5).

Some of these causes correspond to failures during the loading of state from the host-state area. Because the loading of such state may be done in any order (see Section 28.5) a VM exit that might lead to a VMX abort for multiple reasons (for example, the current VMCS may be corrupt and the host PDPTes might not be properly configured). In such cases, the VMX-abort indicator could correspond to any one of those reasons.

A logical processor never reads the VMX-abort indicator in a VMCS region and writes it only with one of the non-zero values mentioned above. The VMX-abort indicator allows software on one logical processor to diagnose the VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:¹

- If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000DH, indicating “VMX abort.” See *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide*.
- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the **VMX-abort shutdown state**. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine-check events; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

28.8 MACHINE-CHECK EVENTS DURING VM EXIT

If a machine-check event occurs during VM exit, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM exit:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:²
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- The machine-check event is handled after VM exit completes:
 - If the VM exit ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM exit ends with CR4.MCE = 1, a machine-check exception (#MC) is delivered through the host IDT.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.

2. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.

VM EXITS

- A VMX abort is generated (see Section 28.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for “machine-check event during VM exit.”

The first option is not used if the machine-check event occurs after any host state has been loaded. The second option is used only if VM entry is able to load all host state.

28.9 USER-INTERRUPT RECOGNITION AFTER VM EXIT

A VM exit results in recognition of a pending user interrupt if it completes with $CR4.UINTR = IA32_EFER.LMA = 1$ and with $UIRR \neq 0$; otherwise, no pending user interrupt is recognized.

16. Updates to Chapter 32, Volume 3C

Change bars and green text show changes to Chapter 32 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Update to Section 32.15.6.1, "Initial Checks."

This chapter describes aspects of IA-64 and IA-32 architecture used in system management mode (SMM).

SMM provides an alternate operating environment that can be used to monitor and manage various system resources for more efficient energy usage, to control system hardware, and/or to run proprietary code. It was introduced into the IA-32 architecture in the Intel386 SL processor (a mobile specialized version of the Intel386 processor). It is also available in the Pentium M, Pentium 4, Intel Xeon, P6 family, and Pentium and Intel486 processors (beginning with the enhanced versions of the Intel486 SL and Intel486 processors).

32.1 SYSTEM MANAGEMENT MODE OVERVIEW

SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary OEM-designed code. It is intended for use only by system firmware, not by applications software or general-purpose systems software. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that operates transparently to the operating system or executive and software applications.

When SMM is invoked through a system management interrupt (SMI), the processor saves the current state of the processor (the processor's context), then switches to a separate operating environment defined by a new address space. The system management software executive (SMI handler) starts execution in that environment, and the critical code and data of the SMI handler reside in a physical memory region (SMRAM) within that address space. While in SMM, the processor executes SMI handler code to perform operations such as powering down unused disk drives or monitors, executing proprietary code, or placing the whole system in a suspended state. When the SMI handler has completed its operations, it executes a resume (RSM) instruction. This instruction causes the processor to reload the saved context of the processor, switch back to protected or real mode, and resume executing the interrupted application or operating-system program or task.

The following SMM mechanisms make it transparent to applications programs and operating systems:

- The only way to enter SMM is by means of an SMI.
- The processor executes SMM code in a separate address space that can be made inaccessible from the other operating modes.
- Upon entering SMM, the processor saves the context of the interrupted program or task.
- All interrupts normally handled by the operating system are disabled upon entry into SMM.
- The RSM instruction can be executed only in SMM.

Section 32.3 describes transitions into and out of SMM. The execution environment after entering SMM is in real-address mode with paging disabled ($CR0.PE = CR0.PG = 0$). In this initial execution environment, the SMI handler can address up to 4 GBytes of memory and can execute all I/O and system instructions. Section 32.5 describes in detail the initial SMM execution environment for an SMI handler and operation within that environment. The SMI handler may subsequently switch to other operating modes while remaining in SMM.

NOTES

Software developers should be aware that, even if a logical processor was using the physical-address extension (PAE) mechanism (introduced in the P6 family processors) or was in IA-32e mode before an SMI, this will not be the case after the SMI is delivered. This is because delivery of an SMI disables paging (see Table 32-4). (This does not apply if the dual-monitor treatment of SMIs and SMM is active; see Section 32.15.)

32.1.1 System Management Mode and VMX Operation

Traditionally, SMM services system management interrupts and then resumes program execution (back to the software stack consisting of executive and application software; see Section 32.2 through Section 32.13).

A virtual machine monitor (VMM) using VMX can act as a host to multiple virtual machines and each virtual machine can support its own software stack of executive and application software. On processors that support VMX, virtual-machine extensions may use system-management interrupts (SMIs) and system-management mode (SMM) in one of two ways:

- **Default treatment.** System firmware handles SMIs. The processor saves architectural states and critical states relevant to VMX operation upon entering SMM. When the firmware completes servicing SMIs, it uses RSM to resume VMX operation.
- **Dual-monitor treatment.** Two VM monitors collaborate to control the servicing of SMIs: one VMM operates outside of SMM to provide basic virtualization in support for guests; the other VMM operates inside SMM (while in VMX operation) to support system-management functions. The former is referred to as **executive monitor**, the latter **SMM-transfer monitor (STM)**.¹

The default treatment is described in Section 32.14, “Default Treatment of SMIs and SMM with VMX Operation and SMX Operation.” Dual-monitor treatment of SMM is described in Section 32.15, “Dual-Monitor Treatment of SMIs and SMM.”

32.2 SYSTEM MANAGEMENT INTERRUPT (SMI)

The only way to enter SMM is by signaling an SMI through the SMI# pin on the processor or through an SMI message received through the APIC bus. The SMI is a nonmaskable external interrupt that operates independently from the processor’s interrupt- and exception-handling mechanism and the local APIC. The SMI takes precedence over an NMI and a maskable interrupt. SMM is non-reentrant; that is, the SMI is disabled while the processor is in SMM.

NOTES

In the Pentium 4, Intel Xeon, and P6 family processors, when a processor that is designated as an application processor during an MP initialization sequence is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However if a SMI is received while an application processor is in the wait for SIPI mode, the SMI will be pended. The processor then responds on receipt of a SIPI by immediately servicing the pended SMI and going into SMM before handling the SIPI.

An SMI may be blocked for one instruction following execution of STI, MOV to SS, or POP into SS.

32.3 SWITCHING BETWEEN SMM AND THE OTHER PROCESSOR OPERATING MODES

Figure 2-3 shows how the processor moves between SMM and the other processor operating modes (protected, real-address, and virtual-8086). Signaling an SMI while the processor is in real-address, protected, or virtual-8086 modes always causes the processor to switch to SMM. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

32.3.1 Entering SMM

The processor always handles an SMI on an architecturally defined “interruptible” point in program execution (which is commonly at an IA-32 architecture instruction boundary). When the processor receives an SMI, it waits for all instructions to retire and for all stores to complete. The processor then saves its current context in SMRAM (see Section 32.4), enters SMM, and begins to execute the SMI handler.

1. The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

Upon entering SMM, the processor signals external hardware that SMI handling has begun. The signaling mechanism used is implementation dependent. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is asserted each time a bus transaction is generated while the processor is in SMM. For the Pentium and Intel486 processors, the SMIACK# pin is asserted.

An SMI has a greater priority than debug exceptions and external interrupts. Thus, if an NMI, maskable hardware interrupt, or a debug exception occurs at an instruction boundary along with an SMI, only the SMI is handled. Subsequent SMI requests are not acknowledged while the processor is in SMM. The first SMI interrupt request that occurs while the processor is in SMM (that is, after SMM has been acknowledged to external hardware) is latched and serviced when the processor exits SMM with the RSM instruction. The processor will latch only one SMI while in SMM.

See Section 32.5 for a detailed description of the execution environment when in SMM.

32.3.2 Exiting From SMM

The only way to exit SMM is to execute the RSM instruction. The RSM instruction is only available to the SMI handler; if the processor is not in SMM, attempts to execute the RSM instruction result in an invalid-opcode exception (#UD) being generated.

The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. The processor then returns an SMIACK transaction on the system bus and returns program control back to the interrupted program.

NOTE

On processors that support the shadow-stack feature, RSM loads the SSP register from the state save image in SMRAM (see Table 32-3). The value is made canonical by sign-extension before loading it into SSP.

Upon successful completion of the RSM instruction, the processor signals external hardware that SMM has been exited. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is no longer generated on bus cycles. For the Pentium and Intel486 processors, the SMIACK# pin is deserted.

If the processor detects invalid state information saved in the SMRAM, it enters the shutdown state and generates a special bus cycle to indicate it has entered shutdown state. Shutdown happens only in the following situations:

- A reserved bit in control register CR4 is set to 1 on a write to CR4. This error should not happen unless SMI handler code modifies reserved areas of the SMRAM saved state map (see Section 32.4.1). CR4 is saved in the state map in a reserved location and cannot be read or modified in its saved state.
- An illegal combination of bits is written to control register CR0, in particular PG set to 1 and PE set to 0, or NW set to 1 and CD set to 0.
- CR4.PCIDE would be set to 1 and IA32_EFER.LMA to 0.
- (For the Pentium and Intel486 processors only.) If the address stored in the SMBASE register when an RSM instruction is executed is not aligned on a 32-KByte boundary. This restriction does not apply to the P6 family processors.
- CR4.CET would be set to 1 and CR0.WP to 0.

In the shutdown state, Intel processors stop executing instructions until a RESET#, INIT# or NMI# is asserted. While Pentium family processors recognize the SMI# signal in shutdown state, P6 family and Intel486 processors do not. Intel does not support using SMI# to recover from shutdown states for any processor family; the response of processors in this circumstance is not well defined. On Pentium 4 and later processors, shutdown will inhibit INTR and A20M but will not change any of the other inhibits. On these processors, NMIs will be inhibited if no action is taken in the SMI handler to uninhibit them (see Section 32.8).

If the processor is in the HALT state when the SMI is received, the processor handles the return from SMM slightly differently (see Section 32.10). Also, the SMBASE address can be changed on a return from SMM (see Section 32.11).

32.4 SMRAM

Upon entering SMM, the processor switches to a new address space. Because paging is disabled upon entering SMM, this initial address space maps all memory accesses to the low 4 GBytes of the processor's physical address space. The SMI handler's critical code and data reside in a memory region referred to as system-management RAM (SMRAM). The processor uses a pre-defined region within SMRAM to save the processor's pre-SMI context. SMRAM can also be used to store system management information (such as the system configuration and specific information about powered-down devices) and OEM-specific information.

The default SMRAM size is 64 KBytes beginning at a base physical address in physical memory called the SMBASE (see Figure 32-1). The SMBASE default value following a hardware reset is 30000H. The processor looks for the first instruction of the SMI handler at the address [SMBASE + 8000H]. It stores the processor's state in the area from [SMBASE + FE00H] to [SMBASE + FFFFH]. See Section 32.4.1 for a description of the mapping of the state save area.

The system logic is minimally required to decode the physical address range for the SMRAM from [SMBASE + 8000H] to [SMBASE + FFFFH]. A larger area can be decoded if needed. The size of this SMRAM can be between 32 KBytes and 4 GBytes.

The location of the SMRAM can be changed by changing the SMBASE value (see Section 32.11). It should be noted that all processors in a multiple-processor system are initialized with the same SMBASE value (30000H). Initialization software must sequentially place each processor in SMM and change its SMBASE so that it does not overlap those of other processors.

The actual physical location of the SMRAM can be in system memory or in a separate RAM memory. The processor generates an SMI acknowledge transaction (P6 family processors) or asserts the SMIACT# pin (Pentium and Intel486 processors) when the processor receives an SMI (see Section 32.3.1).

System logic can use the SMI acknowledge transaction or the assertion of the SMIACT# pin to decode accesses to the SMRAM and redirect them (if desired) to specific SMRAM memory. If a separate RAM memory is used for SMRAM, system logic should provide a programmable method of mapping the SMRAM into system memory space when the processor is not in SMM. This mechanism will enable start-up procedures to initialize the SMRAM space (that is, load the SMI handler) before executing the SMI handler during SMM.

32.4.1 SMRAM State Save Map

When an IA-32 processor that does not support Intel 64 architecture initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area begins at [SMBASE + 8000H + 7FFFH] and extends down to [SMBASE + 8000H + 7E00H]. Table 32-1 shows the state save map. The offset in column 1 is relative to the SMBASE value plus 8000H. Reserved spaces should not be used by software.

Some of the registers in the SMRAM state save area (marked YES in column 3) may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). An SMI handler should not rely on any values stored in an area that is marked as reserved.

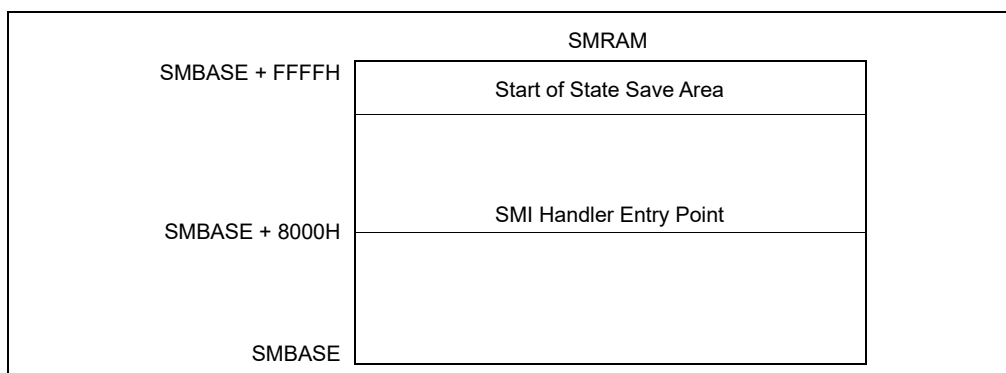


Figure 32-1. SMRAM Usage

Table 32-1. SMRAM State Save Map

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FFCH	CR0	No
7FF8H	CR3	No
7FF4H	EFLAGS	Yes
7FF0H	EIP	Yes
7FECH	EDI	Yes
7FE8H	ESI	Yes
7FE4H	EBP	Yes
7FE0H	ESP	Yes
7FDCH	EBX	Yes
7FD8H	EDX	Yes
7FD4H	ECX	Yes
7FD0H	EAX	Yes
7FCCH	DR6	No
7FC8H	DR7	No
7FC4H	TR ¹	No
7FC0H	Reserved	No
7FBCH	GS ¹	No
7FB8H	FS ¹	No
7FB4H	DS ¹	No
7FB0H	SS ¹	No
7FACH	CS ¹	No
7FA8H	ES ¹	No
7FA4H	I/O State Field, see Section 32.7	No
7FA0H	I/O Memory Address Field, see Section 32.7	No
7F9FH-7F03H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7E00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

The following registers are saved (but not readable) and restored upon exiting SMM:

- Control register CR4. (This register is cleared to all 0s when entering SMM).
- The hidden segment descriptor information stored in segment registers CS, DS, ES, FS, GS, and SS.

If an SMI request is issued for the purpose of powering down the processor, the values of all reserved locations in the SMM state save must be saved to nonvolatile memory.

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The x87 FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium processors) or test registers TR3 through TR7 (for the Pentium and Intel486 processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The microcode update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor, it must also save and restore them.

NOTES

A small subset of the MSRs (such as, the time-stamp counter and performance-monitoring counters) are not arbitrarily writable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers.

Operating system developers should be aware of this fact and ensure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

32.4.1.1 SMRAM State Save Map and Intel 64 Architecture

When the processor initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area on an Intel 64 processor at [SMBASE + 8000H + 7FFFH] and extends to [SMBASE + 8000H + 7C00H].

Support for Intel 64 architecture is reported by CPUID.80000001:EDX[29] = 1. The layout of the SMRAM state save map is shown in Table 32-3.

Additionally, the SMRAM state save map shown in Table 32-3 also applies to processors with the following CPUID signatures listed in Table 32-2, irrespective of the value in CPUID.80000001:EDX[29].

Table 32-2. Processor Signatures and 64-bit SMRAM State Save Map Format

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9xxx, Intel Core 2 Duo processors E8000, T9000,
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad, Intel Core 2 Extreme, Intel Core 2 Duo processors, Intel Pentium dual-core processors
06_1CH	45 nm Intel Atom® processors

Table 32-3. SMRAM State Save Map for Intel 64 Architecture

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FF8H	CR0	No
7FF0H	CR3	No
7FE8H	RFLAGS	Yes
7FE0H	IA32_EFER	Yes
7FD8H	RIP	Yes
7FD0H	DR6	No
7FC8H	DR7	No
7FC4H	TR SEL ¹	No
7FC0H	LDTR SEL ¹	No
7FBCH	GS SEL ¹	No
7FB8H	FS SEL ¹	No
7FB4H	DS SEL ¹	No
7FB0H	SS SEL ¹	No
7FACH	CS SEL ¹	No
7FA8H	ES SEL ¹	No
7FA4H	IO_MISC	No
7F9CH	IO_MEM_ADDR	No
7F94H	RDI	Yes
7F8CH	RSI	Yes
7F84H	RBP	Yes
7F7CH	RSP	Yes
7F74H	RBX	Yes
7F6CH	RDX	Yes
7F64H	RCX	Yes
7F5CH	RAX	Yes
7F54H	R8	Yes
7F4CH	R9	Yes
7F44H	R10	Yes
7F3CH	R11	Yes
7F34H	R12	Yes
7F2CH	R13	Yes
7F24H	R14	Yes
7F1CH	R15	Yes
7F1BH-7F04H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes

Table 32-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7EF7H - 7EE4H	Reserved	No
7EE0H	Setting of "enable EPT" VM-execution control	No
7ED8H	Value of EPTP VM-execution control field	No
7ED7H - 7ECC0H	Reserved	No
7EC8H	SSP	Yes
7EC7H - 7EA0H	Reserved	No
7E9CH	LDT Base (lower 32 bits)	No
7E98H	Reserved	No
7E94H	IDT Base (lower 32 bits)	No
7E90H	Reserved	No
7E8CH	GDT Base (lower 32 bits)	No
7E8BH - 7E48H	Reserved	No
7E40H	CR4 (64 bits)	No
7E3FH - 7DF0H	Reserved	No
7DE8H	IO_RIP	Yes
7DE7H - 7DDCH	Reserved	No
7DD8H	IDT Base (Upper 32 bits)	No
7DD4H	LDT Base (Upper 32 bits)	No
7DD0H	GDT Base (Upper 32 bits)	No
7DCFH - 7C00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

32.4.2 SMRAM Caching

An IA-32 processor does not automatically write back and invalidate its caches before entering SMM or before exiting SMM. Because of this behavior, care must be taken in the placement of the SMRAM in system memory and in the caching of the SMRAM to prevent cache incoherence when switching back and forth between SMM and protected mode operation. Any of the following three methods of locating the SMRAM in system memory will guarantee cache coherency.

- Place the SMRAM in a dedicated section of system memory that the operating system and applications are prevented from accessing. Here, the SMRAM can be designated as cacheable (WB, WT, or WC) for optimum processor performance, without risking cache incoherence when entering or exiting SMM.
- Place the SMRAM in a section of memory that overlaps an area used by the operating system (such as the video memory), but designate the SMRAM as uncacheable (UC). This method prevents cache access when in SMM to maintain cache coherency, but the use of uncacheable memory reduces the performance of SMM code.
- Place the SMRAM in a section of system memory that overlaps an area used by the operating system and/or application code, but explicitly flush (write back and invalidate) the caches upon entering and exiting SMM mode. This method maintains cache coherency, but incurs the overhead of two complete cache flushes.

For Pentium 4, Intel Xeon, and P6 family processors, a combination of the first two methods of locating the SMRAM is recommended. Here the SMRAM is split between an overlapping and a dedicated region of memory. Upon entering SMM, the SMRAM space that is accessed overlaps video memory (typically located in low memory). This SMRAM section is designated as UC memory. The initial SMM code then jumps to a second SMRAM section that is

located in a dedicated region of system memory (typically in high memory). This SMRAM section can be cached for optimum processor performance.

For systems that explicitly flush the caches upon entering SMM (the third method described above), the cache flush can be accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM (generally initiated by asserting the SMI# pin). The priorities of the FLUSH# and SMI# pins are such that the FLUSH# is serviced first. To guarantee this behavior, the processor requires that the following constraints on the interaction of FLUSH# and SMI# be met. In a system where the FLUSH# and SMI# pins are synchronous and the set up and hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first.

Upon leaving SMM (for systems that explicitly flush the caches), the WBINVD instruction should be executed prior to leaving SMM to flush the caches.

NOTES

In systems based on the Pentium processor that use the FLUSH# pin to write back and invalidate cache contents before entering SMM, the processor will prefetch at least one cache line in between when the Flush Acknowledge cycle is run and the subsequent recognition of SMI# and the assertion of SMIACK#.

It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive to the Pentium processor.

32.4.2.1 System Management Range Registers (SMRR)

SMI handler code and data stored by SMM code resides in SMRAM. The SMRR interface is an enhancement in Intel 64 architecture to limit cacheable reference of addresses in SMRAM to code running in SMM. The SMRR interface can be configured only by code running in SMM. Details of SMRR is described in Section 12.11.2.4.

32.5 SMI HANDLER EXECUTION ENVIRONMENT

Section 32.5.1 describes the initial execution environment for an SMI handler. An SMI handler may re-configure its execution environment to other supported operating modes. Section 32.5.2 discusses modifications an SMI handler can make to its execution environment. Section 32.5.3 discusses Control-flow Enforcement Technology (CET) interactions in the environment.

32.5.1 Initial SMM Execution Environment

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 32-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor in an environment similar to real-address mode. The differences between the SMM execution environment and the real-address mode execution environment are as follows:

- The addressable address space ranges from 0 to FFFFFFFFH (4 GBytes).
- The normal 64-KByte segment limit for real-address mode is increased to 4 GBytes.
- The default operand and address sizes are set to 16 bits, which restricts the addressable SMRAM address space to the 1-MByte real-address mode limit for native real-address-mode code. However, operand-size and address-size override prefixes can be used to access the address space beyond the 1-MByte.

Table 32-4. Processor Register Initialization in SMM

Register	Contents
General-purpose registers	Undefined
EFLAGS	00000002H
EIP	00008000H
CS selector	SMM Base shifted right 4 bits (default 3000H)

Table 32-4. Processor Register Initialization in SMM

CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	PE, EM, TS, and PG flags set to 0; others unmodified
CR4	Cleared to zero
DR6	Undefined
DR7	00000400H

- Near jumps and calls can be made to anywhere in the 4-GByte address space if a 32-bit operand-size override prefix is used. Due to the real-address-mode style of base-address formation, a far call or jump cannot transfer control to a segment with a base address of more than 20 bits (1 MByte). However, since the segment limit in SMM is 4 GBytes, offsets into a segment that go beyond the 1-MByte limit are allowed when using 32-bit operand-size override prefixes. Any program control transfer that does not have a 32-bit operand-size override prefix truncates the EIP value to the 16 low-order bits.
- Data and the stack can be located anywhere in the 4-GByte address space, but can be accessed only with a 32-bit address-size override if they are located above 1 MByte. As with the code segment, the base address for a data or stack segment cannot be more than 20 bits.

The value in segment register CS is automatically set to the default of 30000H for the SMBASE shifted 4 bits to the right; that is, 3000H. The EIP register is set to 8000H. When the EIP value is added to shifted CS value (the SMBASE), the resulting linear address points to the first instruction of the SMI handler.

The other segment registers (DS, SS, ES, FS, and GS) are cleared to 0 and their segment limits are set to 4 GBytes. In this state, the SMRAM address space may be treated as a single flat 4-GByte linear address space. If a segment register is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base (hidden part of the segment register). The limits and attributes are not modified.

Maskable hardware interrupts, exceptions, NMI interrupts, SMI interrupts, A20M interrupts, single-step traps, breakpoint traps, and INIT operations are inhibited when the processor enters SMM. Maskable hardware interrupts, exceptions, single-step traps, and breakpoint traps can be enabled in SMM if the SMM execution environment provides and initializes an interrupt table and the necessary interrupt and exception handlers (see Section 32.6).

32.5.2 SMI Handler Operating Mode Switching

Within SMM, an SMI handler may change the processor's operating mode (e.g., to enable PAE paging, enter 64-bit mode, etc.) after it has made proper preparation and initialization to do so. For example, if switching to 32-bit protected mode, the SMI handler should follow the guidelines provided in Chapter 10, "Processor Management and Initialization." If the SMI handler does wish to change operating mode, it is responsible for executing the appropriate mode-transition code after each SMI.

It is recommended that the SMI handler make use of all means available to protect the integrity of its critical code and data. In particular, it should use the system-management range register (SMRR) interface if it is available (see Section 11.11.2.4). The SMRR interface can protect only the first 4 GBytes of the physical address space. The SMI handler should take that fact into account if it uses operating modes that allow access to physical addresses beyond that 4-GByte limit (e.g., PAE paging or 64-bit mode).

Execution of the RSM instruction restores the pre-SMI processor state from the SMRAM state-state map (see Section 32.4.1) into which it was stored when the processor entered SMM. (The SMBASE field in the SMRAM state-state map does not determine the state following RSM but rather the initial environment following the next entry to SMM.) Any required change to operating mode is performed by the RSM instruction; there is no need for the SMI handler to change modes explicitly prior to executing RSM.

32.5.3 Control-flow Enforcement Technology Interactions

On processors that support CET shadow stacks, when the processor enters SMM, the processor saves the SSP register to the SMRAM state save area (see Table 32-3) and clears CR4.CET to 0. Thus, the initial execution environment of the SMI handler has CET disabled and all of the CET state of the interrupted program is still in the machine. An SMM that uses CET is required to save the interrupted program's CET state and restore the CET state prior to exiting SMM.

32.6 EXCEPTIONS AND INTERRUPTS WITHIN SMM

When the processor enters SMM, all hardware interrupts are disabled in the following manner:

- The IF flag in the EFLAGS register is cleared, which inhibits maskable hardware interrupts from being generated.
- The TF flag in the EFLAGS register is cleared, which disables single-step traps.
- Debug register DR7 is cleared, which disables breakpoint traps. (This action prevents a debugger from accidentally breaking into an SMI handler if a debug breakpoint is set in normal address space that overlays code or data in SMRAM.)
- NMI, SMI, and A20M interrupts are blocked by internal SMM logic. (See Section 32.8 for more information about how NMIs are handled in SMM.)

Software-invoked interrupts and exceptions can still occur, and maskable hardware interrupts can be enabled by setting the IF flag. Intel recommends that SMM code be written in so that it does not invoke software interrupts (with the INT *n*, INTO, INT1, INT3, or BOUND instructions) or generate exceptions.

If the SMI handler requires interrupt and exception handling, an SMM interrupt table and the necessary exception and interrupt handlers must be created and initialized from within SMM. Until the interrupt table is correctly initialized (using the LIDT instruction), exceptions and software interrupts will result in unpredictable processor behavior.

The following restrictions apply when designing SMM interrupt and exception-handling facilities:

- The interrupt table should be located at linear address 0 and must contain real-address mode style interrupt vectors (4 bytes containing CS and IP).
- Due to the real-address mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more than 20 bits.
- An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 KBytes).
- When an exception or interrupt occurs, only the 16 least-significant bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 KBytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One solution to this problem is for a handler to adjust the return address on the stack.)
- The SMBASE relocation feature affects the way the processor will return from an interrupt or exception generated while the SMI handler is executing. For example, if the SMBASE is relocated to above 1 MByte, but the exception handlers are below 1 MByte, a normal return to the SMI handler is not possible. One solution is to provide the exception handler with a mechanism for calculating a return address above 1 MByte from the 16-bit return address on the stack, then use a 32-bit far call to return to the interrupted procedure.
- If an SMI handler needs access to the debug trap facilities, it must ensure that an SMM accessible debug handler is available and save the current contents of debug registers DR0 through DR3 (for later restoration). Debug registers DR0 through DR3 and DR7 must then be initialized with the appropriate values.
- If an SMI handler needs access to the single-step mechanism, it must ensure that an SMM accessible single-step handler is available, and then set the TF flag in the EFLAGS register.
- If the SMI design requires the processor to respond to maskable hardware interrupts or software-generated interrupts while in SMM, it must ensure that SMM accessible interrupt handlers are available and then set the IF flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, so they do not need to be enabled.

32.7 MANAGING SYNCHRONOUS AND ASYNCHRONOUS SYSTEM MANAGEMENT INTERRUPTS

When coding for a multiprocessor system or a system with Intel HT Technology, it was not always possible for an SMI handler to distinguish between a synchronous SMI (triggered during an I/O instruction) and an asynchronous SMI. To facilitate the discrimination of these two events, incremental state information has been added to the SMM state save map.

Processors that have an SMM revision ID of 30004H or higher have the incremental state information described below.

32.7.1 I/O State Implementation

Within the extended SMM state save map, a bit (IO_SMI) is provided that is set only when an SMI is either taken immediately after a *successful* I/O instruction or is taken after a *successful* iteration of a REP I/O instruction (the *successful* notion pertains to the processor point of view; not necessarily to the corresponding platform function). When set, the IO_SMI bit provides a strong indication that the corresponding SMI was synchronous. In this case, the SMM State Save Map also supplies the port address of the I/O operation. The IO_SMI bit and the I/O Port Address may be used in conjunction with the information logged by the platform to confirm that the SMI was indeed synchronous.

The IO_SMI bit by itself is a strong indication, not a guarantee, that the SMI is synchronous. This is because an asynchronous SMI might coincidentally be taken after an I/O instruction. In such a case, the IO_SMI bit would still be set in the SMM state save map.

Information characterizing the I/O instruction is saved in two locations in the SMM State Save Map (Table 32-5). The IO_SMI bit also serves as a valid bit for the rest of the I/O information fields. The contents of these I/O information fields are not defined when the IO_SMI bit is not set.

Table 32-5. I/O Instruction Information in the SMM State Save Map

State (SMM Rev. ID: 30004H or higher)	Format								
	31	16	15	8	7	4	3	1	0
I/O State Field SMRAM offset 7FA4		I/O Port	Reserved		I/O Type		I/O Length		IO_SMI
	31								0
I/O Memory Address Field SMRAM offset 7FA0	I/O Memory Address								

When IO_SMI is set, the other fields may be interpreted as follows:

- I/O length:
 - 001 – Byte
 - 010 – Word
 - 100 – Dword
- I/O instruction type (Table 32-6)

Table 32-6. I/O Instruction Type Encodings

Instruction	Encoding
IN Immediate	1001
IN DX	0001
OUT Immediate	1000

Table 32-6. I/O Instruction Type Encodings (Contd.)

Instruction	Encoding
OUT DX	0000
INS	0011
OUTS	0010
REP INS	0111
REP OUTS	0110

32.8 NMI HANDLING WHILE IN SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence. This assumes that NMIs were not blocked before the SMI occurred. If NMIs were blocked before the SMI occurred, they are blocked after execution of RSM.

Although NMI requests are blocked when the processor enters SMM, they may be enabled through software by executing an IRET instruction. If the SMI handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same "real mode" manner in which they are handled outside of SMM.

Also, for the Pentium processor, exceptions that invoke a trap or fault handler will enable NMI interrupts from inside of SMM. This behavior is implementation specific for the Pentium processor and is not part of the IA-32 architecture.

32.9 SMM REVISION IDENTIFIER

The SMM revision identifier field is used to indicate the version of SMM and the SMM extensions that are supported by the processor (see Figure 32-2). The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture.

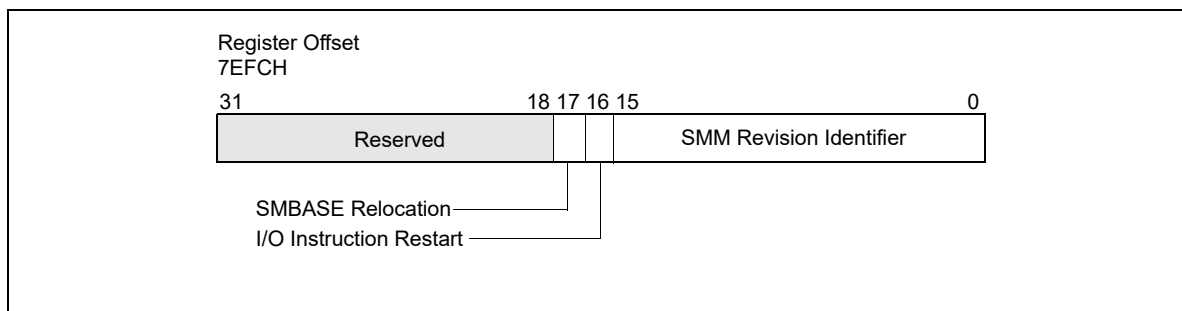


Figure 32-2. SMM Revision Identifier

The upper word of the SMM revision identifier refers to the extensions available. If the I/O instruction restart flag (bit 16) is set, the processor supports the I/O instruction restart (see Section 32.12); if the SMBASE relocation flag (bit 17) is set, SMRAM base address relocation is supported (see Section 32.11).

32.10 AUTO HALT RESTART

If the processor is in a HALT state (due to the prior execution of a HLT instruction) when it receives an SMI, the processor records the fact in the auto HALT restart flag in the saved processor state (see Figure 32-3). (This flag is located at offset 7F02H and bit 0 in the state save area of the SMRAM.)

If the processor sets the auto HALT restart flag upon entering SMM (indicating that the SMI occurred when the processor was in the HALT state), the SMI handler has two options:

- It can leave the auto HALT restart flag set, which instructs the RSM instruction to return program control to the HLT instruction. This option in effect causes the processor to re-enter the HALT state after handling the SMI. (This is the default operation.)
- It can clear the auto HALT restart flag, which instructs the RSM instruction to return program control to the instruction following the HLT instruction.

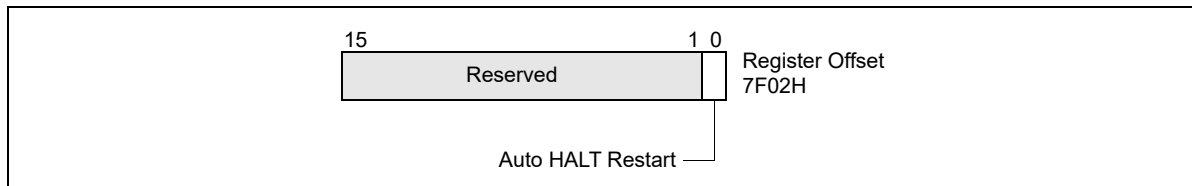


Figure 32-3. Auto HALT Restart Field

These options are summarized in Table 32-7. If the processor was not in a HALT state when the SMI was received (the auto HALT restart flag is cleared), setting the flag to 1 will cause unpredictable behavior when the RSM instruction is executed.

Table 32-7. Auto HALT Restart Flag Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
0	0	Returns to next instruction in interrupted program or task.
0	1	Unpredictable.
1	0	Returns to next instruction after HLT instruction.
1	1	Returns to HALT state.

If the HLT instruction is restarted, the processor will generate a memory access to fetch the HLT instruction (if it is not in the internal cache), and execute a HLT bus transaction. This behavior results in multiple HLT bus transactions for the same HLT instruction.

32.10.1 Executing the HLT Instruction in SMM

The HLT instruction should not be executed during SMM, unless interrupts have been enabled by setting the IF flag in the EFLAGS register. If the processor is halted in SMM, the only event that can remove the processor from this state is a maskable hardware interrupt or a hardware reset.

32.11 SMBASE RELOCATION

The default base address for the SMRAM is 30000H. This value is contained in an internal processor register called the SMBASE register. The operating system or executive can relocate the SMRAM by setting the SMBASE field in the saved state map (at offset 7EF8H) to a new value (see Figure 32-4). The RSM instruction reloads the internal SMBASE register with the value in the SMBASE field each time it exits SMM. All subsequent SMI requests will use the new SMBASE value to find the starting address for the SMI handler (at SMBASE + 8000H) and the SMRAM state

save area (from SMBASE + FE00H to SMBASE + FFFFH). (The processor resets the value in its internal SMBASE register to 30000H on a RESET, but does not change it on an INIT.)

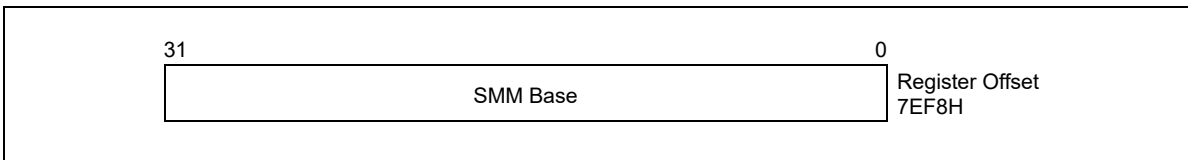


Figure 32-4. SMBASE Relocation Field

In multiple-processor systems, initialization software must adjust the SMBASE value for each processor so that the SMRAM state save areas for each processor do not overlap. (For Pentium and Intel486 processors, the SMBASE values must be aligned on a 32-KByte boundary or the processor will enter shutdown state during the execution of a RSM instruction.)

If the SMBASE relocation flag in the SMM revision identifier field is set, it indicates the ability to relocate the SMBASE (see Section 32.9).

32.12 I/O INSTRUCTION RESTART

If the I/O instruction restart flag in the SMM revision identifier field is set (see Section 32.9), the I/O instruction restart mechanism is present on the processor. This mechanism allows an interrupted I/O instruction to be re-executed upon returning from SMM mode. For example, if an I/O instruction is used to access a powered-down I/O device, a chipset supporting this device can intercept the access and respond by asserting SMI#. This action invokes the SMI handler to power-up the device. Upon returning from the SMI handler, the I/O instruction restart mechanism can be used to re-execute the I/O instruction that caused the SMI.

The I/O instruction restart field (at offset 7F00H in the SMM state-save area, see Figure 32-5) controls I/O instruction restart. When an RSM instruction is executed, if this field contains the value FFH, then the EIP register is modified to point to the I/O instruction that received the SMI request. The processor will then automatically re-execute the I/O instruction that the SMI trapped. (The processor saves the necessary machine state to ensure that re-execution of the instruction is handled coherently.)

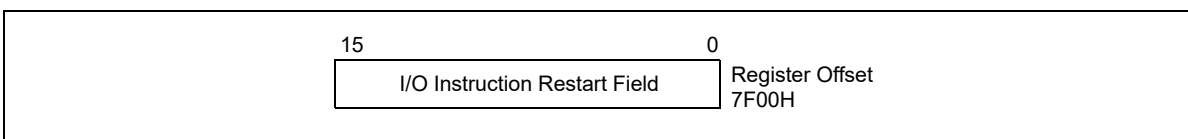


Figure 32-5. I/O Instruction Restart Field

If the I/O instruction restart field contains the value 00H when the RSM instruction is executed, then the processor begins program execution with the instruction following the I/O instruction. (When a repeat prefix is being used, the next instruction may be the next I/O instruction in the repeat loop.) Not re-executing the interrupted I/O instruction is the default behavior; the processor automatically initializes the I/O instruction restart field to 00H upon entering SMM. Table 32-8 summarizes the states of the I/O instruction restart field.

Table 32-8. I/O Instruction Restart Field Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
00H	00H	Does not re-execute trapped I/O instruction.
00H	FFH	Re-executes trapped I/O instruction.

The I/O instruction restart mechanism does not indicate the cause of the SMI. It is the responsibility of the SMI handler to examine the state of the processor to determine the cause of the SMI and to determine if an I/O instruction was interrupted and should be restarted upon exiting SMM. If an SMI interrupt is signaled on a non-I/O instruction boundary, setting the I/O instruction restart field to FFH prior to executing the RSM instruction will likely result in a program error.

32.12.1 Back-to-Back SMI Interrupts When I/O Instruction Restart Is Being Used

If an SMI interrupt is signaled while the processor is servicing an SMI interrupt that occurred on an I/O instruction boundary, the processor will service the new SMI request before restarting the originally interrupted I/O instruction. If the I/O instruction restart field is set to FFH prior to returning from the second SMI handler, the EIP will point to an address different from the originally interrupted I/O instruction, which will likely lead to a program error. To avoid this situation, the SMI handler must be able to recognize the occurrence of back-to-back SMI interrupts when I/O instruction restart is being used and ensure that the handler sets the I/O instruction restart field to 00H prior to returning from the second invocation of the SMI handler.

32.13 SMM MULTIPLE-PROCESSOR CONSIDERATIONS

The following should be noted when designing multiple-processor systems:

- Any processor in a multiprocessor system can respond to an SMI.
- Each processor needs its own SMRAM space. This space can be in system memory or in a separate RAM.
- The SMRAMs for different processors can be overlapped in the same memory space. The only stipulation is that each processor needs its own state save area and its own dynamic data storage area. (Also, for the Pentium and Intel486 processors, the SMBASE address must be located on a 32-KByte boundary.) Code and static data can be shared among processors. Overlapping SMRAM spaces can be done more efficiently with the P6 family processors because they do not require that the SMBASE address be on a 32-KByte boundary.
- The SMI handler will need to initialize the SMBASE for each processor.
- Processors can respond to local SMIs through their SMI# pins or to SMIs received through the APIC interface. The APIC interface can distribute SMIs to different processors.
- Two or more processors can be executing in SMM at the same time.
- When operating Pentium processors in dual processing (DP) mode, the SMIACK# pin is driven only by the MRM processor and should be sampled with ADS#. For additional details, see Chapter 14 of the *Pentium Processor Family User's Manual, Volume 1*.

SMM is not re-entrant, because the SMRAM State Save Map is fixed relative to the SMBASE. If there is a need to support two or more processors in SMM mode at the same time then each processor should have dedicated SMRAM spaces. This can be done by using the SMBASE Relocation feature (see Section 32.11).

32.14 DEFAULT TREATMENT OF SMIS AND SMM WITH VMX OPERATION AND SMX OPERATION

Under the default treatment, the interactions of SMIs and SMM with VMX operation are few. This section details those interactions. It also explains how this treatment affects SMX operation.

32.14.1 Default Treatment of SMI Delivery

Ordinary SMI delivery saves processor state into SMRAM and then loads state based on architectural definitions. Under the default treatment, processors that support VMX operation perform SMI delivery as follows:

```

enter SMM;
save the following internal to the processor:
    CR4.VMXE
        an indication of whether the logical processor was in VMX operation (root or non-root)
IF the logical processor is in VMX operation
    THEN
        save current VMCS pointer internal to the processor;
        leave VMX operation;
        save VMX-critical state defined below;
FI;
IF the logical processor supports SMX operation
    THEN
        save internal to the logical processor an indication of whether the Intel® TXT private space is locked;
        IF the TXT private space is unlocked
            THEN lock the TXT private space;
        FI;
FI;
CR4.VMXE := 0;
perform ordinary SMI delivery:
    save processor state in SMRAM;
    set processor state to standard SMM values;1
    invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H
    are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 29.4);

```

The pseudocode above makes reference to the saving of **VMX-critical state**. This state consists of the following: (1) SS.DPL (the current privilege level); (2) RFLAGS.VM²; (3) the state of blocking by STI and by MOV SS (see Table 25-3 in Section 25.4.2); (4) the state of virtual-NMI blocking (only if the processor is in VMX non-root operation and the “virtual NMIs” VM-execution control is 1); and (5) an indication of whether an MTF VM exit is pending (see Section 26.5.2). These data may be saved internal to the processor or in the VMCS region of the current VMCS. Processors that do not support SMI recognition while there is blocking by STI or by MOV SS need not save the state of such blocking.

If the logical processor supports the 1-setting of the “enable EPT” VM-execution control and the logical processor was in VMX non-root operation at the time of an SMI, it saves the value of that control into bit 0 of the 32-bit field at offset SMBASE + 8000H + 7EE0H (SMBASE + FEE0H; see Table 32-3).³ If the logical processor was not in VMX non-root operation at the time of the SMI, it saves 0 into that bit. If the logical processor saves 1 into that bit (it was in VMX non-root operation and the “enable EPT” VM-execution control was 1), it saves the value of the EPT pointer (EPTP) into the 64-bit field at offset SMBASE + 8000H + 7ED8H (SMBASE + FED8H).

Because SMI delivery causes a logical processor to leave VMX operation, all the controls associated with VMX non-root operation are disabled in SMM and thus cannot cause VM exits while the logical processor in SMM.

-
1. This causes the logical processor to block INIT signals, NMIs, and SMIs.
 2. Section 32.14 and Section 32.15 use the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of these registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to the lower 32 bits of the register.
 3. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, SMI functions as the “enable EPT” VM-execution control were 0. See Section 25.6.2.

32.14.2 Default Treatment of RSM

Ordinary execution of RSM restores processor state from SMRAM. Under the default treatment, processors that support VMX operation perform RSM as follows:

```

IF VMXE = 1 in CR4 image in SMRAM
    THEN fail and enter shutdown state;
    ELSE
        restore state normally from SMRAM;
        invalidate linear mappings and combined mappings associated with all VPIDs and all PCIDs; combined mappings are invalidated
        for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 29.4);
        IF the logical processor supports SMX operation and the Intel® TXT private space was unlocked at the time of the last SMI (as
        saved)
            THEN unlock the TXT private space;
        FI;
        CR4.VMXE := value stored internally;
        IF internal storage indicates that the logical processor
        had been in VMX operation (root or non-root)
            THEN
                enter VMX operation (root or non-root);
                restore VMX-critical state as defined in Section 32.14.1;
                set to their fixed values any bits in CR0 and CR4 whose values must be fixed in VMX operation (see Section 24.8);1
                IF RFLAGS.VM = 0 AND (in VMX root operation OR the “unrestricted guest” VM-execution control is 0)2
                    THEN
                        CS.RPL := SS.DPL;
                        SS.RPL := SS.DPL;
                    FI;
                restore current VMCS pointer;
            FI;
        leave SMM;
        IF logical processor will be in VMX operation or in SMX operation after RSM
            THEN block A20M and leave A20M mode;
        FI;
    FI;

```

RSM unblocks SMIs. It restores the state of blocking by NMI (see Table 25-3 in Section 25.4.2) as follows:

- If the RSM is not to VMX non-root operation or if the “virtual NMIs” VM-execution control will be 0, the state of NMI blocking is restored normally.
- If the RSM is to VMX non-root operation and the “virtual NMIs” VM-execution control will be 1, NMIs are not blocked after RSM. The state of virtual-NMI blocking is restored as part of VMX-critical state.

INIT signals are blocked after RSM if and only if the logical processor will be in VMX root operation.

If RSM returns a logical processor to VMX non-root operation, it re-establishes the controls associated with the current VMCS. If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs immediately after RSM if the enabling conditions apply. The same is true for the “NMI-window exiting” VM-execution control. Such VM exits occur with their normal priority. See Section 26.2.

If an MTF VM exit was pending at the time of the previous SMI, an MTF VM exit is pending on the instruction boundary following execution of RSM. The following items detail the treatment of MTF VM exits that may be pending following RSM:

1. If the RSM is to VMX non-root operation and both the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls will be 1, CR0.PE and CR0.PG retain the values that were loaded from SMRAM regardless of what is reported in the capability MSR IA32_VMX_CRO_FIXED0.
2. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 25.6.2.

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these MTF VM exits. These MTF VM exits take priority over debug-trap exceptions and lower priority events.
- These MTF VM exits wake the logical processor if RSM caused the logical processor to enter the HLT state (see Section 32.10). They do not occur if the logical processor just entered the shutdown state.

32.14.3 Protection of CR4.VMXE in SMM

Under the default treatment, CR4.VMXE is treated as a reserved bit while a logical processor is in SMM. Any attempt by software running in SMM to set this bit causes a general-protection exception. In addition, software cannot use VMX instructions or enter VMX operation while in SMM.

32.14.4 VMXOFF and SMI Unblocking

The VMXOFF instruction can be executed only with the default treatment (see Section 32.15.1) and only outside SMM. If SMIs are blocked when VMXOFF is executed, VMXOFF unblocks them unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 32.15.5 for details regarding this MSR).¹ Section 32.15.7 identifies a case in which SMIs may be blocked when VMXOFF is executed.

Not all processors allow this bit to be set to 1. Software should consult the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine whether this is allowed.

32.15 DUAL-MONITOR TREATMENT OF SMIs AND SMM

Dual-monitor treatment is activated through the cooperation of the **executive monitor** (the VMM that operates outside of SMM to provide basic virtualization) and the **SMM-transfer monitor (STM)**; the VMM that operates inside SMM—while in VMX operation—to support system-management functions). Control is transferred to the STM through VM exits; VM entries are used to return from SMM.

The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

32.15.1 Dual-Monitor Treatment Overview

The dual-monitor treatment uses an executive monitor and an SMM-transfer monitor (STM). Transitions from the executive monitor or its guests to the STM are called **SMM VM exits** and are discussed in Section 32.15.2. SMM VM exits are caused by SMIs as well as executions of VMCALL in VMX root operation. The latter allow the executive monitor to call the STM for service.

The STM runs in VMX root operation and uses VMX instructions to establish a VMCS and perform VM entries to its own guests. This is done all inside SMM (see Section 32.15.3). The STM returns from SMM, not by using the RSM instruction, but by using a VM entry that returns from SMM. Such VM entries are described in Section 32.15.4.

Initially, there is no STM and the default treatment (Section 32.14) is used. The dual-monitor treatment is not used until it is enabled and activated. The steps to do this are described in Section 32.15.5 and Section 32.15.6.

It is not possible to leave VMX operation under the dual-monitor treatment; VMXOFF will fail if executed. The dual-monitor treatment must be deactivated first. The STM deactivates dual-monitor treatment using a VM entry that returns from SMM with the “deactivate dual-monitor treatment” VM-entry control set to 1 (see Section 32.15.7).

The executive monitor configures any VMCS that it uses for VM exits to the executive monitor. SMM VM exits, which transfer control to the STM, use a different VMCS. Under the dual-monitor treatment, each logical processor uses a separate VMCS called the **SMM-transfer VMCS**. When the dual-monitor treatment is active, the logical processor maintains another VMCS pointer called the **SMM-transfer VMCS pointer**. The SMM-transfer VMCS pointer is established when the dual-monitor treatment is activated.

1. Setting IA32_SMM_MONITOR_CTL[bit 2] to 1 prevents VMXOFF from unblocking SMIs regardless of the value of the register’s valid bit (bit 0).

32.15.2 SMM VM Exits

An SMM VM exit is a VM exit that begins outside SMM and that ends in SMM.

Unlike other VM exits, SMM VM exits can begin in VMX root operation. SMM VM exits result from the arrival of an SMI outside SMM or from execution of VMCALL in VMX root operation outside SMM. Execution of VMCALL in VMX root operation causes an SMM VM exit only if the valid bit is set in the IA32_SMM_MONITOR_CTL MSR (see Section 32.15.5).

Execution of VMCALL in VMX root operation causes an SMM VM exit even under the default treatment. This SMM VM exit activates the dual-monitor treatment (see Section 32.15.6).

Differences between SMM VM exits and other VM exits are detailed in Sections 32.15.2.1 through 32.15.2.5. Differences between SMM VM exits that activate the dual-monitor treatment and other SMM VM exits are described in Section 32.15.6.

32.15.2.1 Architectural State Before a VM Exit

System-management interrupts (SMIs) that cause SMM VM exits always do so directly. They do not save state to SMRAM as they do under the default treatment.

32.15.2.2 Updating the Current-VMCS and Executive-VMCS Pointers

SMM VM exits begin by performing the following steps:

1. The executive-VMCS pointer field in the SMM-transfer VMCS is loaded as follows:
 - If the SMM VM exit commenced in VMX non-root operation, it receives the current-VMCS pointer.
 - If the SMM VM exit commenced in VMX root operation, it receives the VMXON pointer.
2. The current-VMCS pointer is loaded with the value of the SMM-transfer VMCS pointer.

The last step ensures that the current VMCS is the SMM-transfer VMCS. VM-exit information is recorded in that VMCS, and VM-entry control fields in that VMCS are updated. State is saved into the guest-state area of that VMCS. The VM-exit controls and host-state area of that VMCS determine how the VM exit operates.

32.15.2.3 Recording VM-Exit Information

SMM VM exits differ from other VM exit with regard to the way they record VM-exit information. The differences follow.

- **Exit reason.**
 - Bits 15:0 of this field contain the basic exit reason. The field is loaded with the reason for the SMM VM exit: I/O SMI (an SMI arrived immediately after retirement of an I/O instruction), other SMI, or VMCALL. See Appendix C, “VMX Basic Exit Reasons.”
 - SMM VM exits are the only VM exits that may occur in VMX root operation. Because the SMM-transfer monitor may need to know whether it was invoked from VMX root or VMX non-root operation, this information is stored in bit 29 of the exit-reason field (see Table 25-17 in Section 25.9.1). The bit is set by SMM VM exits from VMX root operation.
 - If the SMM VM exit occurred in VMX non-root operation and an MTF VM exit was pending, bit 28 of the exit-reason field is set; otherwise, it is cleared.
 - Bits 27:16 and bits 31:30 are cleared.
- **Exit qualification.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, the exit qualification contains information about the I/O instruction that retired immediately before the SMI. It has the format given in Table 32-9.
- **Guest linear address.** This field is used for VM exits due to SMIs that arrive immediately after the retirement of an INS or OUTS instruction for which the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) is usable. The field receives the value of the linear address generated by ES:(E)DI (for INS) or segment:(E)SI (for OUTS; the default segment is DS but can be overridden by a segment override

Table 32-9. Exit Qualification for SMIs That Arrive Immediately After the Retirement of an I/O Instruction

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used.
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

prefix) at the time the instruction started. If the relevant segment is not usable, the value is undefined. On processors that support Intel 64 architecture, bits 63:32 are clear if the logical processor was not in 64-bit mode before the VM exit.

- **I/O RCX, I/O RSI, I/O RDI, and I/O RIP.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, these fields receive the values that were in RCX, RSI, RDI, and RIP, respectively, before the I/O instruction executed. Thus, the value saved for I/O RIP addresses the I/O instruction.

32.15.2.4 Saving Guest State

SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area.

The value of the VMX-preemption timer is saved into the corresponding field in the guest-state area if the “save VMX-preemption timer value” VM-exit control is 1. That field becomes undefined if, in addition, either the SMM VM exit is from VMX root operation or the SMM VM exit is from VMX non-root operation and the “activate VMX-preemption timer” VM-execution control is 0.

32.15.2.5 Updating State

If an SMM VM exit is from VMX non-root operation and the “Intel PT uses guest physical addresses” VM-execution control is 1, the IA32_RTIT_CTL MSR is cleared to 00000000_00000000H.¹ This is done even if the “clear IA32_RTIT_CTL” VM-exit control is 0.

SMM VM exits affect the non-register state of a logical processor as follows:

- SMM VM exits cause non-maskable interrupts (NMIs) to be blocked; they may be unblocked through execution of IRET or through a VM entry (depending on the value loaded for the interruptibility state and the setting of the “virtual NMIs” VM-execution control).
- SMM VM exits cause SMIs to be blocked; they may be unblocked by a VM entry that returns from SMM (see Section 32.15.4).

1. In this situation, the value of this MSR was saved earlier into the guest-state area. All VM exits save this MSR if the 1-setting of the “load IA32_RTIT_CTL” VM-entry control is supported (see Section 28.3.1), which must be the case if the “Intel PT uses guest physical addresses” VM-execution control is 1 (see Section 27.2.1.1).

SMM VM exits invalidate linear mappings and combined mappings associated with VPID 0000H for all PCIDs. Combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 29.4). (Ordinary VM exits are not required to perform such invalidation if the “enable VPID” VM-execution control is 1; see Section 28.5.5.)

32.15.3 Operation of the SMM-Transfer Monitor

Once invoked, the SMM-transfer monitor (STM) is in VMX root operation and can use VMX instructions to configure VMCSs and to cause VM entries to virtual machines supported by those structures. As noted in Section 32.15.1, the VMXOFF instruction cannot be used under the dual-monitor treatment and thus cannot be used by the STM.

The RSM instruction also cannot be used under the dual-monitor treatment. As noted in Section 26.1.3, it causes a VM exit if executed in SMM in VMX non-root operation. If executed in VMX root operation, it causes an invalid-opcode exception. The STM uses VM entries to return from SMM (see Section 32.15.4).

32.15.4 VM Entries that Return from SMM

The SMM-transfer monitor (STM) returns from SMM using a VM entry with the “entry to SMM” VM-entry control clear. VM entries that return from SMM reverse the effects of an SMM VM exit (see Section 32.15.2).

VM entries that return from SMM may differ from other VM entries in that they do not necessarily enter VMX non-root operation. If the executive-VMCS pointer field in the current VMCS contains the VMXON pointer, the logical processor remains in VMX root operation after VM entry.

For differences between VM entries that return from SMM and other VM entries see Sections 32.15.4.1 through 32.15.4.10.

32.15.4.1 Checks on the Executive-VMCS Pointer Field

VM entries that return from SMM perform the following checks on the executive-VMCS pointer field in the current VMCS:

- Bits 11:0 must be 0.
- The pointer must not set any bits beyond the processor’s physical-address width.^{1,2}
- The 32 bits located in memory referenced by the physical address in the pointer must contain the processor’s VMCS revision identifier (see Section 25.2).

The checks above are performed before the checks described in Section 32.15.4.2 and before any of the following checks:

- If the “deactivate dual-monitor treatment” VM-entry control is 0 and the executive-VMCS pointer field does not contain the VMXON pointer, the launch state of the executive VMCS (the VMCS referenced by the executive-VMCS pointer field) must be launched (see Section 25.11.3).
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the executive-VMCS pointer field must contain the VMXON pointer (see Section 32.15.7).³

32.15.4.2 Checks on VM-Execution Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-execution control fields specified in Section 27.2.1.1. They do not apply the checks to the current VMCS. Instead, VM-entry behavior depends on whether the executive-VMCS pointer field contains the VMXON pointer:

-
1. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
 2. If IA32_VMX_BASIC[48] is read as 1, this pointer must not set any bits in the range 63:32; see Appendix A.1.
 3. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are not performed at all.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the checks are performed on the VM-execution control fields in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS). These checks are performed after checking the executive-VMCS pointer field itself (for proper alignment).

Other VM entries ensure that, if “activate VMX-preemption timer” VM-execution control is 0, the “save VMX-preemption timer value” VM-execution control is also 0. This check is not performed by VM entries that return from SMM.

32.15.4.3 Checks on VM-Entry Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-entry control fields specified in Section 27.2.1.3.

Specifically, if the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the VM-entry interruption-information field must not indicate injection of a pending MTF VM exit (see Section 27.6.2). Specifically, the following cannot all be true for that field:

- the valid bit (bit 31) is 1
- the interruption type (bits 10:8) is 7 (other event); and
- the vector (bits 7:0) is 0 (pending MTF VM exit).

32.15.4.4 Checks on the Guest State Area

Section 27.3.1 specifies checks performed on fields in the guest-state area of the VMCS. Some of these checks are conditioned on the settings of certain VM-execution controls (e.g., “virtual NMIs” or “unrestricted guest”).

VM entries that return from SMM modify these checks based on whether the executive-VMCS pointer field contains the VMXON pointer:¹

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are performed as all relevant VM-execution controls were 0. (As a result, some checks may not be performed at all.)
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), this check is performed based on the settings of the VM-execution controls in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS).

For VM entries that return from SMM, the activity-state field must not indicate the wait-for-SIPI state if the executive-VMCS pointer field contains the VMXON pointer (the VM entry is to VMX root operation).

32.15.4.5 Loading Guest State

VM entries that return from SMM load the SMBASE register from the SMBASE field.

VM entries that return from SMM invalidate linear mappings and combined mappings associated with all VPIDs. Combined mappings are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 29.4). (Ordinary VM entries are required to perform such invalidation only for VPID 0000H and are not required to do even that if the “enable VPID” VM-execution control is 1; see Section 27.3.2.5.)

32.15.4.6 VMX-Preemption Timer

A VM entry that returns from SMM activates the VMX-preemption timer only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation) and the “activate VMX-preemption timer” VM-execution control is 1 in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field). In this case, VM entry starts the VMX-preemption timer with the value in the VMX-preemption timer-value field in the current VMCS.

1. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

32.15.4.7 Updating the Current-VMCS and SMM-Transfer VMCS Pointers

Successful VM entries (returning from SMM) load the SMM-transfer VMCS pointer with the current-VMCS pointer. Following this, they load the current-VMCS pointer from a field in the current VMCS:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the current-VMCS pointer is loaded from the VMCS-link pointer field.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the current-VMCS pointer is loaded with the value of the executive-VMCS pointer field.

If the VM entry successfully enters VMX non-root operation, the VM-execution controls in effect after the VM entry are those from the new current VMCS. This includes any structures external to the VMCS referenced by VM-execution control fields.

The updating of these VMCS pointers occurs before event injection. Event injection is determined, however, by the VM-entry control fields in the VMCS that was current when the VM entry commenced.

32.15.4.8 VM Exits Induced by VM Entry

Section 27.6.1.2 describes how the event-delivery process invoked by event injection may lead to a VM exit. Section 27.7.3 to Section 27.7.7 describe other situations that may cause a VM exit to occur immediately after a VM entry.

Whether these VM exits occur is determined by the VM-execution control fields in the current VMCS. For VM entries that return from SMM, they can occur only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation).

In this case, determination is based on the VM-execution control fields in the VMCS that is current after the VM entry. This is the VMCS referenced by the value of the executive-VMCS pointer field at the time of the VM entry (see Section 32.15.4.7). This VMCS also controls the delivery of such VM exits. Thus, VM exits induced by a VM entry returning from SMM are to the executive monitor and not to the STM.

32.15.4.9 SMI Blocking

VM entries that return from SMM determine the blocking of system-management interrupts (SMIs) as follows:

- If the “deactivate dual-monitor treatment” VM-entry control is 0, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the blocking of SMIs depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, SMIs are blocked after VM entry.
 - If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

VM entries that return from SMM and that do not deactivate the dual-monitor treatment may leave SMIs blocked. This feature exists to allow the STM to invoke functionality outside of SMM without unblocking SMIs.

32.15.4.10 Failures of VM Entries That Return from SMM

Section 27.8 describes the treatment of VM entries that fail during or after loading guest state. Such failures record information in the VM-exit information fields and load processor state as would be done on a VM exit. The VMCS used is the one that was current before the VM entry commenced. Control is thus transferred to the STM and the logical processor remains in SMM.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

32.15.5 Enabling the Dual-Monitor Treatment

Code and data for the SMM-transfer monitor (STM) reside in a region of SMRAM called the **monitor segment** (MSEG). Code running in SMM determines the location of MSEG and establishes its content. This code is also responsible for enabling the dual-monitor treatment.

SMM code enables the dual-monitor treatment and specifies the location of MSEG by writing to the IA32_SMM_MONITOR_CTL MSR (index 9BH). The MSR has the following format:

- Bit 0 is the register's valid bit. The STM may be invoked using VMCALL only if this bit is 1. Because VMCALL is used to activate the dual-monitor treatment (see Section 32.15.6), the dual-monitor treatment cannot be activated if the bit is 0. This bit is cleared when the logical processor is reset.
- Bit 1 is reserved.
- Bit 2 determines whether executions of VMXOFF unblock SMIs under the default treatment of SMIs and SMM. Executions of VMXOFF unblock SMIs unless bit 2 is 1 (the value of bit 0 is irrelevant). See Section 32.14.4. Certain leaf functions of the GETSEC instruction clear this bit (see Chapter 7, "Safer Mode Extensions Reference," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D*).
- Bits 11:3 are reserved.
- Bits 31:12 contain a value that, when shifted left 12 bits, is the physical address of MSEG (the MSEG base address).
- Bits 63:32 are reserved.

The following items detail use of this MSR:

- The IA32_SMM_MONITOR_CTL MSR is supported only on processors that support the dual-monitor treatment.¹ On other processors, accesses to the MSR using RDMSR or WRMSR generate a general-protection fault (#GP(0)).
- A write to the IA32_SMM_MONITOR_CTL MSR using WRMSR generates a general-protection fault (#GP(0)) if executed outside of SMM or if an attempt is made to set any reserved bit. An attempt to write to the IA32_SMM_MONITOR_CTL MSR fails if made as part of a VM exit that does not end in SMM or part of a VM entry that does not begin in SMM.
- Reads from the IA32_SMM_MONITOR_CTL MSR using RDMSR are allowed any time RDMSR is allowed. The MSR may be read as part of any VM exit.
- The dual-monitor treatment can be activated only if the valid bit in the MSR is set to 1.

The 32 bytes located at the MSEG base address are called the **MSEG header**. The format of the MSEG header is given in Table 32-10 (each field is 32 bits).

Table 32-10. Format of MSEG Header

Byte Offset	Field
0	MSEG-header revision identifier
4	SMM-transfer monitor features
8	GDTR limit
12	GDTR base offset
16	CS selector
20	EIP offset
24	ESP offset
28	CR3 offset

1. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

To ensure proper behavior in VMX operation, software should maintain the MSEG header in writeback cacheable memory. Future implementations may allow or require a different memory type.¹ Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

SMM code should enable the dual-monitor treatment (by setting the valid bit in IA32_SMM_MONITOR_CTL MSR) only after establishing the content of the MSEG header as follows:

- Bytes 3:0 contain the **MSEG revision identifier**. Different processors may use different MSEG revision identifiers. These identifiers enable software to avoid using an MSEG header formatted for one processor on a processor that uses a different format. Software can discover the MSEG revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).
- Bytes 7:4 contain the **SMM-transfer monitor features** field. Bits 31:1 of this field are reserved and must be zero. Bit 0 of the field is the **IA-32e mode SMM feature bit**. It indicates whether the logical processor will be in IA-32e mode after the STM is activated (see Section 32.15.6).
- Bytes 31:8 contain fields that determine how processor state is loaded when the STM is activated (see Section 32.15.6.5). SMM code should establish these fields so that activating of the STM invokes the STM's initialization code.

32.15.6 Activating the Dual-Monitor Treatment

The dual-monitor treatment may be enabled by SMM code as described in Section 32.15.5. The dual-monitor treatment is activated only if it is enabled and only by the executive monitor. The executive monitor activates the dual-monitor treatment by executing VMCALL in VMX root operation.

When VMCALL activates the dual-monitor treatment, it causes an SMM VM exit. Differences between this SMM VM exit and other SMM VM exits are discussed in Sections 32.15.6.1 through 32.15.6.6. See also "VMCALL—Call to VM Monitor" in Chapter 31.

32.15.6.1 Initial Checks

An execution of VMCALL attempts to activate the dual-monitor treatment if (1) the processor supports the dual-monitor treatment;² (2) the logical processor is in VMX root operation; (3) the logical processor is outside SMM and the valid bit is set in the IA32_SMM_MONITOR_CTL MSR; (4) the logical processor is not in virtual-8086 mode and not in compatibility mode; (5) CPL = 0; and (6) the dual-monitor treatment is not active.

Such an execution of VMCALL begins with some initial checks. These checks are performed before updating the current-VMCS pointer and the executive-VMCS pointer field (see Section 32.15.2.2).

The VMCS that manages SMM VM exit caused by this VMCALL is the current VMCS established by the executive monitor. The VMCALL performs the following checks on the current VMCS in the order indicated:

1. There must be a current VMCS pointer.
2. The launch state of the current VMCS must be clear.
3. **The VM-exit controls in the current VMCS must be set properly:**
 - **Reserved bits in the primary VM-exit controls must be set properly.** Software may consult the VMX capability MSRs to determine the proper setting (see Appendix A.4.1).
 - **If the "activate secondary controls" primary VM-exit control is 1, reserved bits in the secondary VM-exit controls must be cleared.** Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.4.2).

1. Alternatively, software may map the MSEG header with the UC memory type; this may be necessary, depending on how memory is organized. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.

2. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

- If the “activate secondary controls” primary VM-exit control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary VM-exit controls. The logical processor operates as if all the secondary VM-exit controls were 0.

If any of these checks fail, subsequent checks are skipped and VMCALL fails. If all these checks succeed, the logical processor uses the IA32_SMM_MONITOR_CTL MSR to determine the base address of MSEG. The following checks are performed in the order indicated:

1. The logical processor reads the 32 bits at the base of MSEG and compares them to the processor’s MSEG revision identifier.
2. The logical processor reads the SMM-transfer monitor features field:
 - Bit 0 of the field is the IA-32e mode SMM feature bit, and it indicates whether the logical processor will be in IA-32e mode after the SMM-transfer monitor (STM) is activated.
 - If the VMCALL is executed on a processor that does not support Intel 64 architecture, the IA-32e mode SMM feature bit must be 0.
 - If the VMCALL is executed in 64-bit mode, the IA-32e mode SMM feature bit must be 1.
 - Bits 31:1 of this field are currently reserved and must be zero.

If any of these checks fail, subsequent checks are skipped and the VMCALL fails.

32.15.6.2 Updating the Current-VMCS and Executive-VMCS Pointers

Before performing the steps in Section 32.15.2.2, SMM VM exits that activate the dual-monitor treatment begin by loading the SMM-transfer VMCS pointer with the value of the current-VMCS pointer.

32.15.6.3 Saving Guest State

As noted in Section 32.15.2.4, SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area. While this is true also for SMM VM exits that activate the dual-monitor treatment, the VMCS used for those VM exits exists outside SMRAM.

The SMM-transfer monitor (STM) can also discover the current value of the SMBASE register by using the RDMSR instruction to read the IA32_SMBASE MSR (MSR address 9EH). The following items detail use of this MSR:

- The MSR is supported only if IA32_VMX_MISC[15] = 1 (see Appendix A.6).
- A write to the IA32_SMBASE MSR using WRMSR generates a general-protection fault (#GP(0)). An attempt to write to the IA32_SMBASE MSR fails if made as part of a VM exit or part of a VM entry.
- A read from the IA32_SMBASE MSR using RDMSR generates a general-protection fault (#GP(0)) if executed outside of SMM. An attempt to read from the IA32_SMBASE MSR fails if made as part of a VM exit that does not end in SMM.

32.15.6.4 Saving MSRs

The VM-exit MSR-store area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are saved into that area.

32.15.6.5 Loading Host State

The VMCS that is current during an SMM VM exit that activates the dual-monitor treatment was established by the executive monitor. It does not contain the VM-exit controls and host state required to initialize the STM. For this reason, such SMM VM exits do not load processor state as described in Section 28.5. Instead, state is set to fixed values or loaded based on the content of the MSEG header (see Table 32-10):

- CR0 is set to as follows:
 - PG, NE, ET, MP, and PE are all set to 1.
 - CD and NW are left unchanged.

- All other bits are cleared to 0.
- CR3 is set as follows:
 - Bits 63:32 are cleared on processors that support IA-32e mode.
 - Bits 31:12 are set to bits 31:12 of the sum of the MSEG base address and the CR3-offset field in the MSEG header.
 - Bits 11:5 and bits 2:0 are cleared (the corresponding bits in the CR3-offset field in the MSEG header are ignored).
 - Bits 4:3 are set to bits 4:3 of the CR3-offset field in the MSEG header.
- CR4 is set as follows:
 - MCE, PGE, CET, and PCIDE are cleared.
 - PAE is set to the value of the IA-32e mode SMM feature bit.
 - If the IA-32e mode SMM feature bit is clear, PSE is set to 1 if supported by the processor; if the bit is set, PSE is cleared.
 - All other bits are unchanged.
- DR7 is set to 400H.
- The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
- The registers CS, SS, DS, ES, FS, and GS are loaded as follows:
 - All registers are usable.
 - CS.selector is loaded from the corresponding field in the MSEG header (the high 16 bits are ignored), with bits 2:0 cleared to 0. If the result is 0000H, CS.selector is set to 0008H.
 - The selectors for SS, DS, ES, FS, and GS are set to CS.selector+0008H. If the result is 0000H (if the CS selector was FFF8H), these selectors are instead set to 0008H.
 - The base addresses of all registers are cleared to zero.
 - The segment limits for all registers are set to FFFFFFFFH.
 - The AR bytes for the registers are set as follows:
 - CS.Type is set to 11 (execute/read, accessed, non-conforming code segment).
 - For SS, DS, ES, FS, and GS, the Type is set to 3 (read/write, accessed, expand-up data segment).
 - The S bits for all registers are set to 1.
 - The DPL for each register is set to 0.
 - The P bits for all registers are set to 1.
 - On processors that support Intel 64 architecture, CS.L is loaded with the value of the IA-32e mode SMM feature bit.
 - CS.D is loaded with the inverse of the value of the IA-32e mode SMM feature bit.
 - For each of SS, DS, ES, FS, and GS, the D/B bit is set to 1.
 - The G bits for all registers are set to 1.
- LDTR is unusable. The LDTR selector is cleared to 0000H, and the register is otherwise undefined (although the base address is always canonical)
- GDTR.base is set to the sum of the MSEG base address and the GDTR base-offset field in the MSEG header (bits 63:32 are always cleared on processors that support IA-32e mode). GDTR.limit is set to the corresponding field in the MSEG header (the high 16 bits are ignored).
- IDTR.base is unchanged. IDTR.limit is cleared to 0000H.
- RIP is set to the sum of the MSEG base address and the value of the RIP-offset field in the MSEG header (bits 63:32 are always cleared on logical processors that support IA-32e mode).
- RSP is set to the sum of the MSEG base address and the value of the RSP-offset field in the MSEG header (bits 63:32 are always cleared on logical processor that supports IA-32e mode).

- RFLAGS is cleared, except bit 1, which is always set.
- The logical processor is left in the active state.
- Event blocking after the SMM VM exit is as follows:
 - There is no blocking by STI or by MOV SS.
 - There is blocking by non-maskable interrupts (NMIs) and by SMIs.
- There are no pending debug exceptions after the SMM VM exit.
- For processors that support IA-32e mode, the IA32_EFER MSR is modified so that LME and LMA both contain the value of the IA-32e mode SMM feature bit.

If any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs are updated so that, after VM exit, the logical processor does not use translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32_EFER.LMA was 1 before and after the transition).

32.15.6.6 Loading MSRs

The VM-exit MSR-load area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are loaded from that area.

32.15.7 Deactivating the Dual-Monitor Treatment

The SMM-transfer monitor may deactivate the dual-monitor treatment and return the processor to default treatment of SMIs and SMM (see Section 32.14). It does this by executing a VM entry with the “deactivate dual-monitor treatment” VM-entry control set to 1.

As noted in Section 27.2.1.3 and Section 32.15.4.1, an attempt to deactivate the dual-monitor treatment fails in the following situations: (1) the processor is not in SMM; (2) the “entry to SMM” VM-entry control is 1; or (3) the executive-VMCS pointer does not contain the VMXON pointer (the VM entry is to VMX non-root operation).

As noted in Section 32.15.4.9, VM entries that deactivate the dual-monitor treatment ignore the SMI bit in the interruptibility-state field of the guest-state area. Instead, the blocking of SMIs following such a VM entry depends on whether the logical processor is in SMX operation:¹

- If the logical processor is in SMX operation, SMIs are blocked after VM entry. SMIs may later be unblocked by the VMXOFF instruction (see Section 32.14.4) or by certain leaf functions of the GETSEC instruction (see Chapter 7, “Safer Mode Extensions Reference,” in the *Intel*[®] 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D).
- If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

32.16 SMI AND PROCESSOR EXTENDED STATE MANAGEMENT

On processors that support processor extended states using XSAVE/XRSTOR (see Chapter 13, “Managing State Using the XSAVE Feature Set,” of the *Intel*[®] 64 and IA-32 Architectures Software Developer’s Manual, Volume 1), the processor does not save any XSAVE/XRSTOR related state on an SMI. It is the responsibility of the SMI handler code to properly preserve the state information (including CR4.OSXSAVE, XCR0, and possibly processor extended states using XSAVE/XRSTOR). Therefore, the SMI handler must follow the rules described in Chapter 13, “Managing State Using the XSAVE Feature Set,” of the *Intel*[®] 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 7, “Safer Mode Extensions Reference,” in the *Intel*[®] 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

32.17 MODEL-SPECIFIC SYSTEM MANAGEMENT ENHANCEMENT

This section describes enhancement of system management features that apply only to the 4th generation Intel Core processors. These features are model-specific. BIOS and SMM handler must use CPUID to enumerate Display-Family_DisplayModel signature when programming with these interfaces.

32.17.1 SMM Handler Code Access Control

The BIOS may choose to restrict the address ranges of code that SMM handler executes. When SMM handler code execution check is enabled, an attempt by the SMM handler to execute outside the ranges specified by SMRR (see Section 32.4.2.1) will cause the assertion of an unrecoverable machine check exception (MCE).

The interface to enable SMM handler code access check resides in a per-package scope model-specific register MSR_SMM_FEATURE_CONTROL at address 4E0H. An attempt to access MSR_SMM_FEATURE_CONTROL outside of SMM will cause a #GP. Writes to MSR_SMM_FEATURE_CONTROL is further protected by configuration interface of MSR_SMM_MCA_CAP at address 17DH.

Details of the interface of MSR_SMM_FEATURE_CONTROL and MSR_SMM_MCA_CAP are described in Table 2-29 in Chapter 2, "Model-Specific Registers (MSRs)," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

32.17.2 SMI Delivery Delay Reporting

Entry into the system management mode occurs at instruction boundary. In situations where a logical processor is executing an instruction involving a long flow of internal operations, servicing an SMI by that logical processor will be delayed. Delayed servicing of SMI of each logical processor due to executing long flows of internal operation in a physical processor can be queried via a package-scope register MSR_SMM_DELAYED at address 4E2H.

The interface to enable reporting of SMI delivery delay due to long internal flows resides in a per-package scope model-specific register MSR_SMM_DELAYED. An attempt to access MSR_SMM_DELAYED outside of SMM will cause a #GP. Availability to MSR_SMM_DELAYED is protected by configuration interface of MSR_SMM_MCA_CAP at address 17DH.

Details of the interface of MSR_SMM_DELAYED and MSR_SMM_MCA_CAP are described in Table 2-29 in Chapter 2, "Model-Specific Registers (MSRs)," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

32.17.3 Blocked SMI Reporting

A logical processor may have entered into a state and blocked from servicing other interrupts (including SMI). Logical processors in a physical processor that are blocked in servicing SMI can be queried in a package-scope register MSR_SMM_BLOCKED at address 4E3H. An attempt to access MSR_SMM_BLOCKED outside of SMM will cause a #GP.

Details of the interface of MSR_SMM_BLOCKED is described in Table 2-29 in Chapter 2, "Model-Specific Registers (MSRs)," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

17. Updates to Chapter 35, Volume 3D

Change bars and green text show changes to Chapter 35 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Updates throughout the chapter for the AEX Notify feature.

CHAPTER 35

ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

35.1 OVERVIEW OF ENCLAVE EXECUTION ENVIRONMENT

When an enclave is created, it has a range of linear addresses to which the processor applies enhanced access control. This range is called the ELRANGE (see Section 34.3). When an enclave generates a memory access, the existing IA32 segmentation and paging architecture are applied. Additionally, linear addresses inside the ELRANGE must map to an EPC page otherwise when an enclave attempts to access that linear address a fault is generated.

The EPC pages need not be physically contiguous. System software allocates EPC pages to various enclaves. Enclaves must abide by OS/VMM imposed segmentation and paging policies. OS/VMM-managed page tables and extended page tables provide address translation for the enclave pages. Hardware requires that these pages are properly mapped to EPC (any failure generates an exception).

Enclave entry must happen through specific enclave instructions:

- ENCLU[EENTER], ENCLU[ERESUME].

Enclave exit must happen through specific enclave instructions or events:

- ENCLU[EEXIT], Asynchronous Enclave Exit (AEX).

Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations. As an example a read of an enclave page may result in the return of all one's or return of cyphertext of the cache line. Writing to an enclave page may result in a dropped write or a machine check at a later time. The processor will provide the protections as described in Section 35.4 and Section 35.5 on such accesses.

35.2 TERMINOLOGY

A memory access to the ELRANGE and initiated by an instruction executed by an enclave is called a Direct Enclave Access (Direct EA).

Memory accesses initiated by certain Intel® SGX instruction leaf functions such as ECREATE, EADD, EDBGRD, EDBGWR, ELDU/ELDB, EWB, EREMOVE, EENTER, and ERESUME to EPC pages are called Indirect Enclave Accesses (Indirect EA). Table 35-1 lists additional details of the indirect EA of SGX1 and SGX2 extensions.

Direct EAs and Indirect EAs together are called Enclave Accesses (EAs).

Any memory access that is not an Enclave Access is called a non-enclave access.

35.3 ACCESS-CONTROL REQUIREMENTS

Enclave accesses have the following access-control attributes:

- All memory accesses must conform to segmentation and paging protection mechanisms.
- Code fetches from inside an enclave to a linear address outside that enclave result in a #GP(0) exception.
- Shadow-stack-load or shadow-stack-store from inside an enclave to a linear address outside that enclave results in a #GP(0) exception.
- Non-enclave accesses to EPC memory result in undefined behavior. EPC memory is protected as described in Section 35.4 and Section 35.5 on such accesses.
- EPC pages of page types PT_REG, PT_TCS, and PT_TRIM must be mapped to ELRANGE at the linear address specified when the EPC page was allocated to the enclave using ENCLS[EADD] or ENCLS[EAUG] leaf functions. Enclave accesses through other linear address result in a #PF with the PFEC.SGX bit set.

- Direct EAs to any EPC pages must conform to the currently defined security attributes for that EPC page in the EPCM. These attributes may be defined at enclave creation time (EADD) or when the enclave sets them using SGX2 instructions. The failure of these checks results in a #PF with the PFEC.SGX bit set.
 - Target page must belong to the currently executing enclave.
 - Data may be written to an EPC page if the EPCM allow write access.
 - Data may be read from an EPC page if the EPCM allow read access.
 - Instruction fetches from an EPC page are allowed if the EPCM allows execute access.
 - Shadow-stack-load from an EPC page and shadow-stack-store to an EPC page are allowed only if the page type is PT_SS_FIRST or PT_SS_REST.
 - Data writes that are not shadow-stack-store are not allowed if the EPCM page type is PT_SS_FIRST or PT_SS_REST.
 - Target page must not have a restricted page type¹ (PT_SECS, PT_TCS, PT_VA, or PT_TRIM).
 - The EPC page must not be BLOCKED.
 - The EPC page must not be PENDING.
 - The EPC page must not be MODIFIED.

35.4 SEGMENT-BASED ACCESS CONTROL

Intel SGX architecture does not modify the segment checks performed by a logical processor. All memory accesses arising from a logical processor in protected mode (including enclave access) are subject to segmentation checks with the applicable segment register.

To ensure that outside entities do not modify the enclave's logical-to-linear address translation in an unexpected fashion, ENCLU[EENTER] and ENCLU[ERESUME] check that CS, DS, ES, and SS, if usable (i.e., not null), have segment base value of zero. A non-zero segment base value for these registers results in a #GP(0).

On enclave entry either via EENTER or ERESUME, the processor saves the contents of the external FS and GS registers, and loads these registers with values stored in the TCS at build time to enable the enclave's use of these registers for accessing the thread-local storage inside the enclave. On EEXIT and AEX, the contents at time of entry are restored. On AEX, the values of FS and GS are saved in the SSA frame. On ERESUME, FS and GS are restored from the SSA frame. The details of these operations can be found in the descriptions of EENTER, ERESUME, EEXIT, and AEX flows.

35.5 PAGE-BASED ACCESS CONTROL

35.5.1 Access-control for Accesses that Originate from Non-SGX Instructions

Intel SGX builds on the processor's paging mechanism to provide page-granular access-control for enclave pages. Enclave pages are designed to be accessible only from inside the currently executing enclave if they belong to that enclave. In addition, enclave accesses must conform to the access control requirements described in Section 35.3. or through certain Intel SGX instructions. Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations.

35.5.2 Memory Accesses that Split Across ELRANGE

Memory data accesses are allowed to split across ELRANGE (i.e., a part of the access is inside ELRANGE and a part of the access is outside ELRANGE) while the processor is inside an enclave. If an access splits across ELRANGE, the

1. EPCM may allow write, read or execute access only for pages with page type PT_REG.

processor splits the access into two sub-accesses (one inside ELRANGE and the other outside ELRANGE), and each access is evaluated. A code-fetch access that splits across ELRANGE results in a #GP due to the portion that lies outside of the ELRANGE.

35.5.3 Implicit vs. Explicit Accesses

Memory accesses originating from Intel SGX instruction leaf functions are categorized as either explicit accesses or implicit accesses. Table 35-1 lists the implicit and explicit memory accesses made by Intel SGX leaf functions.

35.5.3.1 Explicit Accesses

Accesses to memory locations provided as explicit operands to Intel SGX instruction leaf functions, or their linked data structures are called explicit accesses.

Explicit accesses are always made using logical addresses. These accesses are subject to segmentation, paging, extended paging, and APIC-virtualization checks, and trigger any faults/exit associated with these checks when the access is made.

The interaction of explicit memory accesses with data breakpoints is leaf-function-specific, and is documented in Section 40.3.4.

35.5.3.2 Implicit Accesses

Accesses to data structures whose physical addresses are cached by the processor are called implicit accesses. These addresses are not passed as operands of the instruction but are implied by use of the instruction.

These accesses do not trigger any access-control faults/exits or data breakpoints. Table 35-1 lists memory objects that Intel SGX instruction leaf functions access either by explicit access or implicit access. The addresses of explicit access objects are passed via register operands with the second through fourth column of Table 35-1 matching implicitly encoded registers RBX, RCX, RDX.

Physical addresses used in different implicit accesses are cached via different instructions and for different durations. The physical address of SECS associated with each EPC page is cached at the time the page is added to the enclave via ENCLS[EADD] or ENCLS[EAUG], or when the page is loaded to EPC via ENCLS[ELDB] or ENCLS[ELDU]. This binding is severed when the corresponding page is removed from the EPC via ENCLS[EREMOVE] or ENCLS[EWB]. Physical addresses of TCS and SSA pages are cached at the time of most-recent enclave entry. Exit from an enclave (ENCLU[EEXIT] or AEX) flushes this caching. Details of Asynchronous Enclave Exit is described in Chapter 37.

The physical addresses that are cached for use by implicit accesses are derived from logical (or linear) addresses after checks such as segmentation, paging, EPT, and APIC virtualization checks. These checks may trigger exceptions or VM exits. Note, however, that such exception or VM exits may not occur after a physical address is cached and used for an implicit access.

Table 35-1. List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions

Instr. Leaf	Enum.	Explicit 1	Explicit 2	Explicit 3	Implicit
EACCEPT	SGX2	SECINFO	EPCPAGE		SECS
EACCEPTCOPY	SGX2	SECINFO	EPCPAGE (Src)	EPCPAGE (Dst)	
EADD	SGX1	PAGEINFO and linked structures	EPCPAGE		
EAUG	SGX2	PAGEINFO and linked structures	EPCPAGE		SECS
EBLOCK	SGX1	EPCPAGE			SECS
ECREATE	SGX1	PAGEINFO and linked structures	EPCPAGE		
EDBGRD	SGX1	EPCADDR	Destination		SECS
EDBGWR	SGX1	EPCADDR	Source		SECS
EDECVIRTCHILD	OVERSUB	EPCPAGE	SECS		
EENTER	SGX1	TCS and linked SSA			SECS

Table 35-1. List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions (Contd.)

Instr. Leaf	Enum.	Explicit 1	Explicit 2	Explicit 3	Implicit
EEXIT	SGX1				SECS, TCS
EEXTEND	SGX1	SECS	EPCPAGE		
EGETKEY	SGX1	KEYREQUEST	KEY		SECS
EINCVIRTCHILD	OVERSUB	EPCPAGE	SECS		
EINIT	SGX1	SIGSTRUCT	SECS	EINITTOKEN	
ELDB/ELDU	SGX1	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	
ELDBC/ELDUC	OVERSUB	PAGEINFO and linked structures	EPCPAGE	VAPAGE	
EMODPE	SGX2	SECINFO	EPCPAGE		
EMODPR	SGX2	SECINFO	EPCPAGE		SECS
EMODT	SGX2	SECINFO	EPCPAGE		SECS
EPA	SGX1	EPCADDR			
ERDINFO	OVERSUB	RDINFO	EPCPAGE		
EREMOVE	SGX1	EPCPAGE			SECS
EREPORT	SGX1	TARGETINFO	REPORTDATA	OUTPUTDATA	SECS
ERESUME	SGX1	TCS and linked SSA			SECS
ESETCONTEXT	OVERSUB		SECS	ContextValue	
ETRACK	SGX1	EPCPAGE			
ETRACKC	OVERSUB		EPCPAGE		
EWB	SGX1	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	SECS
Asynchronous Enclave Exit*					SECS, TCS, SSA

*Details of Asynchronous Enclave Exit (AEX) is described in Section 37.4

35.6 INTEL® SGX DATA STRUCTURES OVERVIEW

Enclave operation is managed via a collection of data structures. Many of the top-level data structures contain sub-structures. The top-level data structures relate to parameters that may be used in enclave setup/maintenance, by Intel SGX instructions, or AEX event. The top-level data structures are:

- SGX Enclave Control Structure (SECS)
- Thread Control Structure (TCS)
- State Save Area (SSA)
- Page Information (PAGEINFO)
- Security Information (SECINFO)
- Paging Crypto MetaData (PCMD)
- Enclave Signature Structure (SIGSTRUCT)
- EINIT Token Structure (EINITTOKEN)
- Report Structure (REPORT)
- Report Target Info (TARGETINFO)
- Key Request (KEYREQUEST)
- Version Array (VA)
- Enclave Page Cache Map (EPCM)
- Read Info (RDINFO)

Details of the top-level data structures and associated sub-structures are listed in Section 35.7 through Section 35.20.

35.7 SGX ENCLAVE CONTROL STRUCTURE (SECS)

The SECS data structure requires 4K-Bytes alignment.

Table 35-2. Layout of SGX Enclave Control Structure (SECS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
SIZE	0	8	Size of enclave in bytes; must be power of 2.
BASEADDR	8	8	Enclave Base Linear Address must be naturally aligned to size.
SSAFRAMESIZE	16	4	Size of one SSA frame in pages, including XSAVE, pad, GPR, and MISC (if CPUID.(EAX=12H, ECX=0):EBX != 0).
MISCSELECT	20	4	Bit vector specifying which extended features are saved to the MISC region (see Section 35.7.2) of the SSA frame when an AEX occurs.
CET_LEG_BITMAP_OFFSET	24	8	Page aligned offset of legacy code page bitmap from enclave base. Software is expected to program this offset such that the entire bitmap resides in the ELRANGE when legacy compatibility mode for indirect branch tracking is enabled. However this is not enforced by the hardware. This field exists when CPUID.(EAX=7, ECX=0):EDX.CET_IBT[bit 20] is enumerated as 1, else it is reserved.
CET_ATTRIBUTES	32	1	CET feature attributes of the enclave; see Table 35-5. This field exists when CPUID.(EAX=12, ECX=1):EAX[6] is enumerated as 1, else it is reserved.
RESERVED	33	15	
ATTRIBUTES	48	16	Attributes of the Enclave, see Table 35-3.
MRENCLAVE	64	32	Measurement Register of enclave build process. See SIGSTRUCT for format.
RESERVED	96	32	
MRSIGNER	128	32	Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for format.
RESERVED	160	32	
CONFIGID	192	64	Post EINIT configuration identity.
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Post EINIT configuration security version number (SVN).
RESERVED	262	3834	<p>The RESERVED field consists of the following:</p> <ul style="list-style-type: none"> ▪ EID: An 8 byte Enclave Identifier. Its location is implementation specific. ▪ PAD: A 352 bytes padding pattern from the Signature (used for key derivation strings). It's location is implementation specific. ▪ VIRTCHILDCNT: An 8 byte Count of virtual children that have been paged out by a VMM. Its location is implementation specific. ▪ ENCLAVECONTEXT: An 8 byte Enclave context pointer. Its location is implementation specific. ▪ ISVFAMILYID: A 16 byte value assigned to identify the family of products the enclave belongs to. ▪ ISVEXTPRODID: A 16 byte value assigned to identify the product identity of the enclave. ▪ The remaining 3226 bytes are reserved area. <p>The entire 3834 byte field must be cleared prior to executing ECREATE.</p>

35.7.1 ATTRIBUTES

The ATTRIBUTES data structure is comprised of bit-granular fields that are used in the SECS, the REPORT and the KEYREQUEST structures. CPUID.(EAX=12H, ECX=1) enumerates a bitmap of permitted 1-setting of bits in ATTRIBUTES.

Table 35-3. Layout of ATTRIBUTES Structure

Field	Bit Position	Description
INIT	0	This bit indicates if the enclave has been initialized by EINIT. It must be cleared when loaded as part of ECREATE. For EREPORT instruction, TARGET_INFO.ATTRIBUTES[ENIT] must always be 1 to match the state after EINIT has initialized the enclave.
DEBUG	1	If 1, the enclave permit debugger to read and write enclave data using EDBGD and EDBGWR.
MODE64BIT	2	Enclave runs in 64-bit mode.
RESERVED	3	Must be Zero.
PROVISIONKEY	4	Provisioning Key is available from EGETKEY.
EINITTOKEN_KEY	5	EINIT token key is available from EGETKEY.
CET	6	Enable CET attributes. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0 this bit is reserved and must be 0.
KSS	7	Key Separation and Sharing Enabled.
RESERVED	9:8	Must be zero.
AEXNOTIFY	10	The bit indicates that threads within the enclave may receive AEX notifications.
RESERVED	63:11	Must be zero.
XFRM	127:64	XSAVE Feature Request Mask. See Section 39.7.

35.7.2 SECS.MISCSELECT Field

CPUID.(EAX=12H, ECX=0):EBX[31:0] enumerates which extended information that the processor can save into the MISC region of SSA when an AEX occurs. An enclave writer can specify via SIGSTRUCT how to set the SECS.MISCSELECT field. The bit vector of MISCSELECT selects which extended information is to be saved in the MISC region of the SSA frame when an AEX is generated. The bit vector definition of extended information is listed in Table 35-4.

If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, SECS.MISCSELECT field must be all zeros.

The SECS.MISCSELECT field determines the size of MISC region of the SSA frame, see Section 35.9.2.

Table 35-4. Bit Vector Layout of MISCSELECT Field of Extended Information

Field	Bit Position	Description
EXINFO	0	Report information about page fault and general protection exception that occurred inside an enclave.
CPINFO	1	Report information about control protection exception that occurred inside an enclave. When CPUID.(EAX=12H, ECX=0):EBX[1] is 0, this bit is reserved.
Reserved	31:2	Reserved (0).

35.7.3 SECS.CET_ATTRIBUTES Field

The SECS.CET_ATTRIBUTES field can be used by the enclave writer to enable various CET attributes in an enclave. This field exists when CPUID.(EAX=12, ECX=1):EAX[6] is enumerated as 1. Bits 1:0 are defined when CPUID.(EAX=7, ECX=0):ECX.CET_SS is 1, and bits 5:2 are defined when CPUID.(EAX=7, ECX=0):EDX.CET_IBT is 1.

Table 35-5. Bit Vector Layout of CET_ATTRIBUTES Field of Extended Information

Field	Bit Position	Description
SH_STK_EN	0	When set to 1, enable shadow stacks.
WR_SHSTK_EN	1	When set to 1, enables the WRSS{D,Q}W instructions.
ENDBR_EN	2	When set to 1, enables indirect branch tracking.
LEG_IW_EN	3	Enable legacy compatibility treatment for indirect branch tracking.
NO_TRACK_EN	4	When set to 1, enables use of no-track prefix for indirect branch tracking.
SUPPRESS_DIS	5	When set to 1, disables suppression of CET indirect branch tracking on legacy compatibility.
Reserved	7:6	Reserved (0).

35.8 THREAD CONTROL STRUCTURE (TCS)

Each executing thread in the enclave is associated with a Thread Control Structure. It requires 4K-Bytes alignment.

Table 35-6. Layout of Thread Control Structure (TCS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
STAGE	0	8	Enclave execution state of the thread controlled by this TCS. A value of 0 indicates that this TCS is available for enclave entry. A value of 1 indicates that a logical processor is currently executing an enclave in the context of this TCS.
FLAGS	8	8	The thread's execution flags (see Section 35.8.1).
OSSA	16	8	Offset of the base of the State Save Area stack, relative to the enclave base. Must be page aligned.
CSSA	24	4	Current slot index of an SSA frame, cleared by EADD and EACCEPT.
NSSA	28	4	Number of available slots for SSA frames.
OENTRY	32	8	Offset in enclave to which control is transferred on EENTER relative to the base of the enclave.
AEP	40	8	The value of the Asynchronous Exit Pointer that was saved at EENTER time.
OFSBASE	48	8	Offset to add to the base address of the enclave for producing the base address of FS segment inside the enclave. Must be page aligned.
OGSBASE	56	8	Offset to add to the base address of the enclave for producing the base address of GS segment inside the enclave. Must be page aligned.
FSLIMIT	64	4	Size to become the new FS limit in 32-bit mode.
GSLIMIT	68	4	Size to become the new GS limit in 32-bit mode.
OCETSSA	72	8	When CPUID.(EAX=12H, ECX=1);EAX[6] is 1, this field provides the offset of the CET state save area from enclave base. When CPUID.(EAX=12H, ECX=1);EAX[6] is 0, this field is reserved and must be 0.
PREVSSP	80	8	When CPUID.(EAX=07H, ECX=00h);ECX[CET_SS] is 1, this field records the SSP at the time of AEX or EEXIT; used to setup SSP on entry. When CPUID.(EAX=07H, ECX=00h);ECX[CET_SS] is 0, this field is reserved and must be 0.
RESERVED	72	4024	Must be zero.

35.8.1 TCS.FLAGS

Table 35-7. Layout of TCS.FLAGS Field

Field	Bit Position	Description
DBGOPTIN	0	If set, allows debugging features (single-stepping, breakpoints, etc.) to be enabled and active while executing in the enclave on this TCS. Hardware clears this bit on EADD. A debugger may later modify it if the enclave's ATTRIBUTES.DEBUG is set.
AEXNOTIFY	1	A thread that enters the enclave cannot receive AEX notifications unless this flag is set to 1.
RESERVED	63:2	Must be zero.

35.8.2 State Save Area Offset (OSSA)

The OSSA points to a stack of State Save Area (SSA) frames (see Section 35.9) used to save the processor state when an interrupt or exception occurs while executing in the enclave.

35.8.3 Current State Save Area Frame (CSSA)

CSSA is the index of the current SSA frame that will be used by the processor to determine where to save the processor state on an interrupt or exception that occurs while executing in the enclave. It is an index into the array of frames addressed by OSSA. CSSA is incremented on an AEX and decremented on an ERESUME.

35.8.4 Number of State Save Area Frames (NSSA)

NSSA specifies the number of SSA frames available for this TCS. There must be at least one available SSA frame when EENTER-ing the enclave or the EENTER will fail.

35.9 STATE SAVE AREA (SSA) FRAME

When an AEX occurs while running in an enclave, the architectural state is saved in the thread's current SSA frame, which is pointed to by TCS.CSSA. An SSA frame must be page aligned, and contains the following regions:

- The XSAVE region starts at the base of the SSA frame, this region contains extended feature register state in an XSAVE/FXSAVE-compatible non-compacted format.
- A Pad region: software may choose to maintain a pad region separating the XSAVE region and the MISC region. Software choose the size of the pad region according to the sizes of the MISC and GPRSGX regions.
- The GPRSGX region. The GPRSGX region is the last region of an SSA frame (see Table 35-8). This is used to hold the processor general purpose registers (RAX ... R15), the RIP, the outside RSP and RBP, RFLAGS, and the AEX information.
- The MISC region (If CPUIDEAX=12H, ECX=0):EBX[31:0] != 0). The MISC region is adjacent to the GRPSGX region, and may contain zero or more components of extended information that would be saved when an AEX occurs. If the MISC region is absent, the region between the GPRSGX and XSAVE regions is the pad region that software can use. If the MISC region is present, the region between the MISC and XSAVE regions is the pad region that software can use. See additional details in Section 35.9.2.

Table 35-8. Top-to-Bottom Layout of an SSA Frame

Region	Offset (Byte)	Size (Bytes)	Description
XSAVE	0	Calculate using CPUID leaf ODH information	The size of XSAVE region in SSA is derived from the enclave's support of the collection of processor extended states that would be managed by XSAVE. The enablement of those processor extended state components in conjunction with CPUID leaf ODH information determines the XSAVE region size in SSA.
Pad	End of XSAVE region	Chosen by enclave writer	Ensure the end of GPRSGX region is aligned to the end of a 4KB page.
MISC	base of GPRSGX - sizeof(MISC)	Calculate from highest set bit of SECS.MISCSELECT	See Section 35.9.2.
GPRSGX	SSAFRAMESIZE - 176	176	See Table 35-9 for layout of the GPRSGX region.

35.9.1 GPRSGX Region

The layout of the GPRSGX region is shown in Table 35-9.

Table 35-9. Layout of GPRSGX Portion of the State Save Area

Field	OFFSET (Bytes)	Size (Bytes)	Description
RAX	0	8	
RCX	8	8	
RDX	16	8	
RBX	24	8	
RSP	32	8	
RBP	40	8	
RSI	48	8	
RDI	56	8	
R8	64	8	
R9	72	8	
R10	80	8	
R11	88	8	
R12	96	8	
R13	104	8	
R14	112	8	
R15	120	8	
RFLAGS	128	8	Flag register.
RIP	136	8	Instruction pointer.
URSP	144	8	Non-Enclave (outside) stack pointer. Saved by EENTER, restored on AEX.
URBP	152	8	Non-Enclave (outside) RBP pointer. Saved by EENTER, restored on AEX.
EXITINFO	160	4	Contains information about exceptions that cause AEXs, which might be needed by enclave software (see Section 35.9.1.1).
RESERVED	164	3	

Table 35-9. Layout of GPRSGX Portion of the State Save Area (Contd.)

Field	OFFSET (Bytes)	Size (Bytes)	Description
AEXNOTIFY	167	1	Bit 0: This bit allows enclave software to dynamically enable/disable AEX notifications. An enclave thread cannot receive AEX notifications unless this bit is set to 1 in the thread's current SSA frame. All other bits are reserved.
FSBASE	168	8	FS BASE.
GSBASE	176	8	GS BASE.

35.9.1.1 EXITINFO

EXITINFO contains the information used to report exit reasons to software inside the enclave. It is a 4 byte field laid out as in Table 35-10. The VALID bit is set only for the exceptions conditions which are reported inside an enclave. See Table 35-11 for which exceptions are reported inside the enclave. If the exception condition is not one reported inside the enclave then VECTOR and EXIT_TYPE are cleared.

When a higher priority event, such as SMI, and a pending debug exception occur at the same time when executing inside an enclave, the higher priority event has precedence. As an example for an SMI, the SSA exit info is zero. The debug exception will be delivered upon return from the SMI. In such cases, the EXITINFO field will not contain the information of a debug exception.

Table 35-10. Layout of EXITINFO Field

Field	Bit Position	Description
VECTOR	7:0	Exception number of exceptions reported inside enclave.
EXIT_TYPE	10:8	011b: Hardware exceptions. 110b: Software exceptions. Other values: Reserved.
RESERVED	30:11	Reserved as zero.
VALID	31	0: unsupported exceptions. 1: Supported exceptions. Includes two categories: <ul style="list-style-type: none"> • Unconditionally supported exceptions: #DE, #DB, #BP, #BR, #UD, #MF, #AC, #XM. • Conditionally supported exception: <ul style="list-style-type: none"> – #PF, #GP if SECS.MISCSELECT.EXINFO = 1. – #CP if SECS.MISCSELECT.CPINFO=1.

35.9.1.2 VECTOR Field Definition

Table 35-11 contains the VECTOR field. This field contains information about some exceptions which occur inside the enclave. These vector values are the same as the values that would be used when vectoring into regular exception handlers. All values not shown are not reported inside an enclave.

Table 35-11. Exception Vectors

Name	Vector #	Description
#DE	0	Divider exception.
#DB	1	Debug exception.
#BP	3	Breakpoint exception.
#BR	5	Bound range exceeded exception.
#UD	6	Invalid opcode exception.
#GP	13	General protection exception. Only reported if SECS.MISCSELECT.EXINFO = 1.
#PF	14	Page fault exception. Only reported if SECS.MISCSELECT.EXINFO = 1.

Table 35-11. Exception Vectors (Contd.)

Name	Vector #	Description
#MF	16	x87 FPU floating-point error.
#AC	17	Alignment check exceptions.
#XM	19	SIMD floating-point exceptions.
#CP	21	Control protection exception. Only reported if SECS.MISCSELECT.CPINFO=1.

35.9.2 MISC Region

The layout of the MISC region is shown in Table 35-12. The number of components that the processor supports in the MISC region corresponds to the bits of CPUID.(EAX=12H, ECX=0):EBX[31:0] set to 1. Each set bit in CPUID.(EAX=12H, ECX=0):EBX[31:0] has a defined size for the corresponding component, as shown in Table 35-12. Enclave writers needs to do the following:

- Decide which MISC region components will be supported for the enclave.
- Allocate an SSA frame large enough to hold the components chosen above.
- Instruct each enclave builder software to set the appropriate bits in SECS.MISCSELECT.

The first component, EXINFO, starts next to the GPRSGX region. Additional components in the MISC region grow in ascending order within the MISC region towards the XSAVE region.

The size of the MISC region is calculated as follows:

- If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISC region is not supported.
- If CPUID.(EAX=12H, ECX=0):EBX[31:0] != 0, the size of MISC region is derived from sum of the highest bit set in SECS.MISCSELECT and the size of the MISC component corresponding to that bit. Offset and size information of currently defined MISC components are listed in Table 35-12. For example, if the highest bit set in SECS.MISCSELECT is bit 0, the MISC region offset is OFFSET(GPRSGX)-16 and size is 16 bytes.
- The processor saves a MISC component *i* in the MISC region if and only if SECS.MISCSELECT[*i*] is 1.

Table 35-12. Layout of MISC region of the State Save Area

MISC Components	OFFSET (Bytes)	Size (Bytes)	Description
EXINFO	Offset(GPRSGX) -16	16	If CPUID.(EAX=12H, ECX=0):EBX[0] = 1, exception information on #GP or #PF that occurred inside an enclave can be written to the EXINFO structure if specified by SECS.MISCSELECT[0] = 1. If CPUID.(EAX=12H, ECX=0):EBX[1] = 1, exception information on #CP that occurred inside an enclave can be written to the EXINFO structure if specified by SECS.MISCSELECT[1] = 1.
Future Extension	Below EXINFO	TBD	Reserved. (Zero size if CPUID.(EAX=12H, ECX=0):EBX[31:1]=0).

35.9.2.1 EXINFO Structure

Table 35-13 contains the layout of the EXINFO structure that provides additional information.

Table 35-13. Layout of EXINFO Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MADDR	0	8	If #PF: contains the page fault linear address that caused a page fault. If #GP: the field is cleared. If #CP: the field is cleared.
ERRCD	8	4	Exception error code for either #GP or #PF.
RESERVED	12	4	

35.9.2.2 Page Fault Error Code

Table 35-14 contains page fault error code that may be reported in EXINFO.ERRCD.

Table 35-14. Page Fault Error Code

Name	Bit Position	Description
P	0	Same as non-SGX page fault exception P flag.
W/R	1	Same as non-SGX page fault exception W/R flag.
U/S ¹	2	Always set to 1 (user mode reference).
RSVD	3	Same as non-SGX page fault exception RSVD flag.
I/D	4	Same as non-SGX page fault exception I/D flag.
PK	5	Protection Key induced fault.
RSVD	14:6	Reserved.
SGX	15	EPCM induced fault.
RSVD	31:5	Reserved.

NOTES:

1. Page faults incident to enclave mode that report U/S=0 are not reported in EXINFO.

35.10 CET STATE SAVE AREA FRAME

The CET state save area consists of an array of CET state save frames. The number of CET state save frames is equal to the TCS.NSSA. The current CET SSA frame is indicated by TCS.CSSA. The offset of the CET state save area is specified by TCS.OCETSSA.

Table 35-15. Layout of CET State Save Area Frame

Field	Offset (Bytes)	Size (Bytes)	Description
SSP	0	8	Shadow Stack Pointer. This field is reserved when CPUID.(EAX=7, ECX=0):ECX[CET_SS] is 0.
IB_TRACK_STATE	8	8	Indirect branch tracker state: Bit 0: SUPPRESS - suppressed(1), tracking(0) Bit 1: TRACKER - IDLE (0), WAIT_FOR_ENDBRANCH (1) Bits 63:2 - Reserved This field is reserved when CPUID.(EAX=7, ECX=0):EDX[CET_IBT] is 0.

35.11 PAGE INFORMATION (PAGEINFO)

PAGEINFO is an architectural data structure that is used as a parameter to the EPC-management instructions. It requires 32-Byte alignment.

Table 35-16. Layout of PAGEINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
LINADDR	0	8	Enclave linear address.
SRCPGE	8	8	Effective address of the page where contents are located.
SECINFO/PCMD	16	8	Effective address of the SECINFO or PCMD (for ELDU, ELDB, EWB) structure for the page.
SECS	24	8	Effective address of EPC slot that currently contains the SECS.

35.12 SECURITY INFORMATION (SECINFO)

The SECINFO data structure holds meta-data about an enclave page.

Table 35-17. Layout of SECINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
FLAGS	0	8	Flags describing the state of the enclave page.
RESERVED	8	56	Must be zero.

35.12.1 SECINFO.FLAGS

The SECINFO.FLAGS are a set of fields describing the properties of an enclave page.

Table 35-18. Layout of SECINFO.FLAGS Field

Field	Bit Position	Description
R	0	If 1 indicates that the page can be read from inside the enclave; otherwise the page cannot be read from inside the enclave.
W	1	If 1 indicates that the page can be written from inside the enclave; otherwise the page cannot be written from inside the enclave.
X	2	If 1 indicates that the page can be executed from inside the enclave; otherwise the page cannot be executed from inside the enclave.
PENDING	3	If 1 indicates that the page is in the PENDING state; otherwise the page is not in the PENDING state.
MODIFIED	4	If 1 indicates that the page is in the MODIFIED state; otherwise the page is not in the MODIFIED state.
PR	5	If 1 indicates that a permission restriction operation on the page is in progress, otherwise a permission restriction operation is not in progress.
RESERVED	7:6	Must be zero.
PAGE_TYPE	15:8	The type of page that the SECINFO is associated with.
RESERVED	63:16	Must be zero.

35.12.2 PAGE_TYPE Field Definition

The SECINFO flags and EPC flags contain bits indicating the type of page.

Table 35-19. Supported PAGE_TYPE

TYPE	Value	Description
PT_SECS	0	Page is an SECS.
PT_TCS	1	Page is a TCS.
PT_REG	2	Page is a regular page.
PT_VA	3	Page is a Version Array.
PT_TRIM	4	Page is in trimmed state.
PT_SS_FIRST	5	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, Page is first page of a shadow stack. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this value is reserved.
PT_SS_REST	6	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, Page is not first page of a shadow stack. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this value is reserved.
	All others	Reserved.

35.13 PAGING CRYPTO METADATA (PCMD)

The PCMD structure is used to keep track of crypto meta-data associated with a paged-out page. Combined with PAGEINFO, it provides enough information for the processor to verify, decrypt, and reload a paged-out EPC page. The size of the PCMD structure (128 bytes) is architectural.

EWB calculates the Message Authentication Code (MAC) value and writes out the PCMD. ELDB/U reads the fields and checks the MAC.

The format of PCMD is as follows:

Table 35-20. Layout of PCMD Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
SECINFO	0	64	Flags describing the state of the enclave page; R/W by software.
ENCLAVEID	64	8	Enclave Identifier used to establish a cryptographic binding between paged-out page and the enclave.
RESERVED	72	40	Must be zero.
MAC	112	16	Message Authentication Code for the page, page meta-data and reserved field.

35.14 ENCLAVE SIGNATURE STRUCTURE (SIGSTRUCT)

SIGSTRUCT is a structure created and signed by the enclave developer that contains information about the enclave. SIGSTRUCT is processed by the EINIT leaf function to verify that the enclave was properly built.

SIGSTRUCT includes ENCLAVEHASH as SHA256 digest, as defined in FIPS PUB 180-4. The digests are byte strings of length 32. Each of the 8 HASH dwords is stored in little-endian order.

SIGSTRUCT includes four 3072-bit integers (MODULUS, SIGNATURE, Q1, Q2). Each such integer is represented as a byte strings of length 384, with the most significant byte at the position "offset + 383", and the least significant byte at position "offset".

The (3072-bit integer) SIGNATURE should be an RSA signature, where: a) the RSA modulus (MODULUS) is a 3072-bit integer; b) the public exponent is set to 3; c) the signing procedure uses the EMSA-PKCS1-v1.5 format with DER encoding of the "DigestInfo" value as specified in of PKCS#1 v2.1/RFC 3447.

The 3072-bit integers Q1 and Q2 are defined by:

$q1 = \text{floor}(\text{Signature}^2 / \text{Modulus});$

$q2 = \text{floor}((\text{Signature}^3 - q1 * \text{Signature} * \text{Modulus}) / \text{Modulus});$

SIGSTRUCT must be page aligned

In column 5 of Table 35-21, 'Y' indicates that this field should be included in the signature generated by the developer.

Table 35-21. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
HEADER	0	16	Must be byte stream 06000000E10000000000010000000000H	Y
VENDOR	16	4	Intel Enclave: 00008086H Non-Intel Enclave: 00000000H	Y
DATE	20	4	Build date is yyyyymmdd in hex: yyyy=4 digit year, mm=1-12, dd=1-31	Y
HEADER2	24	16	Must be byte stream 01010000600000006000000001000000H	Y
SWDEFINED	40	4	Available for software use.	Y

Table 35-21. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
RESERVED	44	84	Must be zero.	Y
MODULUS	128	384	Module Public Key (keylength=3072 bits).	N
EXPONENT	512	4	RSA Exponent = 3.	N
SIGNATURE	516	384	Signature over Header and Body.	N
MISCSELECT*	900	4	Bit vector specifying Extended SSA frame feature set to be used.	Y
MISCMASK*	904	4	Bit vector mask of MISCSELECT to enforce.	Y
CET_ATTRIBUTES	908	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Enclave CET attributes that must be set. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.	Y
CET_ATTRIBUTES_MASK	909	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Mask of CET attributes to enforce. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.	Y
RESERVED	910	2	Must be zero.	Y
ISVFAMILYID	912	16	ISV assigned Product Family ID.	Y
ATTRIBUTES	928	16	Enclave Attributes that must be set.	Y
ATTRIBUTEMASK	944	16	Mask of Attributes to enforce.	Y
ENCLAVEHASH	960	32	MRENCLAVE of enclave this structure applies to.	Y
RESERVED	992	16	Must be zero.	Y
ISVEXTPRODID	1008	16	ISV assigned extended Product ID.	Y
ISVPRODID	1024	2	ISV assigned Product ID.	Y
ISVSVN	1026	2	ISV assigned SVN (security version number).	Y
RESERVED	1028	12	Must be zero.	N
Q1	1040	384	Q1 value for RSA Signature Verification.	N
Q2	1424	384	Q2 value for RSA Signature Verification.	N
* If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISCSELECT must be 0. If CPUID.(EAX=12H, ECX=0):EBX[31:0] !=0, enclave writers must specify MISCSELECT such that each cleared bit in MISCMASK must also specify the corresponding bit as 0 in MISCSELECT.				

35.15 EINIT TOKEN STRUCTURE (EINITTOKEN)

The EINIT token is used by EINIT to verify that the enclave is permitted to launch. EINIT token is generated by an enclave in possession of the EINITTOKEN key (the Launch Enclave).

EINIT token must be 512-Byte aligned.

Table 35-22. Layout of EINIT Token (EINITTOKEN)

Field	OFFSET (Bytes)	Size (Bytes)	MACed	Description
Valid	0	4	Y	Bit 0: 1: Valid; 0: Invalid. All other bits reserved.
RESERVED	4	44	Y	Must be zero.
ATTRIBUTES	48	16	Y	ATTRIBUTES of the Enclave.
MRENCLAVE	64	32	Y	MRENCLAVE of the Enclave.
RESERVED	96	32	Y	Reserved.
MRSIGNER	128	32	Y	MRSIGNER of the Enclave.
RESERVED	160	32	Y	Reserved.
CPUSVNLE	192	16	N	Launch Enclave's CPUSVN.
ISVPRODIDLE	208	02	N	Launch Enclave's ISVPRODID.
ISVSVNLE	210	02	N	Launch Enclave's ISVSVN.
CET_MASKED_ATTRIBUTES_LE	212	1	N	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Launch enclaves masked CET attributes. This should be set to LE's CET_ATTRIBUTES masked with CET_ATTRIBUTES_MASK of the LE's KEYREQUEST. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved.
RESERVED	213	23	N	Reserved.
MASKEDMISCSELECTLE	236	4		Launch Enclave's MASKEDMISCSELECT: set by the LE to the resolved MISCSELECT value, used by EGETKEY (after applying KEYREQUEST's masking).
MASKEDATTRIBUTESLE	240	16	N	Launch Enclave's MASKEDATTRIBUTES: This should be set to the LE's ATTRIBUTES masked with ATTRIBUTEMASK of the LE's KEYREQUEST.
KEYID	256	32	N	Value for key wear-out protection.
MAC	288	16	N	Message Authentication Code on EINITTOKEN using EINITTOKEN_KEY.

35.16 REPORT (REPORT)

The REPORT structure is the output of the EREPORT instruction, and must be 512-Byte aligned.

Table 35-23. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CPUSVN	0	16	The security version number of the processor.
MISCSELECT	16	4	Bit vector specifying which extended features are saved to the MISC region of the SSA frame when an AEX occurs.
CET_ATTRIBUTES	20	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field reports the CET_ATTRIBUTES of the Enclave. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.
RESERVED	21	11	Zero.
ISVEXTNPRODID	32	16	The value of SECS.ISVEXTPRODID.
ATTRIBUTES	48	16	ATTRIBUTES of the Enclave. See Section 35.7.1.
MRENCLAVE	64	32	The value of SECS.MRENCLAVE.
RESERVED	96	32	Zero.
MRSIGNER	128	32	The value of SECS.MRSIGNER.
RESERVED	160	32	Zero.

Table 35-23. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CONFIGID	192	64	Value provided by SW to identify enclave's post EINIT configuration.
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Value provided by SW to indicate expected SVN of enclave's post EINIT configuration.
RESERVED	262	42	Zero.
ISVFAMILYID	304	16	The value of SECS.ISVFAMILYID.
REPORTDATA	320	64	Data provided by the user and protected by the REPORT's MAC, see Section 35.16.1.
KEYID	384	32	Value for key wear-out protection.
MAC	416	16	Message Authentication Code on the report using report key.

35.16.1 REPORTDATA

REPORTDATA is a 64-Byte data structure that is provided by the enclave and included in the REPORT. It can be used to securely pass information from the enclave to the target enclave.

35.17 REPORT TARGET INFO (TARGETINFO)

This structure is an input parameter to the EREPORT leaf function. The address of TARGETINFO is specified as an effective address in RBX. It is used to identify the target enclave which will be able to cryptographically verify the REPORT structure returned by EREPORT. TARGETINFO must be 512-Byte aligned.

Table 35-24. Layout of TARGETINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MEASUREMENT	0	32	The MRENCLAVE of the target enclave.
ATTRIBUTES	32	16	The ATTRIBUTES field of the target enclave.
CET_ATTRIBUTES	48	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the CET_ATTRIBUTES field of the target enclave. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved.
RESERVED	49	1	Must be zero.
CONFIGSVN	50	2	CONFIGSVN of the target enclave.
MISCSELECT	52	4	The MISCSELECT of the target enclave.
RESERVED	56	8	Must be zero.
CONFIGID	64	64	CONFIGID of target enclave.
RESERVED	128	384	Must be zero.

35.18 KEY REQUEST (KEYREQUEST)

This structure is an input parameter to the EGETKEY leaf function. It is passed in as an effective address in RBX and must be 512-Byte aligned. It is used for selecting the appropriate key and any additional parameters required in the derivation of that key.

Table 35-25. Layout of KEYREQUEST Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	2	Identifies the Key Required.
KEYPOLICY	2	2	Identifies which inputs are required to be used in the key derivation.
ISVSVN	4	2	The ISV security version number that will be used in the key derivation.
CET_ATTRIBUTES_MASK	6	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides a mask that defines which CET_ATTRIBUTES bits will be included in key derivation. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, then this field is reserved and must be 0.
RESERVED	7	1	Must be zero.
CPUSVN	8	16	The security version number of the processor used in the key derivation.
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in key derivation.
KEYID	40	32	Value for key wear-out protection.
MISCMASK	72	4	A mask defining which MISCSELECT bits will be included in key derivation.
CONFIGSVN	76	2	Identifies which enclave Configuration's Security Version should be used in key derivation.
RESERVED	78	434	

35.18.1 KEY REQUEST KeyNames

Table 35-26. Supported KEYName Values

Key Name	Value	Description
EINIT_TOKEN_KEY	0	EINIT_TOKEN key
PROVISION_KEY	1	Provisioning Key
PROVISION_SEAL_KEY	2	Provisioning Seal Key
REPORT_KEY	3	Report Key
SEAL_KEY	4	Seal Key
	All others	Reserved

35.18.2 Key Request Policy Structure

Table 35-27. Layout of KEYPOLICY Field

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register.
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register.
NOISVPRODID	2	If 1, derive key WITHOUT using the enclave' ISVPRODID value.
CONFIGID	3	If 1, derive key using the enclave's CONFIGID value.
ISVFAMILYID	4	If 1, derive key using the enclave ISVFAMILYID value.
ISVEXTPRODID	5	If 1, derive key using enclave's ISVEXTPRODID value.
RESERVED	15:6	Must be zero.

35.19 VERSION ARRAY (VA)

In order to securely store the versions of evicted EPC pages, Intel SGX defines a special EPC page type called a Version Array (VA). Each VA page contains 512 slots, each of which can contain an 8-byte version number for a page evicted from the EPC. When an EPC page is evicted, software chooses an empty slot in a VA page; this slot receives the unique version number of the page being evicted. When the EPC page is reloaded, there must be a VA slot that must hold the version of the page. If the page is successfully reloaded, the version in the VA slot is cleared.

VA pages can be evicted, just like any other EPC page. When evicting a VA page, a version slot in some other VA page must be used to hold the version for the VA being evicted. A Version Array Page must be 4K-Bytes aligned.

Table 35-28. Layout of Version Array Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
Slot 0	0	8	Version Slot 0
Slot 1	8	8	Version Slot 1
...			
Slot 511	4088	8	Version Slot 511

35.20 ENCLAVE PAGE CACHE MAP (EPCM)

EPCM is a secure structure used by the processor to track the contents of the EPC. The EPCM holds exactly one entry for each page that is currently loaded into the EPC. EPCM is not accessible by software, and the layout of EPCM fields is implementation specific.

Table 35-29. Content of an Enclave Page Cache Map Entry

Field	Description
VALID	Indicates whether the EPCM entry is valid.
R	Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.
W	Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.
X	Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.
PT	EPCM page type (PT_SECS, PT_TCS, PT_REG, PT_VA, PT_TRIM, PT_SS_FIRST, PT_SS_REST).
ENCLAVESECS	SECS identifier of the enclave to which the EPC page belongs.
ENCLAVEADDRESS	Linear enclave address of the EPC page.
BLOCKED	Indicates whether the EPC page is in the blocked state.
PENDING	Indicates whether the EPC page is in the pending state.
MODIFIED	Indicates whether the EPC page is in the modified state.
PR	Indicates whether the EPC page is in a permission restriction state.

35.21 READ INFO (RDINFO)

The RDINFO structure contains status information about an EPC page. It must be aligned to 32-Bytes.

Table 35-30. Layout of RDINFO Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
STATUS	0	8	Page status information.
FLAGS	8	8	EPCM state of the page.
ENCLAVECONTEXT	16	8	Context pointer describing the page's parent location.

35.21.1 RDINFO Status Structure

Table 35-31. Layout of RDINFO STATUS Structure

Field	Bit Position	Description
CHILDPRESENT	0	Indicates that the page has one or more child pages present (always zero for non-SECS pages). In VMX non-root operation includes the presence of virtual children.
VIRTCHLDPRESENT	1	Indicates that the page has one or more virtual child pages present (always zero for non-SECS pages). In VMX non-root operation this value is always zero.
RESERVED	63:2	

35.21.2 RDINFO Flags Structure

Table 35-32. Layout of RDINFO FLAGS Structure

Field	Bit Position	Description
R	0	Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.
W	1	Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.
X	2	Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.
PENDING	3	Indicates whether the EPC page is in the pending state.
MODIFIED	4	Indicates whether the EPC page is in the modified state.
PR	5	Indicates whether the EPC page is in a permission restriction state.
RESERVED	7:6	
PAGE_TYPE	15:8	Indicates the page type of the EPC page.
RESERVED	62:16	
BLOCKED	63	Indicates whether the EPC page is in the blocked state.

18. Updates to Chapter 36, Volume 3D

Change bars and green text show changes to Chapter 36 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Updates throughout the chapter for the AEX Notify feature.

CHAPTER 36

ENCLAVE OPERATION

The following aspects of enclave operation are described in this chapter:

- Enclave creation: Includes loading code and data from outside of enclave into the EPC and establishing the enclave entity.
- Adding pages and measuring the enclave.
- Initialization of an enclave: Finalizes the cryptographic log and establishes the enclave identity and sealing identity.
- Enclave entry and exiting including:
 - Controlled entry and exit.
 - Asynchronous Enclave Exit (AEX) and resuming execution after an AEX.

36.1 CONSTRUCTING AN ENCLAVE

Figure 36-1 illustrates a typical Enclave memory layout.

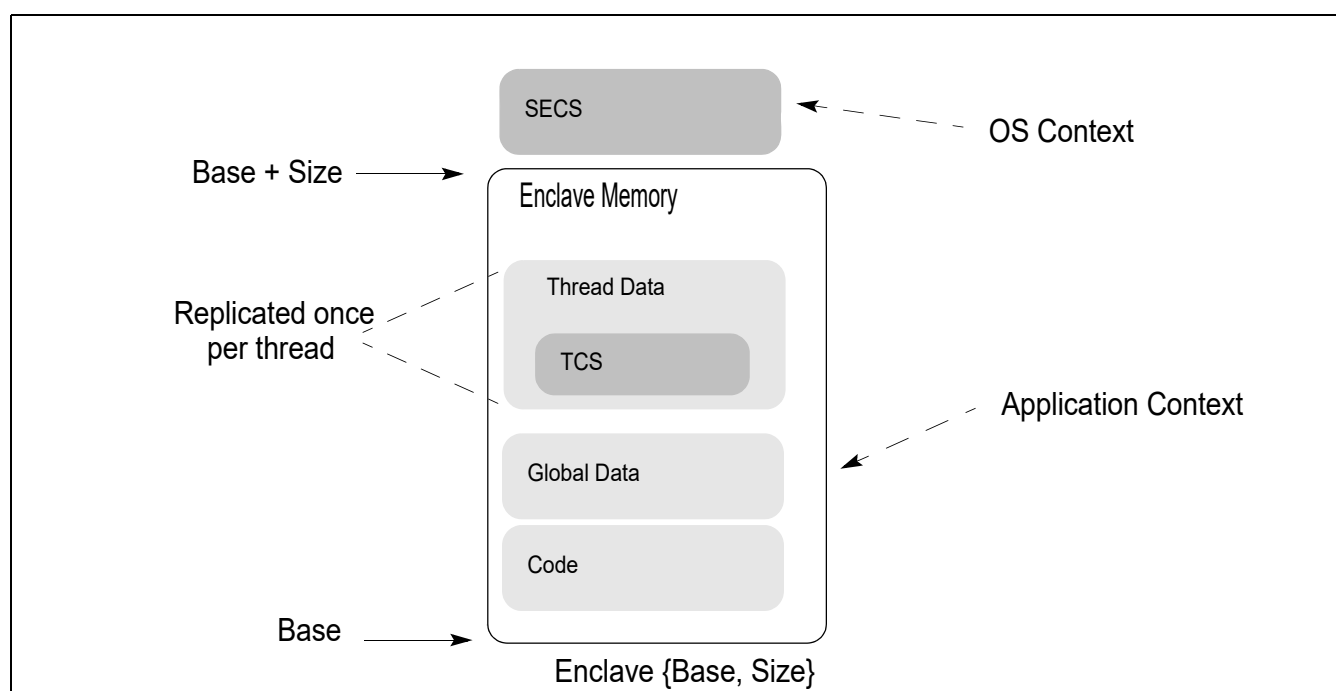


Figure 36-1. Enclave Memory Layout

The enclave creation, commitment of memory resources, and finalizing the enclave's identity with measurement comprises multiple phases. This process can be illustrated by the following exemplary steps:

1. The application hands over the enclave content along with additional information required by the enclave creation API to the enclave creation service running at privilege level 0.
2. The enclave creation service running at privilege level 0 uses the ECREATE leaf function to set up the initial environment, specifying base address and size of the enclave. This address range, the ELRANGE, is part of the application's address space. This reserves the memory range. The enclave will now reside in this address

region. ECREATE also allocates an Enclave Page Cache (EPC) page for the SGX Enclave Control Structure (SECS). Note that this page is not required to be a part of the enclave linear address space and is not required to be mapped into the process.

3. The enclave creation service uses the EADD leaf function to commit EPC pages to the enclave, and use EEXTEND to measure the committed memory content of the enclave. For each page to be added to the enclave:
 - Use EADD to add the new page to the enclave.
 - If the enclave developer requires measurement of the page as a proof for the content, use EEXTEND to add a measurement for 256 bytes of the page. Repeat this operation until the entire page is measured.
4. The enclave creation service uses the EINIT leaf function to complete the enclave creation process and finalize the enclave measurement to establish the enclave identity. Until an EINIT is executed, the enclave is not permitted to execute any enclave code (i.e., entering the enclave by executing EENTER would result in a fault).

36.1.1 ECREATE

The ECREATE leaf function sets up the initial environment for the enclave by reading an SGX Enclave Control Structure (SECS) that contains the enclave's address range (ELRANGE) as defined by BASEADDR and SIZE, the ATTRIBUTES and MISCSELECT bitmaps, and the SSAFRAMESIZE. It then securely stores this information in an Enclave Page Cache (EPC) page. ELRANGE is part of the application's address space. ECREATE also initializes a cryptographic log of the enclave's build process.

36.1.2 EADD and EEXTEND Interaction

Once the SECS has been created, enclave pages can be added to the enclave via EADD. This involves converting a free EPC page into either a PT_REG or a PT_TCS page.

When EADD is invoked, the processor will update the EPCM entry with the type of page (PT_REG or PT_TCS), the linear address used by the enclave to access the page, and the enclave access permissions for the page. It associates the page to the SECS provided as input. The EPCM entry information is used by hardware to manage access control to the page. EADD records EPCM information in the cryptographic log stored in the SECS and copies 4 KBytes of data from unprotected memory outside the EPC to the allocated EPC page.

System software is responsible for selecting a free EPC page. System software is also responsible for providing the type of page to be added, the attributes of the page, the contents of the page, and the SECS (enclave) to which the page is to be added as requested by the application. Incorrect data would lead to a failure of EADD or to an incorrect cryptographic log and a failure at EINIT time.

After a page has been added to an enclave, software can measure a 256 byte region as determined by the developer by invoking EEXTEND. Thus to measure an entire 4KB page, system software must execute EEXTEND 16 times. Each invocation of EEXTEND adds to the cryptographic log information about which region is being measured and the measurement of the section.

Entries in the cryptographic log define the measurement of the enclave and are critical in gaining assurance that the enclave was correctly constructed by the untrusted system software.

36.1.3 EINIT Interaction

Once system software has completed the process of adding and measuring pages, the enclave needs to be initialized by the EINIT leaf function. After an enclave is initialized, EADD and EEXTEND are disabled for that enclave (An attempt to execute EADD/EEXTEND to enclave after enclave initialization will result in a fault). The initialization process finalizes the cryptographic log and establishes the **enclave identity** and **sealing identity** used by EGETKEY and EREPORT.

A cryptographic hash of the log is stored as the **enclave identity**. Correct construction of the enclave results in the cryptographic hash matching the one built by the enclave owner and included as the ENCLAVEHASH field of SIGSTRUCT. The **enclave identity** provided by the EREPORT leaf function can be verified by a remote party.

The EINIT leaf function checks the EINIT token to validate that the enclave has been enabled on this platform. If the enclave is not correctly constructed, or the EINIT token is not valid for the platform, or SIGSTRUCT isn't properly signed, then EINIT will fail. See the EINIT leaf function for details on the error reporting.

The **enclave identity** is a cryptographic hash that reflects the enclave attributes and MISCSELECT value, content of the enclave, the order in which it was built, the addresses it occupies in memory, the security attributes, and access right permissions of each page. The **enclave identity** is established by the EINIT leaf function.

The **sealing identity** is managed by a sealing authority represented by the hash of the public key used to sign the SIGSTRUCT structure processed by EINIT. The sealing authority assigns a product ID (ISVPRODID) and security version number (ISVSVN) to a particular enclave identity.

EINIT establishes the sealing identity using the following steps:

1. Verifies that SIGSTRUCT is properly signed using the public key enclosed in the SIGSTRUCT.
2. Checks that the measurement of the enclave matches the measurement of the enclave specified in SIGSTRUCT.
3. Checks that the enclave's attributes and MISCSELECT values are compatible with those specified in SIGSTRUCT.
4. Finalizes the measurement of the enclave and records the **sealing identity** (the sealing authority, product id and security version number) and **enclave identity** in the SECS.
5. Sets the ATTRIBUTES.INIT bit for the enclave.

36.1.4 Intel® SGX Launch Control Configuration

Intel® SGX Launch Control is a set of controls that govern the creation of enclaves. Before the EINIT leaf function will successfully initialize an enclave, a designated Launch Enclave must create an EINITTOKEN for that enclave. Launch Enclaves have SECS.ATTRIBUTES.EINITTOKEN_KEY = 1, granting them access to the EINITTOKEN_KEY from the EGETKEY leaf function. EINITTOKEN_KEY must be used by the Launch Enclave when computing EINITTOKEN.MAC, the Message Authentication Code of the EINITTOKEN.

The hash of the public key used to sign the SIGSTRUCT of the Launch Enclave must equal the value in the IA32_SGXLEPUBKEYHASH MSRs. Only Launch Enclaves are allowed to launch without a valid token.

The IA32_SGXLEPUBKEYHASH MSRs are provided to designate the platform's Launch Enclave. IA32_SGXLEPUBKEYHASH defaults to digest of Intel's launch enclave signing key after reset.

IA32_FEATURE_CONTROL bit 17 controls the permissions on the IA32_SGXLEPUBKEYHASH MSRs when CPUID.(EAX=12H, ECX=00H):EAX[0] = 1. If IA32_FEATURE_CONTROL is locked with bit 17 set, IA32_SGXLEPUBKEYHASH MSRs are reconfigurable (writeable). If either IA32_FEATURE_CONTROL is not locked or bit 17 is clear, the MSRs are read only. By leaving these MSRs writable, system SW or a VMM can support a plurality of Launch Enclaves for hosting multiple execution environments. See Table 40.2.2 for more details.

36.2 ENCLAVE ENTRY AND EXITING

36.2.1 Controlled Entry and Exit

The EENTER leaf function is the method to enter the enclave under program control. To execute EENTER, software must supply an address of a TCS that is part of the enclave to be entered. The TCS holds the location inside the enclave to transfer control to and a pointer to the SSA frame inside the enclave that an AEX should store the register state to.

When a logical processor enters an enclave, the TCS is considered busy until the logical processors exits the enclave. An attempt to enter an enclave through a busy TCS results in a fault. Intel® SGX allows an enclave builder to define multiple TCSs, thereby providing support for multithreaded enclaves.

Software must also supply to EENTER the Asynchronous Exit Pointer (AEP) parameter. AEP is an address external to the enclave which an exception handler will return to using IRET. Typically the location would contain the ERESUME instruction. ERESUME transfers control back to the enclave, to the address retrieved from the enclave thread's saved state.

EENTER performs the following operations:

ENCLAVE OPERATION

1. Check that TCS is not busy and flush all cached linear-to-physical mappings.
2. Change the mode of operation to be in enclave mode.
3. Save the old RSP, RBP for later restore on AEX (Software is responsible for setting up the new RSP, RBP to be used inside enclave).
4. Save XCR0 and replace it with the XFRM value for the enclave.
5. Check if software wishes to debug (applicable to a debuggable enclave):
 - If not debugging, then configure hardware so the enclave appears as a single instruction.
 - If debugging, then configure hardware to allow traps, breakpoints, and single steps inside the enclave.
6. Set the TCS as busy.
7. Transfer control from outside enclave to predetermined location inside the enclave specified by the TCS.

The EEXIT leaf function is the method of leaving the enclave under program control. EEXIT receives the target address outside of the enclave that the enclave wishes to transfer control to. It is the responsibility of enclave software to erase any secret from the registers prior to invoking EEXIT. To allow enclave software to easily perform an external function call and re-enter the enclave (using EEXIT and EENTER leaf functions), EEXIT returns the value of the AEP that was used when the enclave was entered.

EEXIT performs the following operations:

1. Clear enclave mode and flush all cached linear-to-physical mappings.
2. Mark TCS as not busy.
3. Transfer control from inside the enclave to a location on the outside specified as parameter to the EEXIT leaf function.

36.2.2 Asynchronous Enclave Exit (AEX)

Asynchronous and synchronous events, such as exceptions, interrupts, traps, SMIs, and VM exits may occur while executing inside an enclave. These events are referred to as Enclave Exiting Events (EEE). Upon an EEE, the processor state is securely saved inside the enclave (in the thread's current SSA frame) and then replaced by a synthetic state to prevent leakage of secrets. The process of securely saving state and establishing the synthetic state is called an Asynchronous Enclave Exit (AEX). Details of AEX is described in Chapter 37, "Enclave Exiting Events."

As part of most EEEs, the AEP is pushed onto the stack as the location of the eventing address. This is the location where control will return to after executing the IRET. The ERESUME leaf function can be executed from that point to reenter the enclave and resume execution from the interrupted point.

After AEX has completed, the logical processor is no longer in enclave mode and the exiting event is processed normally. Any new events that occur after the AEX has completed are treated as having occurred outside the enclave (e.g., a #PF in dispatching to an interrupt handler).

36.2.3 Resuming Execution After AEX

After system software has serviced the event that caused the logical processor to exit an enclave, the logical processor can continue enclave execution using ERESUME. ERESUME restores processor state and returns control to where execution was interrupted.

If the cause of the exit was an exception or a fault and was not resolved, the event will be triggered again if the enclave is re-entered using ERESUME. For example, if an enclave performs a divide by 0 operation, executing ERESUME will cause the enclave to attempt to re-execute the faulting instruction and result in another divide by 0 exception. Intel® SGX provides the means for an enclave developer to handle enclave exceptions from within the enclave. Software can enter the enclave at a different location and invoke the exception handler within the enclave by executing the EENTER leaf function. The exception handler within the enclave can read the fault information from the SSA frame and attempt to resolve the faulting condition or simply return and indicate to software that the enclave should be terminated (e.g., using EEXIT).

36.2.3.1 ERESUME Interaction

ERESUME restores registers depending on the mode of the enclave (32 or 64 bit).

- In 32-bit mode (IA32_EFER.LMA = 0 || CS.L = 0), the low 32-bits of the legacy registers (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP, and EFLAGS) are restored from the thread's GPR area of the current SSA frame. Neither the upper 32 bits of the legacy registers nor the 64-bit registers (R8 ... R15) are loaded.
- In 64-bit mode (IA32_EFER.LMA = 1 && CS.L = 1), all 64 bits of the general processor registers (RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8 ... R15, RIP, and RFLAGS) are loaded.

Extended features specified by SECS.ATTRIBUTES.XFRM are restored from the XSAVE area of the current SSA frame. The layout of the x87 area depends on the current values of IA32_EFER.LMA and CS.L:

- IA32_EFER.LMA = 0 || CS.L = 0
 - 32-bit load in the same format that XSAVE/FXSAVE uses with these values.
- IA32_EFER.LMA = 1 && CS.L = 1
 - 64-bit load in the same format that XSAVE/FXSAVE uses with these values as if REX.W = 1.

36.2.3.2 Asynchronous Enclave Exit Notify and EDECCSSA

Asynchronous Enclave Exit Notify (AEX-Notify) is an extension to Intel SGX that allows Intel SGX enclaves to be notified after an asynchronous enclave exit (AEX) has occurred. EDECCSSA is a new Intel SGX user leaf function (ENCLU[EDECCSSA]) that can facilitate AEX notification handling, as well as software exception handling. This section provides information about changes to the Intel SGX architecture that support AEX-Notify and ENCLU[EDECCSSA].

NOTE

On some platforms, AEX-Notify and the EDECCSSA user leaf function may be enumerated by CPUID following a microcode update.

The following list summarizes the additions to existing Intel SGX data structures to support AEX-Notify:

- SECS.ATTRIBUTES.AEXNOTIFY: This enclave supports AEX-Notify.
- TCS.FLAGS.AEXNOTIFY: This enclave thread may receive AEX notifications.
- SSA.GPRSGX.AEXNOTIFY: Enclave-writable byte that allows enclave software to dynamically enable/disable AEX notifications.

An AEX notification is delivered by ENCLU[ERESUME] when the following conditions are met:

1. TCS.FLAGS.AEXNOTIFY is set.
2. TCS.CSSA (the current slot index of an SSA frame) is greater than zero.
3. TCS.SSA[TCS.CSSA-1].GPRSGX.AEXNOTIFY[0] is set.

Note that AEX increments TCS.CSSA, and ENCLU[ERESUME] decrements TCS.CSSA, except when an AEX notification is delivered. Instead of decrementing TCS.CSSA and restoring state from the SSA, ENCLU[ERESUME] delivers an AEX notification by behaving as ENCLU[EENTER]. Implications of this behavior include:

- The enclave thread is resumed at EnclaveBase + TCS.OENTRY.
- EAX contains the (non-decremented) value of TCS.CSSA.
- RCX contains the address of the IP following ENCLU[ERESUME].
- The architectural state saved by the most recent AEX is preserved in TCS.SSA[TCS.CSSA-1].

The enclave thread can return to the previous SSA context by invoking ENCLU[EDECCSSA], which decrements TCS.CSSA.

36.3 CALLING ENCLAVE PROCEDURES

36.3.1 Calling Convention

In standard call conventions subroutine parameters are generally pushed onto the stack. The called routine, being aware of its own stack layout, knows how to find parameters based on compile-time-computable offsets from the SP or BP register (depending on runtime conventions used by the compiler).

Because of the stack switch when calling an enclave, stack-located parameters cannot be found in this manner. Entering the enclave requires a modified parameter passing convention.

For example, the caller might push parameters onto the untrusted stack and then pass a pointer to those parameters in RAX to the enclave software. The exact choice of calling conventions is up to the writer of the edge routines; be those routines hand-coded or compiler generated.

36.3.2 Register Preservation

As with most systems, it is the responsibility of the callee to preserve all registers except that used for returning a value. This is consistent with conventional usage and tends to optimize the number of register save/restore operations that need be performed. It has the additional security result that it ensures that data is scrubbed from any registers that were used by enclave to temporarily contain secrets.

36.3.3 Returning to Caller

No registers are modified during EEXIT. It is the responsibility of software to remove secrets in registers before executing EEXIT.

36.4 INTEL® SGX KEY AND ATTESTATION

36.4.1 Enclave Measurement and Identification

During the enclave build process, two “measurements” are taken of each enclave and are stored in two 256-bit Measurement Registers (MR): MRENCLAVE and MRSIGNER. MRENCLAVE represents the enclave's contents and build process. MRSIGNER represents the entity that signed the enclave's SIGSTRUCT.

The values of the Measurement Registers are included in attestations to identify the enclave to remote parties. The MRs are also included in most keys, binding keys to enclaves with specific MRs.

36.4.1.1 MRENCLAVE

MRENCLAVE is a unique 256 bit value that identifies the code and data that was loaded into the enclave during the initial launch. It is computed as a SHA256 hash that is initialized by the ECREATE leaf function. EADD and EEXTEND leaf functions record information about each page and the content of those pages. The EINIT leaf function finalizes the hash, which is stored in SECS.MRENCLAVE. Any tampering with the build process, contents of a page, page permissions, etc will result in a different MRENCLAVE value.

Figure 36-2 illustrates a simplified flow of changes to the MRENCLAVE register when building an enclave:

- Enclave creation with ECREATE.
- Copying a non-enclave source page into the EPC of an un-initialized enclave with EADD.
- Updating twice of the MRENCLAVE after modifying the enclave’s page content, i.e., EEXTEND twice.
- Finalizing the enclave build with EINIT.

Details on specific values inserted in the hash are available in the individual instruction definitions.

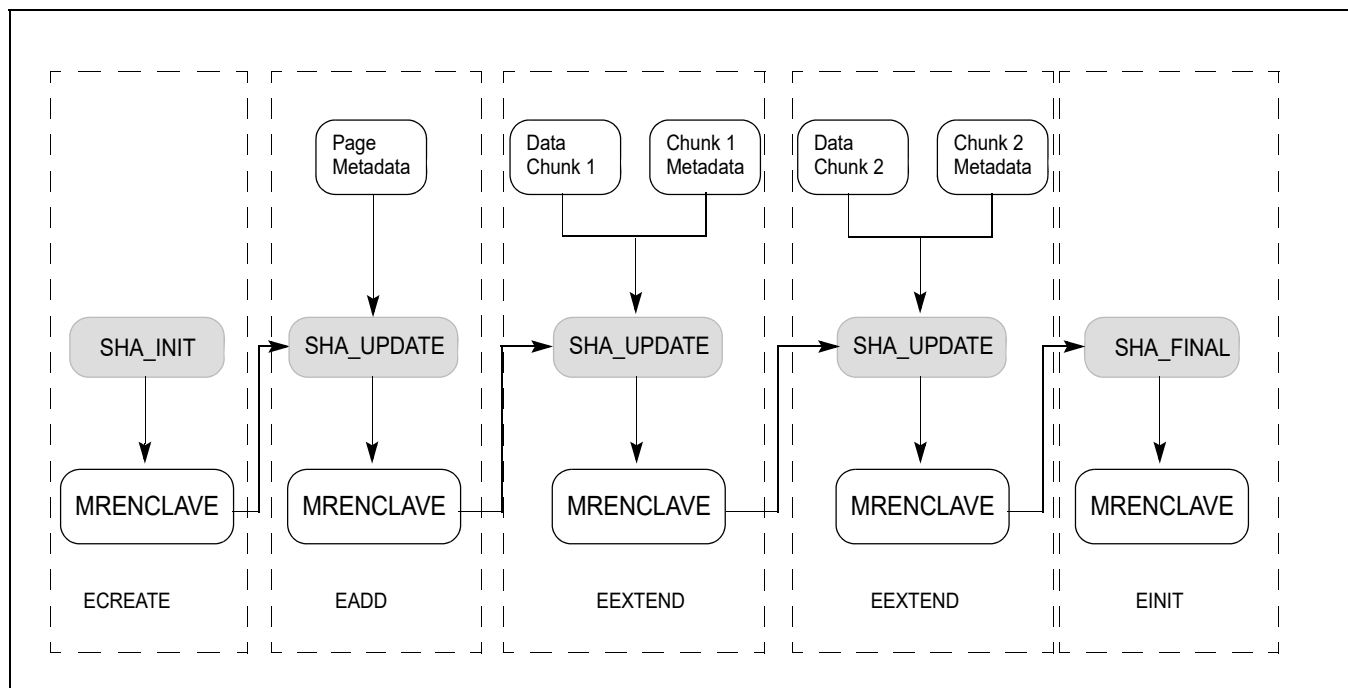


Figure 36-2. Measurement Flow of Enclave Build Process

36.4.1.2 MRSIGNER

Each enclave is signed using a 3072 bit RSA key. The signature is stored in the SIGSTRUCT. In the SIGSTRUCT, the enclave's signer also assigns a product ID (ISVPRODID) and a security version (ISVSVN) to the enclave. MRSIGNER is the SHA-256 hash of the signer's public key. For platforms that support Key Separation and Sharing (CPUID.(EAX=12H, ECX=1).EAX.KSS[7]) the SIGSTRUCT can additionally specify an 16 byte extended product ID (ISVEXTPRODID), and a 16 byte family ID (ISVFAMILYID).

In attestation, MRSIGNER can be used to allow software to approve of an enclave based on the author rather than maintaining a list of MRENCLAVES. It is used in key derivation to allow software to create a lineage of an application. By signing multiple enclaves with the same key, the enclaves will share the same keys and data. Combined with security version numbering, the author can release multiple versions of an application which can access keys for previous versions, but not future versions of that application.

36.4.1.3 CONFIGID

For platforms that support enhancements for key separation and sharing (CPUID.(EAX=12H, ECX=1).EAX.KSS[7]) when the enclave is created the platform can additionally provide 32-byte configuration identifier (CONFIGID). How this value is used is dependent on the enclave but it is intended to allow enclave creators to indicate what additional content may be accepted by the enclave post-initialization.

36.4.2 Security Version Numbers (SVN)

Intel® SGX supports a versioning system that allows the signer to identify different versions of the same software released by an author. The security version is independent of the functional version an author uses and is intended to specify security equivalence. Multiple releases with functional enhancements may all share the same SVN if they all have the same security properties or posture. Each enclave has an SVN and the underlying hardware has an SVN.

The SVNs are attested to in EREPORT and are included in the derivation of most keys, thus providing separation between data for older/newer versions.

36.4.2.1 Enclave Security Version

In the SIGSTRUCT, the MRSIGNER is associated with a 16-bit Product ID (ISVPRODID) and a 16 bit integer SVN (ISVSVN). Together they define a specific group of versions of a specific product. Most keys, including the Seal Key, can be bound to this pair.

To support upgrading from one release to another, EGETKEY will return keys corresponding to any value less than or equal to the software's ISVSVN.

36.4.2.2 Hardware Security Version

CPUSVN is a 128 bit value that reflects the microcode update version and authenticated code modules supported by the processor. Unlike ISVSVN, CPUSVN is not an integer and cannot be compared mathematically. Not all values are valid CPUSVNs.

Software must ensure that the CPUSVN provided to EGETKEY is valid. EREPORT will return the CPUSVN of the current environment. Software can execute EREPORT with TARGETINFO set to zeros to retrieve a CPUSVN from REPORTDATA. Software can access keys for a CPUSVN recorded previously, provided that each of the elements reflected in CPUSVN are the same or have been upgraded.

36.4.2.3 CONFIGID Security Version

The CONFIGID field can be used to contain the hash of a signing key for verifying the additional content. In this case, similar to the relationship between MRSIGNER and ISVSVN, CONFIGID needs a CONFIGID Security Version Number. CONFIGIDSVN can be specified at the same time as CONFIGID.

36.4.3 Keys

Intel® SGX provides software with access to keys unique to each processor and rooted in HW keys inserted into the processor during manufacturing.

Each enclave requests keys using the EGETKEY leaf function. The key is based on enclave parameters such as measurement, the enclave signing key, security attributes of the enclave, and the Hardware Security version of the processor itself. A full list of parameter options is specified in the KEYREQUEST structure, see details in Section 35.18.

By deriving keys using enclave properties, SGX guarantees that if two enclaves call EGETKEY, they will receive a unique key only accessible by the respective enclave. It also guarantees that the enclave will receive the same key on every future execution of EGETKEY. Some parameters are optional or configurable by software. For example, a Seal key can be based on the signer of the enclave, resulting in a key available to multiple enclaves signed by the same party.

The EGETKEY leaf function provides several key types. Each key is specific to the processor, CPUSVN, and the enclave that executed EGETKEY. The EGETKEY instruction definition details how each of these keys is derived, see Table 38-66. Additionally,

- **SEAL Key:** The Seal key is a general purpose key for the enclave to use to protect secrets. Typical uses of the Seal key are encrypting and calculating MAC of secrets on disk. There are 2 types of Seal Key described in Section 36.4.3.1.
- **REPORT Key:** This key is used to compute the MAC on the REPORT structure. The EREPORT leaf function is used to compute this MAC, and destination enclave uses the Report key to verify the MAC. The software usage flow is detailed in Section 36.4.3.2.
- **EINITTOKEN_KEY:** This key is used by Launch Enclaves to compute the MAC on EINITTOKENs. These tokens are then verified in the EINIT leaf function. The key is only available to enclaves with ATTRIBUTE.EINITTOKEN_KEY set to 1.
- **PROVISIONING Key and PROVISIONING SEAL Key:** These keys are used by attestation key provisioning software to prove to remote parties that the processor is genuine and identify the currently executing TCB. These keys are only available to enclaves with ATTRIBUTE.PROVISIONKEY set to 1.

36.4.3.1 Sealing Enclave Data

Enclaves can protect persistent data using Seal keys to provide encryption and/or integrity protection. EGETKEY provides two types of Seal keys specified in KEYREQUEST.KEYPOLICY field: MRENCLAVE-based key and MRSIGNER-based key.

The MRENCLAVE-based keys are available only to enclave instances sharing the same MRENCLAVE. If a new version of the enclave is released, the Seal keys will be different. Retrieving previous data requires additional software support.

The MRSIGNER-based keys are bound to the 3 tuple (MRSIGNER, ISVPRODID, ISVSVN). These keys are available to any enclave with the same MRSIGNER and ISVPRODID and an ISVSVN equal to or greater than the key in questions. This is valuable for allowing new versions of the same software to retrieve keys created before an upgrade.

For platforms that support enhancements for key separation and sharing (CPUID.(EAX=12H, ECX=1).EAX.KSS[7]) four additional key policies for seal key derivation are provided. These add the ISVEXTPRODID, ISVFAMILYID, and CONFIGID/CONFIGSVN to the key derivation. Additionally, there is a policy to remove ISVPRODID from a key derivation to create a shared between different products that share the same MRSIGNER.

36.4.3.2 Using REPORTs for Local Attestation

Intel SGX provides a means for enclaves to securely identify one another, this is referred to as “Local Attestation”. SGX provides a hardware assertion, REPORT that contains calling enclaves Attributes, Measurements and User supplied data (described in detail in Section 35.16). Figure 36-3 shows the basic flow of information.

1. The source enclave determines the identity of the target enclave to populate TARGETINFO.
2. The source enclave calls EREPORT instruction to generate a REPORT structure. The EREPORT instruction conducts the following:
 - Populates the REPORT with identify information about the calling enclave.
 - Derives the Report Key that is returned when the target enclave executes the EGETKEY. TARGETINFO provides information about the target.
 - Computes a MAC over the REPORT using derived target enclave Report Key.
3. Non-enclave software copies the REPORT from source to destination.
4. The target enclave executes the EGETKEY instruction to request its REPORT key, which is the same key used by EREPORT at the source.
5. The target enclave verifies the MAC and can then inspect the REPORT to identify the source.

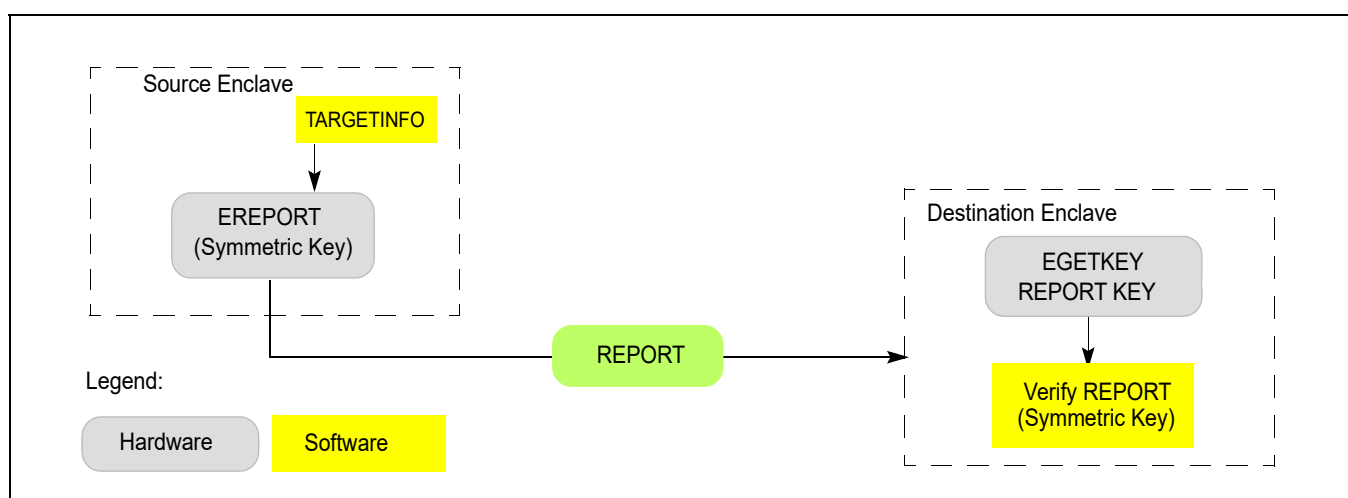


Figure 36-3. SGX Local Attestation

36.5 EPC AND MANAGEMENT OF EPC PAGES

EPC layout is implementation specific, and is enumerated through CPUID (see Table 34-7 for EPC layout). EPC is typically configured by BIOS at system boot time.

36.5.1 EPC Implementation

EPC must be properly protected against attacks. One example of EPC implementation could use a Memory Encryption Engine (MEE). An MEE provides a cost-effective mechanism of creating cryptographically protected volatile storage using platform DRAM. These units provide integrity, replay, and confidentiality protection. Details are implementation specific.

36.5.2 OS Management of EPC Pages

The EPC is a finite resource. SGX1 (i.e., CPUID.(EAX=12H, ECX=0):EAX.SGX1 = 1 but CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 0) provides the EPC manager with leaf functions to manage this resource and properly swap pages out of and into the EPC. For that, the EPC manager would need to keep track of all EPC entries, type and state, context affiliation, and SECS affiliation.

Enclave pages that are candidates for eviction should be moved to BLOCKED state using EBLOCK instruction that ensures no new cached virtual to physical address mappings can be created by attempts to reference a BLOCKED page.

Before evicting blocked pages, EPC manager should execute ETRACK leaf function on that enclave and ensure that there are no stale cached virtual to physical address mappings for the blocked pages remain on any thread on the platform.

After removing all stale translations from blocked pages, system software should use the EWB leaf function for securely evicting pages out of the EPC. EWB encrypts a page in the EPC, writes it to unprotected memory, and invalidates the copy in EPC. In addition, EWB also creates a cryptographic MAC (PCMD.MAC) of the page and stores it in unprotected memory. A page can be reloaded back to the processor only if the data and MAC match. To ensure that only the latest version of the evicted page can be loaded back, the version of the evicted page is stored securely in a Version Array (VA) in EPC.

SGX1 includes two instructions for reloading pages that have been evicted by system software: ELDU and ELDB. The difference between the two instructions is the value of the paging state at the end of the instruction. ELDU results in a page being reloaded and set to an UNBLOCKED state, while ELDB results in a page loaded to a BLOCKED state.

ELDB is intended for use by a Virtual Machine Monitor (VMM). When a VMM reloads an evicted page, it needs to restore it to the correct state of the page (BLOCKED vs. UNBLOCKED) as it existed at the time the page was evicted. Based on the state of the page at eviction, the VMM chooses either ELDB or ELDU.

36.5.2.1 Enhancement to Managing EPC Pages

On processors supporting SGX2 (i.e., CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 1), the EPC manager can manage EPC resources (while enclave is running) with more flexibility provided by the SGX2 leaf functions. The additional flexibility is described in Section 36.5.7 through Section 36.5.11.

36.5.3 Eviction of Enclave Pages

Intel SGX paging is optimized to allow the Operating System (OS) to evict multiple pages out of the EPC under a single synchronization.

The suggested flow for evicting a list of pages from the EPC is:

1. For each page to be evicted from the EPC:
 - a. Select an empty slot in a Version Array (VA) page.
 - If no empty VA page slots exist, create a new VA page using the EPA leaf function.

- b. Remove linear-address to physical-address mapping from the enclave context's mapping tables (page table and EPT tables).
 - c. Execute the EBLOCK leaf function for the target page. This sets the target page state to BLOCKED. At this point no new mappings of the page will be created. So any access which does not have the mapping cached in the TLB will generate a #PF.
2. For each enclave containing pages selected in step 1:
 - Execute an ETRACK leaf function pointing to that enclave's SECS. This initiates the tracking process that ensures that all caching of linear-address to physical-address translations for the blocked pages is cleared.
 3. For all logical processors executing in processes (OS) or guests (VMM) that contain the enclaves selected in step 1:
 - Issue an IPI (inter-processor interrupt) to those threads. This causes those logical processors to asynchronously exit any enclaves they might be in, and as a result flush cached linear-address to physical-address translations that might hold stale translations to blocked pages. There is no need for additional measures such as performing a "TLB shutdown".
 4. After enclaves exit, allow logical processors to resume normal operation, including enclave re-entry as the tracking logic keeps track of the activity.
 5. For each page to be evicted:
 - Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents, and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

At this point, system software has the only copy of each page data encrypted with its page metadata in main memory.

36.5.4 Loading an Enclave Page

To reload a previously evicted page, system software needs four elements: the VA slot used when the page was evicted, a buffer containing the encrypted page contents, a buffer containing the page metadata, and the parent SECS to associate this page with. If the VA page or the parent SECS are not already in the EPC, they must be reloaded first.

1. Execute ELDB/ELDU (depending on the desired BLOCKED state for the page), passing as parameters: the EPC page linear address, the VA slot, the encrypted page, and the page metadata.
2. Create a mapping in the enclave context's mapping tables (page tables and EPT tables) to allow the application to access that page (OS: system page table; VMM: EPT).

The ELDB/ELDU instruction marks the VA slot empty so that the page cannot be replayed at a later date.

36.5.5 Eviction of an SECS Page

The eviction of an SECS page is similar to the eviction of an enclave page. The only difference is that an SECS page cannot be evicted until all other pages belonging to the enclave have been evicted. Since all other pages have been evicted, there will be no threads executing inside the enclave and tracking with ETRACK isn't necessary. When reloading an enclave, the SECS page must be reloaded before all other constituent pages.

1. Ensure all pages are evicted from enclave.
2. Select an empty slot in a Version Array page.
 - If no VA page exists with an empty slot, create a new one using the EPA function leaf.
3. Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

36.5.6 Eviction of a Version Array Page

VA pages do not belong to any enclave and tracking with ETRACK isn't necessary. When evicting the VA page, a slot in a different VA page must be specified in order to provide versioning of the evicted VA page.

1. Select a slot in a Version Array page other than the page being evicted.
 - If no VA page exists with an empty slot, create a new one using the EPA leaf function.
2. Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents, and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

36.5.7 Allocating a Regular Page

On processors that support SGX2, allocating a new page to an already initialized enclave is accomplished by invoking the EAUG leaf function. Typically, the enclave requests that the OS allocates a new page at a particular location within the enclave's address space. Once allocated, the page remains in a pending state until the enclave executes the corresponding EACCEPT leaf function to accept the new page into the enclave. Page allocation operations may be batched to improve efficiency.

The typical process for allocating a regular page is as follows:

1. Enclave requests additional memory from OS when the current allocation becomes insufficient.
2. The OS invokes the EAUG leaf function to add a new memory page to the enclave.
 - a. EAUG may only be called on a free EPC page.
 - b. Successful completion of the EAUG instruction places the target page in the VALID and PENDING state.
 - c. All dynamically created pages have the type PT_REG and content of all zeros.
3. The OS maps the page in the enclave context's mapping tables.
4. The enclave issues an EACCEPT instruction, which verifies the page's attributes and clears the PENDING state. At that point the page becomes accessible for normal enclave use.

36.5.8 Allocating a TCS Page

On processors that support SGX2, allocating a new TCS page to an already initialized enclave is a two-step process. First the OS allocates a regular page with a call to EAUG. This page must then be accepted and initialized by the enclave to which it belongs. Once the page has been initialized with appropriate values for a TCS page, the enclave requests the OS to change the page's type to PT_TCS. This change must also be accepted. As with allocating a regular page, TCS allocation operations may be batched.

A typical process for allocating a TCS page is as follows:

1. Enclave requests an additional page from the OS.
2. The OS invokes EAUG to add a new regular memory page to the enclave.
 - a. EAUG may only be called on a free EPC page.
 - b. Successful completion of the EAUG instruction places the target page in the VALID and PENDING state.
3. The OS maps the page in the enclave context's mapping tables.
4. The enclave issues an EACCEPT instruction, at which point the page becomes accessible for normal enclave use.
5. The enclave initializes the contents of the new page.
6. The enclave requests that the OS convert the page from type PT_REG to PT_TCS.
7. OS issues an EMODT instruction on the page.
 - a. The parameters to EMODT indicate that the regular page should be converted into a TCS.

- b. EMODT forces all access rights to a page to be removed because TCS pages may not be accessed by enclave code.
8. The enclave issues an EACCEPT instruction to confirm the requested modification.

36.5.9 Trimming a Page

On processors that support SGX2, Intel SGX supports the trimming of an enclave page as a special case of EMODT. Trimming allows an enclave to actively participate in the process of removing a page from the enclave (deallocation) by splitting the process into first removing it from the enclave's access and then removing it from the EPC using the EREMOVE leaf function. The page type PT_TRIM indicates that a page has been trimmed from the enclave's address space and that the page is no longer accessible to enclave software. Modifications to a page in the PT_TRIM state are not permitted; the page must be removed and then reallocated by the OS before the enclave may use the page again. Page deallocation operations may be batched to improve efficiency.

The typical process for trimming a page from an enclave is as follows:

1. Enclave signals OS that a particular page is no longer in use.
2. OS invokes the EMODT leaf function on the page, requesting that the page's type be changed to PT_TRIM.
 - a. SECS and VA pages cannot be trimmed in this way, so the initial type of the page must be PT_REG or PT_TCS.
 - b. EMODT may only be called on valid enclave pages.
3. OS invokes the ETRACK leaf function on the enclave containing the page to track removal the TLB addresses from all the processors.
4. Issue an IPI (inter-processor interrupt) to flush the stale linear-address to physical-address translations for all logical processors executing in processes that contain the enclave.
5. Enclave issues an EACCEPT leaf function.
6. The OS may now permanently remove the page from the EPC (by issuing EREMOVE).

36.5.10 Restricting the EPCM Permissions of a Page

On processors that support SGX2, restricting the EPCM permissions associated with an enclave page is accomplished using the EMODPR leaf function. This operation requires the cooperation of the OS to flush stale entries to the page and to update the page-table permissions of the page to match. Permissions restriction operations may be batched.

The typical process for restricting the permissions of an enclave page is as follows:

1. Enclave requests that the OS to restrict the permissions of an EPC page.
2. OS performs permission restriction, flushing cached linear-address to physical-address translations, and page-table modifications.
 - a. Invokes the EMODPR leaf function to restrict permissions (EMODPR may only be called on VALID pages).
 - b. Invokes the ETRACK leaf function on the enclave containing the page to track removal of the TLB addresses from all the processor.
 - c. Issue an IPI (inter-processor interrupt) to flush the stale linear-address to physical-address translations for all logical processors executing in processes that contain the enclave.
 - d. Sends IPIs to trigger enclave thread exit and TLB shutdown.
 - e. OS informs the Enclave that all logical processors should now see the new restricted permissions.
3. Enclave invokes the EACCEPT leaf function.
 - a. Enclave may access the page throughout the entire process.
 - b. Successful call to EACCEPT guarantees that no stale cached linear-address to physical-address translations are present.

36.5.11 Extending the EPCM Permissions of a Page

On processors that support SGX2, extending the EPCM permissions associated with an enclave page is accomplished directly by the enclave using the EMODPE leaf function. After performing the EPCM permission extension, the enclave requests the OS to update the page table permissions to match the extended permission. Security wise, permission extension does not require enclave threads to leave the enclave as TLBs with stale references to the more restrictive permissions will be flushed on demand, but to allow forward progress, an OS needs to be aware that an application might signal a page fault.

The typical process for extending the permissions of an enclave page is as follows:

1. Enclave invokes EMODPE to extend the EPCM permissions associated with an EPC page (EMODPE may only be called on VALID pages).
2. Enclave requests that OS update the page tables to match the new EPCM permissions.
3. Enclave code resumes.
 - a. If cached linear-address to physical-address translations are present to the more restrictive permissions, the enclave thread will page fault. The SGX2-aware OS will see that the page tables permit the access and resume the thread, which can now successfully access the page because exiting cleared the TLB.
 - b. If cached linear-address to physical-address translations are not present, access to the page with the new permissions will succeed without an enclave exit.

36.5.12 VMM Oversubscription of EPC

On processors supporting oversubscription enhancements (i.e., CPUID.(EAX=12H, ECX=0):EAX[5]=1 & EAX[6] = 1) a Virtual Machine Monitor or other executive can more efficiently manage the EPC space available on the platform between virtualized entities. A typical process for using these instructions to support oversubscribing the physical EPC space on the platform is as follows:

1. VMM creates data structures for SECS tracking including a count of child pages.
2. VMM selects possible EPC victim pages.
3. VMM ages the victim pages. Some of the selected pages will be accessed by the guest. In this case the VMM will remove these pages from the victim pool and return them to the guest.
4. VMM makes remaining pages not present in EPT. It then issues IPI on each page to remove TLB mappings.
5. For every EPC victim page the VMM obtains the victim's SECS page info using ERDINFO.
 - a. ENCLAVECONTEXT field in RDINFO structure will indicate the location of SECS, and the PAGE_TYPE field will indicate the page type.
 - b. Child pages of SECS can be evicted.
 - c. SECS pages may be evicted if the child count is zero.
 - d. Some pages may be returned to active state depending on such things as page type or child count.
6. VMM increments its evicted page count for the SECS of each page (stored in the data structure created in 1).
7. If this is the first evicted page of that SECS, set Marker on SECS of the victim page (EINCVIRTCHILD). This locks the SECS in the guest. The guest cannot page out the SECS.
8. EBLOCK, ETRACK, EWB eviction sequence is executed for page.
9. After loading an SECS page back in, the VMM will set the correct ENCLAVECONTEXT for the guest using ESETCONTEXT instruction.

36.6 CHANGES TO INSTRUCTION BEHAVIOR INSIDE AN ENCLAVE

This section covers instructions whose behavior changes when executed in enclave mode.

36.6.1 Illegal Instructions

The instructions listed in Table 36-1 are ring 3 instructions which become illegal when executed inside an enclave. Executing these instructions inside an enclave will generate an exception.

The first row of Table 36-1 enumerates instructions that may cause a VM exit for VMM emulation. Since a VMM cannot emulate enclave execution, execution of any of these instructions inside an enclave results in an invalid-opcode exception (#UD) and no VM exit.

The second row of Table 36-1 enumerates I/O instructions that may cause a fault or a VM exit for emulation. Again, enclave execution cannot be emulated, so execution of any of these instructions inside an enclave results in #UD.

The third row of Table 36-1 enumerates instructions that load descriptors from the GDT or the LDT or that change privilege level. The former class is disallowed because enclave software should not depend on the contents of the descriptor tables and the latter because enclave execution must be entirely with CPL = 3. Again, execution of any of these instructions inside an enclave results in #UD.

The fourth row of Table 36-1 enumerates instructions that provide access to kernel information from user mode and can be used to aid kernel exploits from within enclave. Execution of any of these instructions inside an enclave results in #UD.

Table 36-1. Illegal Instructions Inside an Enclave

Instructions	Result	Comment
CPUID, GETSEC, RDPMS, SGDT, SIDT, SLDT, STR, VMCALL, VMFUNC	#UD	Might cause VM exit.
IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD	#UD	I/O fault may not safely recover. May require emulation.
Far call, Far jump, Far Ret, INT <i>n</i> /INTO, IRET, LDS/LES/LFS/LGS/LSS, MOV to DS/ES/SS/FS/GS, POP DS/ES/SS/FS/GS, SYSCALL, SYSENTER	#UD	Access segment register could change privilege level.
SMSW	#UD	Might provide access to kernel information.
ENCLU[EENTER], ENCLU[ERESUME]	#GP	Cannot enter an enclave from within an enclave.

RDTSC and RDTSCP are legal inside an enclave for processors that support SGX2 (subject to the value of CR4.TSD). For processors which support SGX1 but not SGX2, RDTSC and RDTSCP will cause #UD.

RDTSC and RDTSCP instructions may cause a VM exit when inside an enclave.

Software developers must take into account that the RDTSC/RDTSCP results are not immune to influences by other software, e.g., the TSC can be manipulated by software outside the enclave.

36.6.2 RDRAND and RDSEED Instructions

These instructions may cause a VM exit if the “RDRAND exiting” VM-execution control is 1. Unlike other instructions that can cause VM exits, these instructions are legal inside an enclave. As noted in Section 28.1 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C, any VM exit originating on an instruction boundary inside an enclave sets bit 27 of the exit-reason field of the VMCS. If a VMM receives a VM exit due to an attempt to execute either of these instructions determines (by that bit) that the execution was inside an enclave, it can do either of two things. It can clear the “RDRAND exiting” VM-execution control and execute VMRESUME; this will result in the enclave executing RDRAND or RDSEED again, and this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

NOTE

It is expected that VMMs that virtualize Intel SGX will not set “RDRAND exiting” to 1.

36.6.3 PAUSE Instruction

The PAUSE instruction may cause a VM exit from an enclave if the “PAUSE exiting” VM-execution control is 1. Unlike other instructions that can cause VM exits, the PAUSE instruction is legal inside an enclave. If a VMM receives a VM exit due to the 1-setting of “PAUSE exiting”, it can do either of two things. It can clear the “PAUSE exiting” VM-

ENCLAVE OPERATION

execution control and execute VMRESUME; this will result in the enclave executing PAUSE again, but this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

The PAUSE instruction may also cause a VM exit outside of an enclave if the "PAUSE-loop exiting" VM-execution control is 1, but as the "PAUSE-loop exiting" control is ignored at CPL > 0 (see Section 26.1.3), VM exit from an enclave due to the 1-setting of "PAUSE-LOOP exiting" will never occur.

NOTE

It is expected that VMMs that virtualize Intel SGX will not set "PAUSE exiting" to 1.

36.6.4 Executions of INT1 and INT3 Inside an Enclave

The INT1 and INT3 instructions are legal inside an enclave, however, their behavior inside an enclave differs from that outside an enclave. See Section 40.4.1 for details.

36.6.5 INVD Handling when Enclaves Are Enabled

Once processor reserved memory protections are activated (see Section 36.5), any execution of INVD will result in a #GP(0).

19. Updates to Chapter 38, Volume 3D

Change bars and green text show changes to Chapter 38 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Updates throughout the chapter for the AEX Notify feature.

CHAPTER 38

INTEL® SGX INSTRUCTION REFERENCES

This chapter describes the supervisor and user level instructions provided by Intel® Software Guard Extensions (Intel® SGX). In general, various functionality is encoded as leaf functions within the ENCLS (supervisor), ENCLU (user), and the ENCLV (virtualization operation) instruction mnemonics. Different leaf functions are encoded by specifying an input value in the EAX register of the respective instruction mnemonic.

38.1 INTEL® SGX INSTRUCTION SYNTAX AND OPERATION

ENCLS, ENCLU, and ENCLV instruction mnemonics for all leaf functions are covered in this section.

For all instructions, the value of CS.D is ignored; addresses and operands are 64 bits in 64-bit mode and are otherwise 32 bits. Aside from EAX specifying the leaf number as input, each instruction leaf may require all or some subset of the RBX/RCX/RDX as input parameters. Some leaf functions may return data or status information in one or more of the general purpose registers.

38.1.1 ENCLS Register Usage Summary

Table 38-1 summarizes the implicit register usage of supervisor mode enclave instructions.

Table 38-1. Register Usage of Privileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
ECREATE	00H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EADD	01H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EINIT	02H (In)	SIGSTRUCT (In, EA)	SECS (In, EA)	EINITTOKEN (In, EA)
EREMOVE	03H (In)		EPCPAGE (In, EA)	
EDBGGRD	04H (In)	Result Data (Out)	EPCPAGE (In, EA)	
EDBGWR	05H (In)	Source Data (In)	EPCPAGE (In, EA)	
EEXTEND	06H (In)	SECS (In, EA)	EPCPAGE (In, EA)	
ELDB	07H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDU	08H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
EBLOCK	09H (In)		EPCPAGE (In, EA)	
EPA	0AH (In)	PT_VA (In)	EPCPAGE (In, EA)	
EWB	0BH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ETRACK	0CH (In)		EPCPAGE (In, EA)	
EAUG	0DH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EMODPR	0EH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODT	0FH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
ERDINFO	010H (In)	RDINFO (In, EA*)	EPCPAGE (In, EA)	
ETRACKC	011H (In)		EPCPAGE (In, EA)	
ELDBC	012H (In)	PAGEINFO (In, EA*)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDUC	013H (In)	PAGEINFO (In, EA*)	EPCPAGE (In, EA)	VERSION (In, EA)

EA: Effective Address

38.1.2 ENCLU Register Usage Summary

Table 38-2 summarizes the implicit register usage of user mode enclave instructions.

Table 38-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDY
EREPOR	00H (In)	TARGETINFO (In, EA)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)
EGETKEY	01H (In)	KEYREQUEST (In, EA)	KEY (In, EA)	
EENTER	02H (In)	TCS (In, EA)	AEP (In, EA)	
	RBX.CSSA (Out)		Return (Out, EA)	
ERESUME	03H (In)	TCS (In, EA)	AEP (In, EA)	
EEXIT	04H (In)	Target (In, EA)	Current AEP (Out)	
EACCEPT	05H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODPE	06H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EACCEPTCOPY	07H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	EPCPAGE (In, EA)
EDECCSSA	09H (In)			

EA: Effective Address

38.1.3 ENCLV Register Usage Summary

Table 38-3 summarizes the implicit register usage of virtualization operation enclave instructions.

Table 38-3. Register Usage of Virtualization Operation Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDY
EDEVIRTCCHILD	00H (In)	EPCPAGE (In, EA)	SECS (In, EA)	
EINCVIRTCCHILD	01H (In)	EPCPAGE (In, EA)	SECS (In, EA)	
ESETCONTEXT	02H (In)		EPCPAGE (In, EA)	Context Value (In, EA)

EA: Effective Address

38.1.4 Information and Error Codes

Information and error codes are reported by various instruction leaf functions to show an abnormal termination of the instruction or provide information which may be useful to the developer. Table 38-4 shows the various codes and the instruction which generated the code. Details of the meaning of the code is provided in the individual instruction.

Table 38-4. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
No Error	0	
SGX_INVALID_SIG_STRUCT	1	EINIT
SGX_INVALID_ATTRIBUTE	2	EINIT, EGETKEY
SGX_BLKSTATE	3	EBLOCK
SGX_INVALID_MEASUREMENT	4	EINIT
SGX_NOTBLOCKABLE	5	EBLOCK
SGX_PG_INVLD	6	EBLOCK, ERDINFO, ETRACKC

Table 38-4. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
SGX_EPC_PAGE_CONFLICT	7	EBLOCK, EMODPR, EMODT, ERDINFO, EDECVIRTCHILD, EINCVIRTCHILD, ELDBC, ELDUC, ESETCONTEXT, ETRACKC
SGX_INVALID_SIGNATURE	8	EINIT
SGX_MAC_COMPARE_FAIL	9	ELDB, ELDU, ELDBC, ELDUC
SGX_PAGE_NOT_BLOCKED	10	EWB
SGX_NOT_TRACKED	11	EWB, EACCEPT
SGX_VA_SLOT_OCCUPIED	12	EWB
SGX_CHILD_PRESENT	13	EWB, EREMOVE
SGX_ENCLAVE_ACT	14	EREMOVE
SGX_ENTRYEPOCH_LOCKED	15	EBLOCK
SGX_INVALID_EINITTOKEN	16	EINIT
SGX_PREV_TRK_INCMPL	17	ETRACK, ETRACKC
SGX_PG_IS_SECS	18	EBLOCK
SGX_PAGE_ATTRIBUTES_MISMATCH	19	EACCEPT, EACCEPTCOPY
SGX_PAGE_NOT_MODIFIABLE	20	EMODPR, EMODT
SGX_PAGE_NOT_DEBUGGABLE	21	EDBGRD, EDBGWR
SGX_INVALID_COUNTER	25	EDECVIRTCHILD
SGX_PG_NONEPC	26	ERDINFO
SGX_TRACK_NOT_REQUIRED	27	ETRACKC
SGX_INVALID_CPUSVN	32	EINIT, EGETKEY
SGX_INVALID_ISVSVN	64	EGETKEY
SGX_UNMASKED_EVENT	128	EINIT
SGX_INVALID_KEYNAME	256	EGETKEY

38.1.5 Internal CREGs

The CREGs as shown in Table 5-4 are hardware specific registers used in this document to indicate values kept by the processor. These values are used while executing in enclave mode or while executing an Intel SGX instruction. These registers are not software visible and are implementation specific. The values in Table 38-5 appear at various places in the pseudo-code of this document. They are used to enhance understanding of the operations.

Table 38-5. List of Internal CREG

Name	Size (Bits)	Scope
CR_ENCLAVE_MODE	1	LP
CR_DBGOPTIN	1	LP
CR_TCS_LA	64	LP
CR_TCS_PA	64	LP
CR_ACTIVE_SECS	64	LP
CR_ELRANGE	128	LP
CR_SAVE_TF	1	LP
CR_SAVE_FS	64	LP
CR_GPR_PA	64	LP
CR_XSAVE_PAGE_n	64	LP

Table 38-5. List of Internal CREG

Name	Size (Bits)	Scope
CR_SAVE_DR7	64	LP
CR_SAVE_PERF_GLOBAL_CTRL	64	LP
CR_SAVE_DEBUGCTL	64	LP
CR_SAVE_PEBS_ENABLE	64	LP
CR_CPUSVN	128	PACKAGE
CR_SGXOWNEREPOCH	128	PACKAGE
CR_SAVE_XCRO	64	LP
CR_SGX_ATTRIBUTES_MASK	128	LP
CR_PAGING_VERSION	64	PACKAGE
CR_VERSION_THRESHOLD	64	PACKAGE
CR_NEXT_EID	64	PACKAGE
CR_BASE_PK	128	PACKAGE
CR_SEAL_FUSES	128	PACKAGE
CR_CET_SAVE_AREA_PA	64	LP
CR_ENCLAVE_SS_TOKEN_PA	64	LP
CR_SAVE_IA32_U_CET	64	LP
CR_SAVE_SSP	64	LP

38.1.6 Concurrent Operation Restrictions

Under certain conditions, Intel SGX disallows certain leaf functions from operating concurrently. Listed below are some examples of concurrency that are not allowed.

- For example, Intel SGX disallows the following leaves to concurrently operate on the same EPC page.
 - ECREATE, EADD, and EREMOVE are not allowed to operate on the same EPC page concurrently with themselves.
 - EADD, EEXTEND, and EINIT leaves are not allowed to operate on the same SECS concurrently.
- Intel SGX disallows the EREMOVE leaf from removing pages from an enclave that is in use.
- Intel SGX disallows entry (EENTER and ERESUME) to an enclave while a page from that enclave is being removed.

When disallowed operation is detected, a leaf function may do one of the following:

- Return an SGX_EPC_PAGE_CONFLICT error code in RAX.
- Cause a #GP(0) exception.

To prevent such exceptions, software must serialize leaf functions or prevent these leaf functions from accessing the same EPC page.

38.1.6.1 Concurrency Tables of Intel® SGX Instructions

The tables below detail the concurrent operation restrictions of all SGX leaf functions. For each leaf function, the table has a separate line for each of the EPC pages the leaf function accesses.

For each such EPC page, the base concurrency requirements are detailed as follows:

- **Exclusive Access** means that no other leaf function that requires either shared or exclusive access to the same EPC page may be executed concurrently. For example, EADD requires an exclusive access to the target page it accesses.

- **Shared Access** means that no other leaf function that requires an exclusive access to the same EPC page may be executed concurrently. Other leaf functions that require shared access may run concurrently. For example, EADD requires a shared access to the SECS page it accesses.
- **Concurrent Access** means that any other leaf function that requires any access to the same EPC page may be executed concurrently. For example, EGETKEY has no concurrency requirements for the KEYREQUEST page.

In addition to the base concurrency requirements, additional concurrency requirements are listed, which apply only to specific sets of leaf functions. For example, there are additional requirements that apply for EADD, EXTEND, and EINIT. EADD and EEXTEND can't execute concurrently on the same SECS page.

The tables also detail the leaf function's behavior when a conflict happens, i.e., a concurrency requirement is not met. In this case, the leaf function may return an SGX_EPC_PAGE_CONFLICT error code in RAX, or it may cause an exception. In addition, the tables detail those conflicts where a VM Exit may be triggered, and list the Exit Qualification code that is provided in such cases.

Table 38-6. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions		
			Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPT	Target	[DS:RCX]	Shared	#GP	
	SECINFO	[DS:RBX]	Concurrent		
EACCEPTCOPY	Target	[DS:RCX]	Concurrent		
	Source	[DS:RDX]	Concurrent		
	SECINFO	[DS:RBX]	Concurrent		
EADD	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS	[DS:RBX]PAGEINFO. SECS	Shared	#GP	
EAUG	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS	[DS:RBX]PAGEINFO. SECS	Shared	#GP	
EBLOCK	Target	[DS:RCX]	Shared	SGX_EPC_PAGE _CONFLICT	
ECREATE	SECS	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
EDBGGRD	Target	[DS:RCX]	Shared	#GP	
EDBGWR	Target	[DS:RCX]	Shared	#GP	
EDECVIRTCHILD	Target	[DS:RBX]	Shared	SGX_EPC_PAGE _CONFLICT	
	SECS	[DS:RCX]	Concurrent		
EENTERTCS	SECS	[DS:RBX]	Shared	#GP	
EEXIT			Concurrent		
EEXTEND	Target	[DS:RCX]	Shared	#GP	
	SECS	[DS:RBX]	Concurrent		
EGETKEY	KEYREQUEST	[DS:RBX]	Concurrent		
	OUTPUTDATA	[DS:RCX]	Concurrent		
EINCVIRTCHILD	Target	[DS:RBX]	Shared	SGX_EPC_PAGE _CONFLICT	
	SECS	[DS:RCX]	Concurrent		
EINIT	SECS	[DS:RCX]	Shared	#GP	

Table 38-6. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions		
			Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ELDB/ELDU	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA	[DS:RDX]	Shared	#GP	
	SECS	[DS:RBX]PAGEINFO. SECS	Shared	#GP	
EDLBC/ELDUC	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
	VA	[DS:RDX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	[DS:RBX]PAGEINFO. SECS	Shared	SGX_EPC_PAGE_CONFLICT	
EMODPE	Target	[DS:RCX]	Concurrent		
	SECINFO	[DS:RBX]	Concurrent		
EMODPR	Target	[DS:RCX]	Shared	#GP	
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
EPA	VA	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
ERDINFO	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
EREMOVE	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
EReport	TARGETINFO	[DS:RBX]	Concurrent		
	REPORTDATA	[DS:RCX]	Concurrent		
	OUTPUTDATA	[DS:RDX]	Concurrent		
ERESUME	TCS	[DS:RBX]	Shared	#GP	
ESETCONTEXT	SECS	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
ETRACK	SECS	[DS:RCX]	Shared	#GP	
ETRACKC	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	Implicit	Concurrent		
EWB	Source	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA	[DS:RDX]	Shared	#GP	

Table 38-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPT	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	

Table 38-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	
EPA	VA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ERDINFO	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREMOVE	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREPORT	TARGETINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
ERESUME	TCS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
ESETCONTEXT	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ETRACK	SECS	[DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT ¹
ETRACKC	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	Implicit	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT ¹
EWB	Source	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	

NOTES:

1. SGX_CONFLICT VM Exit Qualification =TRACKING_RESOURCE_CONFLICT.

38.2 INTEL® SGX INSTRUCTION REFERENCE

ENCLS—Execute an Enclave System Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 CF ENCLS	Z0	V/V	NA	This instruction is used to execute privileged Intel SGX leaf functions that are used for managing and debugging the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
Z0	NA	NA	NA	See Section 38.3

Description

The ENCLS instruction invokes the specified privileged Intel SGX leaf function for managing and debugging enclaves. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLS instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute the instruction when CPL > 0 results in #UD. The instruction produces a general-protection exception (#GP) if CR0.PG = 0 or if an attempt is made to invoke an undefined leaf function.

In VMX non-root operation, execution of ENCLS may cause a VM exit if the “enable ENCLS exiting” VM-execution control is 1. In this case, execution of individual leaf functions of ENCLS is governed by the ENCLS-exiting bitmap field in the VMCS. Each bit in that field corresponds to the index of an ENCLS leaf function (as provided in EAX).

Software in VMX root operation can thus intercept the invocation of various ENCLS leaf functions in VMX non-root operation by setting the “enable ENCLS exiting” VM-execution control and setting the corresponding bits in the ENCLS-exiting bitmap.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 || CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 || CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, and is the REX prefix in 64-bit mode.

Operation

IF TSX_ACTIVE

THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE = 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0

THEN #UD; FI;

IF (CPL > 0)

THEN #UD; FI;

IF in VMX non-root operation and the “enable ENCLS exiting” VM-execution control is 1

THEN

IF EAX < 63 and ENCLS_exiting_bitmap[EAX] = 1 or EAX > 62 and ENCLS_exiting_bitmap[63] = 1

THEN VM exit;

FI;

FI;

IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0

THEN #GP(0); FI;

IF (EAX is an invalid leaf number)

THEN #GP(0); FI;

IF CR0.PG = 0
THEN #GP(0); FI;

(* DS must not be an expanded down segment *)
IF not in 64-bit mode and DS.Type is expand-down data
THEN #GP(0); FI;

Jump to leaf specific flow

Flags Affected

See individual leaf functions

Protected Mode Exceptions

- #UD
If any of the LOCK/66H/REP/VEX prefixes are used.
If current privilege level is not 0.
If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
If logical processor is in SMM.
- #GP(0)
If IA32_FEATURE_CONTROL.LOCK = 0.
If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
If input value in EAX encodes an unsupported leaf.
If data segment expand down.
If CR0.PG=0.

Real-Address Mode Exceptions

- #UD
ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

- #UD
ENCLS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

- #UD
If any of the LOCK/66H/REP/VEX prefixes are used.
If current privilege level is not 0.
If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
If logical processor is in SMM.
- #GP(0)
If IA32_FEATURE_CONTROL.LOCK = 0.
If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
If input value in EAX encodes an unsupported leaf.

ENCLU—Execute an Enclave User Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 D7 ENCLU	Z0	V/V	NA	This instruction is used to execute non-privileged Intel SGX leaf functions.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
Z0	NA	NA	NA	See Section 38.4

Description

The ENCLU instruction invokes the specified non-privileged Intel SGX leaf functions. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLU instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute this instruction when CPL < 3 results in #UD. The instruction produces a general-protection exception (#GP) if either CR0.PG or CR0.NE is 0, or if an attempt is made to invoke an undefined leaf function. The ENCLU instruction produces a device not available exception (#NM) if CR0.TS = 1.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 or CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 and CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, as is the REX prefix in 64-bit mode.

Operation

```
IN_64BIT_MODE := 0;
```

```
IF TSX_ACTIVE
```

```
    THEN GOTO TSX_ABORT_PROCESSING; FI;
```

(* If enclosing app has CET indirect branch tracking enabled then if it is not ERESUME leaf cause a #CP fault *)

(* If the ERESUME is not successful it will leave tracker in WAIT_FOR_ENDBRANCH *)

```
TRACKER = (CPL == 3) ? IA32_U_CET.TRACKER : IA32_S_CET.TRACKER
```

```
IF EndbranchEnabledAndNotSuppressed(CPL) and TRACKER = WAIT_FOR_ENDBRANCH and  
(EAX != ERESUME or CR0.TS or (in SMM) or (CPUID.SGX_LEAF.0:EAX.SE1 = 0) or (CPL < 3))
```

```
    THEN
```

```
        Handle CET State machine violation
```

(* see Section 17.3.6, “Legacy Compatibility Treatment” in the
Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1. *)

```
    FI;
```

```
IF CR0.PE= 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0
```

```
    THEN #UD; FI;
```

```
IF CR0.TS = 1
```

```
    THEN #NM; FI;
```

```
IF CPL < 3
```

```
    THEN #UD; FI;
```

```
IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
```

```
    THEN #GP(0); FI;
```


IF EAX is invalid leaf number
THEN #GP(0); FI;

IF CR0.PG = 0 or CR0.NE = 0
THEN #GP(0); FI;

IN_64BIT_MODE := IA32_EFER.LMA AND CS.L ? 1 : 0;
(* Check not in 16-bit mode and DS is not a 16-bit segment *)
IF not in 64-bit mode and CS.D = 0
THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 1 and (EAX = 2 or EAX = 3) (* EENTER or ERESUME *)
THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 0 and (EAX = 0 or EAX = 1 or EAX = 4 or EAX = 5 or EAX = 6 or EAX = 7 or EAX = 9)
(* EREPORT, EGETKEY, EEXIT, EACCEPT, EMODPE, EACCEPTCOPY, or EDECCSSA *)
THEN #GP(0); FI;

Jump to leaf specific flow

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	<p>If any of the LOCK/66H/REP/VEX prefixes are used. If current privilege level is not 3. If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0. If logical processor is in SMM.</p>
#GP(0)	<p>If IA32_FEATURE_CONTROL.LOCK = 0. If IA32_FEATURE_CONTROL.SGX_ENABLE = 0. If input value in EAX encodes an unsupported leaf. If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1. If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0. If operating in 16-bit mode. If data segment is in 16-bit mode. If CR0.PG = 0 or CR0.NE = 0.</p>
#NM	<p>If CR0.TS = 1.</p>

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.
-----	---------------------------------------

Virtual-8086 Mode Exceptions

#UD	ENCLS is not recognized in virtual-8086 mode.
-----	---

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used. If current privilege level is not 3. If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0. If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0. If IA32_FEATURE_CONTROL.SGX_ENABLE = 0. If input value in EAX encodes an unsupported leaf. If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1. If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0. If CR0.NE = 0.
#NM	If CR0.TS = 1.

ENCLV—Execute an Enclave VMM Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP OF 01 C0 ENCLV	Z0	V/V	NA	This instruction is used to execute privileged SGX leaf functions that are reserved for VMM use. They are used for managing the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
Z0	NA	NA	NA	See Section 38.3

Description

The ENCLV instruction invokes the virtualization SGX leaf functions for managing enclaves in a virtualized environment. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In non 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLV instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, if it is executed in system-management mode (SMM), or not in VMX operation. Additionally, any attempt to execute the instruction when CPL > 0 results in #UD. The instruction produces a general-protection exception (#GP) if CR0.PG = 0 or if an attempt is made to invoke an undefined leaf function.

Software in VMX root mode of operation can enable execution of the ENCLV instruction in VMX non-root mode by setting enable ENCLV execution control in the VMCS. If enable ENCLV execution control in the VMCS is clear, execution of the ENCLV instruction in VMX non-root mode results in #UD.

When execution of ENCLV instruction in VMX non-root mode is enabled, software in VMX root operation can intercept the invocation of various ENCLV leaf functions in VMX non-root operation by setting the corresponding bits in the ENCLV-exiting bitmap.

Addresses and operands are 32 bits in 32-bit mode (IA32_EFER.LMA == 0 || CS.L == 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA == 1 && CS.L == 1). CS.D value has no impact on address calculation.

Segment override prefixes and address-size override prefixes are ignored, as is the REX prefix in 64-bit mode.

Operation

IF TSX_ACTIVE

THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE = 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.OSS = 0

THEN #UD; FI;

IF not in VMX Operation or (IA32_EFER.LMA = 1 and CS.L = 0)

THEN #UD; FI;

IF (CPL > 0)

THEN #UD; FI;

IF in VMX non-root operation

IF “enable ENCLV exiting” VM-execution control is 1

THEN

IF EAX < 63 and ENCLV_exiting_bitmap[EAX] = 1 or EAX > 62 and ENCLV_exiting_bitmap[63] = 1

THEN VM exit;

FI;

ELSE

#UD; FI;

FI;

IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
THEN #GP(0); FI;

IF (EAX is an invalid leaf number)
THEN #GP(0); FI;

IF CR0.PG = 0
THEN #GP(0); FI;

(* DS must not be an expanded down segment *)
IF not in 64-bit mode and DS.Type is expand-down data
THEN #GP(0); FI;

Jump to leaf specific flow

Flags Affected

See individual leaf functions.

Protected Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used. If current privilege level is not 0. If CPUID.(EAX=12H,ECX=0):EAX.OSS [bit 5] = 0. If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0. If IA32_FEATURE_CONTROL.SGX_ENABLE = 0. If input value in EAX encodes an unsupported leaf. If data segment expand down. If CR0.PG=0.

Real-Address Mode Exceptions

#UD	ENCLV is not recognized in real mode.
-----	---------------------------------------

Virtual-8086 Mode Exceptions

#UD	ENCLV is not recognized in virtual-8086 mode.
-----	---

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used. If current privilege level is not 0. If CPUID.(EAX=12H,ECX=0):EAX.OSS [bit 5] = 0. If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0. If IA32_FEATURE_CONTROL.SGX_ENABLE = 0. If input value in EAX encodes an unsupported leaf.

38.3 INTEL® SGX SYSTEM LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLS instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EADD—Add a Page to an Uninitialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLS[EADD]	IR	V/V	SGX1	This leaf function adds a page to an uninitialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EADD (In)	Address of a PAGEINFO (In)	Address of the destination EPC page (In)

Description

This leaf function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. As part of the association, the enclave offset and the security attributes are measured and extended into the SECS.MRENCLAVE. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of EADD leaf function.

EADD Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EADD Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	If security attributes specifies a TCS and the source page specifies unsupported TCS values or fields.
The SECS has been initialized.	The specified enclave offset is outside of the enclave address space.

Concurrency Restrictions

Table 38-8. Base Concurrency Restrictions of EADD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EADD	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	

Table 38-9. Additional Concurrency Restrictions of EADD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EADD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGE-INFO.SECS	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EADD Operational Flow

Name	Type	Size (bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

IF (DS:RBX is not 32Byte Aligned)
THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
TMP_LINADDR := DS:RBX.LINADDR;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECS is not 4KByte aligned or DS:TMP_SECINFO is not 64Byte aligned or TMP_LINADDR is not 4KByte aligned)
THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
THEN #PF(DS:TMP_SECS); FI;

SCRATCH_SECINFO := DS:TMP_SECINFO;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero or

```

!(SCRATCH_SECINFO.FLAGS.PT is PT_REG or SCRATCH_SECINFO.FLAGS.PT is PT_TCS or
(SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
(SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
THEN #GP(0); FI;

```

```

(* If PT_SS_FIRST/PT_SS_REST page types are requested then CR4.CET must be 1 *)
IF ( (SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST OR
SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST) AND CR4.CET == 0)
THEN #GP(0); FI;

```

```

(* Check the EPC page for concurrency *)
IF (EPC page is not available for EADD)
THEN
    IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
        THEN
            VMCS.Exit_reason := SGX_CONFLICT;
            VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
            VMCS.Exit_qualification.error := 0;
            VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
            VMCS.Guest-linear_address := DS:RCX;
            Deliver VMEXIT;
        ELSE
            #GP(0);
    FI;
FI;

```

```

IF (EPCM(DS:RCX).VALID ≠ 0)
THEN #PF(DS:RCX); FI;

```

```

(* Check the SECS for concurrency *)
IF (SECS is not available for EADD)
THEN #GP(0); FI;

```

```

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
THEN #PF(DS:TMP_SECS); FI;

```

```

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPAGE[32767:0];

```

```

CASE (SCRATCH_SECINFO.FLAGS.PT)

```

```

PT_TCS:
    IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
    IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and
        ((DS:TCS.FSLIMIT & 0FFFH ≠ 0FFFH) or (DS:TCS.GSLIMIT & 0FFFH ≠ 0FFFH)) ) #GP(0); FI;
    (* Ensure TCS.PREVSSP is zero *)
    IF (CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1) and (DS:RCX.PREVSSP != 0) #GP(0); FI;
    BREAK;

```

```

PT_REG:
    IF (SCRATCH_SECINFO.FLAGS.W = 1 and SCRATCH_SECINFO.FLAGS.R = 0) #GP(0); FI;
    BREAK;

```

```

PT_SS_FIRST:

```

```

PT_SS_REST:

```

```

(* SS pages cannot be created on first or last page of ELRANGE *)

```



```

IF ( TMP_LINADDR = DS:TMP_SECS.BASEADDR or TMP_LINADDR = (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE - 0x1000) )
  THEN #GP(0); FI;
IF ( DS:RCX[4087:0] != 0 ) #GP(0); FI;
IF (SCRATCH_SECINFO.FLAGS.PT == PT_SS_FIRST)
  THEN
    (* Check that valid RSTORSSP token exists *)
    IF ( DS:RCX[4095:4088] != ((TMP_LINADDR + 0x1000) | DS:TMP_SECS.ATTRIBUTES.MODE64BIT) ) #GP(0); FI;
    (* Check the 8 bytes are zero *)
    IF ( DS:RCX[4095:4088] != 0 ) #GP(0); FI;
  FI;
IF (SCRATCH_SECINFO.FLAGS.W = 0 OR SCRATCH_SECINFO.FLAGS.R = 0 OR
  SCRATCH_SECINFO.FLAGS.X = 1) #GP(0); FI;
  BREAK;
ESAC;

```

```

(* Check the enclave offset is within the enclave linear address space *)
IF (TMP_LINADDR < DS:TMP_SECS.BASEADDR or TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE)
  THEN #GP(0); FI;

```

```

(* Check concurrency of measurement resource*)
IF (Measurement being updated)
  THEN #GP(0); FI;

```

```

(* Check if the enclave to which the page will be added is already in Initialized state *)
IF (DS:TMP_SECS already initialized)
  THEN #GP(0); FI;

```

```

(* For TCS pages, force EPCM.rwx bits to 0 and no debug access *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
  THEN
    SCRATCH_SECINFO.FLAGS.R := 0;
    SCRATCH_SECINFO.FLAGS.W := 0;
    SCRATCH_SECINFO.FLAGS.X := 0;
    (DS:RCX).FLAGS.DBGOPTIN := 0; // force TCS.FLAGS.DBGOPTIN off
    DS:RCX.CSSA := 0;
    DS:RCX.AEP := 0;
    DS:RCX.STATE := 0;
  FI;

```

```

(* Add enclave offset and security attributes to MRENCLAVE *)
TMP_ENCLAVEOFFSET := TMP_LINADDR - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] := 0000000044444145H; // "EADD"
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] := SCRATCH_SECINFO[375:0]; // 48 bytes
DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

```

```

(* Add enclave offset and security attributes to MRENCLAVE *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_LINADDR;

```

(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
 Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;

(* Set EPCM entry fields *)
 EPCM(DS:RCX).BLOCKED := 0;
 EPCM(DS:RCX).PENDING := 0;
 EPCM(DS:RCX).MODIFIED := 0;
 EPCM(DS:RCX).VALID := 1;

Flags Affected

None

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
 If a memory operand is not properly aligned.
 If an enclave memory operand is outside of the EPC.
 If an enclave memory operand is the wrong type.
 If a memory operand is locked.
 If the enclave is initialized.
 If the enclave's MRENCLAVE is locked.
 If the TCS page reserved bits are set.
 If the TCS page PREVSSP field is not zero.
 If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
 If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.

#PF(error code) If a page fault occurs in accessing memory operands.
 If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0) If a memory operand is non-canonical form.
 If a memory operand is not properly aligned.
 If an enclave memory operand is outside of the EPC.
 If an enclave memory operand is the wrong type.
 If a memory operand is locked.
 If the enclave is initialized.
 If the enclave's MRENCLAVE is locked.
 If the TCS page reserved bits are set.
 If the TCS page PREVSSP field is not zero.
 If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
 If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.

#PF(error code) If a page fault occurs in accessing memory operands.
 If the EPC page is valid.

EAUG—Add a Page to an Initialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0DH ENCLS[EAUG]	IR	V/V	SGX2	This leaf function adds a page to an initialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EAUG (In)	Address of a SECFINFO (In)	Address of the destination EPC page (In)

Description

This leaf function zeroes a page of EPC memory, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in the EPCM. As part of the association, the security attributes are configured to prevent access to the EPC page until a corresponding invocation of the EACCEPT leaf or EACCEPT-COPY leaf confirms the addition of the new page into the enclave. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EAUG leaf function.

EAUG Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Must be zero	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EAUG Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	The specified enclave offset is outside of the enclave address space.
The SECS has been initialized.	

Concurrency Restrictions

Table 38-10. Base Concurrency Restrictions of EAUG

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EAUG	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	

Table 38-11. Additional Concurrency Restrictions of EAUG

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EAUG	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGE-INFO.SECS	Concurrent		Concurrent		Concurrent	

Operation**Temp Variables in EAUG Operational Flow**

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.

IF (DS:RBX is not 32Byte Aligned)
THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
IF (DS:RBX.SECINFO is not 0)
THEN
 IF (DS:TMP_SECINFO is not 64B aligned)
 THEN #GP(0); FI;

FI;

TMP_LINADDR := DS:RBX.LINADDR;

IF (DS:TMP_SECS is not 4KByte aligned or TMP_LINADDR is not 4KByte aligned)
THEN #GP(0); FI;

IF DS:RBX.SRCPAGE is not 0
THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
THEN #PF(DS:TMP_SECS); FI;

(* Check the EPC page for concurrency *)

```

IF (EPC page in use)
  THEN
    IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
      THEN
        VMCS.Exit_reason := SGX_CONFLICT;
        VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
        VMCS.Guest-linear_address := DS:RCX;
        Deliver VMEXIT;
      ELSE
        #GP(0);
    FI;
  FI;

IF (EPCM(DS:RCX).VALID ≠ 0)
  THEN #PF(DS:RCX); FI;

(* copy SECINFO contents into a scratch SECINFO *)
IF (DS:RBX.SECINFO is 0)
  THEN
    (* allocate and initialize a new scratch SECINFO structure *)
    SCRATCH_SECINFO.PT := PT_REG;
    SCRATCH_SECINFO.R := 1;
    SCRATCH_SECINFO.W := 1;
    SCRATCH_SECINFO.X := 0;
    << zero out remaining fields of SCRATCH_SECINFO >>
  ELSE
    (* copy SECINFO contents into scratch SECINFO *)
    SCRATCH_SECINFO := DS:TMP_SECINFO;
    (* check SECINFO flags for misconfiguration *)
    (* reserved flags must be zero *)
    (* SECINFO.FLAGS.PT must either be PT_SS_FIRST, or PT_SS_REST *)
    IF ( (SCRATCH_SECINFO reserved fields are not 0) or
        CPUID.(EAX=12H, ECX=1):EAX[6] is 0) OR
        (SCRATCH_SECINFO.PT is not PT_SS_FIRST, or PT_SS_REST) OR
        ( (SCRATCH_SECINFO.FLAGS.R is 0) OR (SCRATCH_SECINFO.FLAGS.W is 0) OR (SCRATCH_SECINFO.FLAGS.X is 1) ) )
      THEN #GP(0); FI;
  FI;

(* Check if PT_SS_FIRST/PT_SS_REST page types are requested then CR4.CET must be 1 *)
IF ( (SCRATCH_SECINFO.PT is PT_SS_FIRST OR SCRATCH_SECINFO.PT is PT_SS_REST) AND CR4.CET == 0 )
  THEN #GP(0); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EAUG)
  THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
  THEN #PF(DS:TMP_SECS); FI;

(* Check if the enclave to which the page will be added is in the Initialized state *)
IF (DS:TMP_SECS is not initialized)
  THEN #GP(0); FI;

```

```
(* Check the enclave offset is within the enclave linear address space *)
IF ( (TMP_LINADDR < DS:TMP_SECS.BASEADDR) or (TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE) )
  THEN #GP(0); FI;
```

```
IF ( (SCRATCH_SECINFO.PT is PT_SS_FIRST OR SCRATCH_SECINFO.PT is PT_SS_REST) )
  THEN
    (* SS pages cannot be created on first or last page of ELRANGE *)
    IF ( TMP_LINADDR == DS:TMP_SECS.BASEADDR OR
        TMP_LINADDR == (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE - 0x1000) )
      THEN
        #GP(0); FI;
```

```
FI;
```

```
(* Clear the content of EPC page*)
```

```
DS:RCX[32767:0] := 0;
```

```
IF (CPUID.(EAX=07H, ECX=0H):ECX[CET_SS] = 1)
```

```
  THEN
```

```
    (* set up shadow stack RSTORSSP token *)
```

```
    IF (SCRATCH_SECINFO.PT is PT_SS_FIRST)
```

```
      THEN
```

```
        DS:RCX[0xFF8] := (TMP_LINADDR + 0x1000) | TMP_SECS.ATTRIBUTES.MODE64BIT; FI;
```

```
FI;
```

```
(* Set EPCM security attributes *)
```

```
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
```

```
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
```

```
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
```

```
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;
```

```
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_LINADDR;
```

```
EPCM(DS:RCX).BLOCKED := 0;
```

```
EPCM(DS:RCX).PENDING := 1;
```

```
EPCM(DS:RCX).MODIFIED := 0;
```

```
EPCM(DS:RCX).PR := 0;
```

```
(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
```

```
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;
```

```
(* Set EPCM valid fields *)
```

```
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If a memory operand is locked. If the enclave is not initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

- #GP(0) If a memory operand is non-canonical form.
 If a memory operand is not properly aligned.
 If a memory operand is locked.
 If the enclave is not initialized.
- #PF(error code) If a page fault occurs in accessing memory operands.

EBLOCK—Mark a page in EPC as Blocked

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLS[EBLOCK]	IR	V/V	SGX1	This leaf function marks a page in the EPC as blocked.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	EBLOCK (In)	Return error code (Out)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be marked as BLOCKED. This instruction can only be executed when current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

An error code is returned in RAX.

The table below provides additional information on the memory parameter of EBLOCK leaf function.

EBLOCK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 38-12. EBLOCK Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EBLOCK successful.
SGX_BLKSTATE	Page already blocked. This value is used to indicate to a VMM that the page was already in BLOCKED state as a result of EBLOCK and thus will need to be restored to this state when it is eventually reloaded (using ELDB).
SGX_ENTRYEPOCH_LOCKED	SECS locked for Entry Epoch update. This value indicates that an ETRACK is currently executing on the SECS. The EBLOCK should be reattempted.
SGX_NOTBLOCKABLE	Page type is not one which can be blocked.
SGX_PG_INVLD	Page is not valid and cannot be blocked.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB.

Concurrency Restrictions

Table 38-13. Base Concurrency Restrictions of EBLOCK

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EBLOCK	Target [DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	

Table 38-14. Additional Concurrency Restrictions of EBLOCK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EBLOCK	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EBLOCK Operational Flow

Name	Type	Size (Bits)	Description
TMP_BLKSTATE	Integer	64	Page is already blocked.

IF (DS:RCX is not 4KByte Aligned)
 THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
 THEN #PF(DS:RCX); FI;

RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
 RAX := 0;

(* Check the EPC page for concurrency*)

IF (EPC page in use)
 THEN
 RFLAGS.ZF := 1;
 RAX := SGX_EPC_PAGE_CONFLICT;
 GOTO DONE;

FI;

IF (EPCM(DS:RCX).VALID = 0)
 THEN
 RFLAGS.ZF := 1;
 RAX := SGX_PG_INVLD;
 GOTO DONE;

FI;

IF ((EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_TRIM)
 and EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
 THEN
 RFLAGS.CF := 1;
 IF (EPCM(DS:RCX).PT = PT_SECS)
 THEN RAX := SGX_PG_IS_SECS;
 ELSE RAX := SGX_NOTBLOCKABLE;
 FI;
 GOTO DONE;

FI;

(* Check if the page is already blocked and report blocked state *)

TMP_BLKSTATE := EPCM(DS:RCX).BLOCKED;

```
(* at this point, the page must be valid and PT_TCS or PT_REG or PT_TRIM*)
IF (TMP_BLKSTATE = 1)
  THEN
    RFLAGS.CF := 1;
    RAX := SGX_BLKSTATE;
  ELSE
    EPCM(DS:RCX).BLOCKED := 1
FI;
DONE:
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Sets CF if page is BLOCKED or not blockable, otherwise cleared. Clears PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form. If a memory operand is not properly aligned. If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

ECREATE—Create an SECS page in the Enclave Page Cache

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLS[ECREATE]	IR	V/V	SGX1	This leaf function begins an enclave build by creating an SECS page in EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	ECREATE (In)	Address of a PAGEINFO (In)	Address of the destination SECS page (In)

Description

ENCLS[ECREATE] is the first instruction executed in the enclave build process. ECREATE copies an SECS structure outside the EPC into an SECS page inside the EPC. The internal structure of SECS is not accessible to software.

ECREATE will set up fields in the protected SECS and mark the page as valid inside the EPC. ECREATE initializes or checks unused fields.

Software sets the following fields in the source structure: SECS:BASEADDR, SECS:SIZE in bytes, ATTRIBUTES, CONFIGID, and CONFIGSVN. SECS:BASEADDR must be naturally aligned on an SECS.SIZE boundary. SECS.SIZE must be at least 2 pages (8192).

The source operand RBX contains an effective address of a PAGEINFO structure. PAGEINFO contains an effective address of a source SECS and an effective address of an SECINFO. The SECS field in PAGEINFO is not used.

The RCX register is the effective address of the destination SECS. It is an address of an empty slot in the EPC. The SECS structure must be page aligned. SECINFO flags must specify the page as an SECS page.

ECREATE Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

ECREATE will fault if the SECS target page is in use; already valid; outside the EPC. It will also fault if addresses are not aligned; unused PAGEINFO fields are not zero.

If the amount of space needed to store the SSA frame is greater than the amount specified in SECS.SSAFRAME-SIZE, a #GP(0) results. The amount of space needed for an SSA frame is computed based on DS:TMP_SECS.ATTRIBUTES.XFRM size. Details of computing the size can be found Section 39.7.

Concurrency Restrictions

Table 38-15. Base Concurrency Restrictions of ECREATE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ECREATE	SECS [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 38-16. Additional Concurrency Restrictions of ECREATE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ECREATE	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ECREATE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the SECS source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the SECS page to be added.
TMP_XSIZE	SSA Size	64	The size calculation of SSA frame.
TMP_MISC_SIZE	MISC Field Size	64	Size of the selected MISC field components.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

IF (DS:RBX is not 32Byte Aligned)
THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECINFO := DS:RBX.SECINFO;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECINFO is not 64Byte aligned)
THEN #GP(0); FI;

IF (DS:RBX.LINADDR != 0 or DS:RBX.SECS != 0)
THEN #GP(0); FI;

(* Check for misconfigured SECINFO flags*)

IF (DS:TMP_SECINFO reserved fields are not zero or DS:TMP_SECINFO.FLAGS.PT != PT_SECS)
THEN #GP(0); FI;

TMP_SECS := RCX;

IF (EPC entry in use)
THEN

IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
THEN
VMCS.Exit_reason := SGX_CONFLICT;

```

        VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address :=
            << translation of DS:TMP_SECS produced by paging >>;
        VMCS.Guest-linear_address := DS:TMP_SECS;
    Deliver VMEXIT;
    ELSE
        #GP(0);
FI;

IF (EPC entry in use)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPAGE[32767:0];

(* Check lower 2 bits of XFRM are set *)
IF ( ( DS:TMP_SECS.ATTRIBUTES.XFRM BitwiseAND 03H) ≠ 03H)
    THEN #GP(0); FI;

IF (XFRM is illegal)
    THEN #GP(0); FI;

(* Check legality of CET_ATTRIBUTES *)
IF ((DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_ATTRIBUTES ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[5:2] ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[1:0] ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1 and
    (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) not canonical) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0 and
    (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) & 0xFFFFFFFF00000000) ||
    (DS:TMP_SECS.CET_ATTRIBUTES.reserved fields not 0) or
    (DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) is not page aligned))
    THEN
        #GP(0);
FI;

(* Make sure that the SECS does not have any unsupported MISCSELECT options*)
IF ( !(CPUID.(EAX=12H, ECX=0):EBX[31:0] & DS:TMP_SECS.MISCSELECT[31:0]) )
    THEN
        EPCM(DS:TMP_SECS).EntryLock.Release();
        #GP(0);
FI;

(* Compute size of MISC area *)
TMP_MISC_SIZE := compute_misc_region_size();

(* Compute the size required to save state of the enclave on async exit, see Section 39.7.2.2*)

```

```
TMP_XSIZE := compute_xsave_size(DS:TMP_SECS.ATTRIBUTES.XFRM) + GPR_SIZE + TMP_MISC_SIZE;
```

```
(* Ensure that the declared area is large enough to hold XSAVE and GPR stat *)
```

```
IF ( DS:TMP_SECS.SSAFRAMESIZE*4096 < TMP_XSIZE)
```

```
    THEN #GP(0); FI;
```

```
IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.BASEADDR is not canonical) )
```

```
    THEN #GP(0); FI;
```

```
IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.BASEADDR and 0FFFFFFFF00000000H) )
```

```
    THEN #GP(0); FI;
```

```
IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[7:0] ) ) )
```

```
    THEN #GP(0); FI;
```

```
IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[15:8] ) ) )
```

```
    THEN #GP(0); FI;
```

```
(* Enclave size must be at least 8192 bytes and must be power of 2 in bytes*)
```

```
IF (DS:TMP_SECS.SIZE < 8192 or popcnt(DS:TMP_SECS.SIZE) > 1)
```

```
    THEN #GP(0); FI;
```

```
(* Ensure base address of an enclave is aligned on size*)
```

```
IF ( ( DS:TMP_SECS.BASEADDR and (DS:TMP_SECS.SIZE-1) ) )
```

```
    THEN #GP(0); FI;
```

```
(* Ensure the SECS does not have any unsupported attributes*)
```

```
IF ( DS:TMP_SECS.ATTRIBUTES and (~CR_SGX_ATTRIBUTES_MASK) )
```

```
    THEN #GP(0); FI;
```

```
IF ( DS:TMP_SECS reserved fields are not zero)
```

```
    THEN #GP(0); FI;
```

```
(* Verify that CONFIGID/CONFIGSVN are not set with attribute *)
```

```
IF ( ((DS:TMP_SECS.CONFIGID ≠ 0) or (DS:TMP_SECS.CONFIGSVN ≠ 0)) AND (DS:TMP_SECS.ATTRIBUTES.KSS == 0) )
```

```
    THEN #GP(0); FI;
```

```
Clear DS:TMP_SECS to Uninitialized;
```

```
DS:TMP_SECS.MRENCLAVE := SHA256INITIALIZE(DS:TMP_SECS.MRENCLAVE);
```

```
DS:TMP_SECS.ISVSVN := 0;
```

```
DS:TMP_SECS.ISVPRODID := 0;
```

```
(* Initialize hash updates etc*)
```

```
Initialize enclave's MRENCLAVE update counter;
```

```
(* Add "ECREATE" string and SECS fields to MRENCLAVE *)
```

```
TMPUPDATEFIELD[63:0] := 0045544145524345H; // "ECREATE"
```

```
TMPUPDATEFIELD[95:64] := DS:TMP_SECS.SSAFRAMESIZE;
```

```
TMPUPDATEFIELD[159:96] := DS:TMP_SECS.SIZE;
```

```
IF (CPUID.(EAX=7, ECX=0):.EDX[CET_IBT] = 1)
```

```
    THEN
```

```
        TMPUPDATEFIELD[223:160] := DS:TMP_SECS.CET_LEG_BITMAP_OFFSET;
```

```
    ELSE
```

```
        TMPUPDATEFIELD[223:160] := 0;
```

```

FI;
TMPUPDATEFIELD[511:160] := 0;
DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

```

(* Set EID *)

```
DS:TMP_SECS.EID := LockedXAdd(CR_NEXT_EID, 1);
```

(* Initialize the virtual child count to zero *)

```
DS:TMP_SECS.VIRTCHILDCNT := 0;
```

(* Load ENCLAVECONTEXT with Address out of paging of SECS *)

```
<< store translation of DS:RCX produced by paging in SECS(DS:RCX).ENCLAVECONTEXT >>
```

(* Set the EPCM entry, first create SECS identifier and store the identifier in EPCM *)

```

EPCM(DS:TMP_SECS).PT := PT_SECS;
EPCM(DS:TMP_SECS).ENCLAVEADDRESS := 0;
EPCM(DS:TMP_SECS).R := 0;
EPCM(DS:TMP_SECS).W := 0;
EPCM(DS:TMP_SECS).X := 0;

```

(* Set EPCM entry fields *)

```

EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).VALID := 1;

```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If the reserved fields are not zero. If PAGEINFO.SECS is not zero. If PAGEINFO.LINADDR is not zero. If the SECS destination is locked. If SECS.SSAFRAMESIZE is insufficient.
#PF(error code)	<ul style="list-style-type: none"> If a page fault occurs in accessing memory operands. If the SECS destination is outside the EPC.

64-Bit Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory address is non-canonical form. If a memory operand is not properly aligned. If the reserved fields are not zero. If PAGEINFO.SECS is not zero. If PAGEINFO.LINADDR is not zero. If the SECS destination is locked. If SECS.SSAFRAMESIZE is insufficient.
--------	--

#PF(error code) If a page fault occurs in accessing memory operands.
If the SECS destination is outside the EPC.

EDBGRD—Read From a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLS[EDBGRD]	IR	V/V	SGX1	This leaf function reads a dword/quadword from a debug enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDBGRD (In)	Return error code (Out)	Data read from a debug enclave (Out)	Address of source memory in the EPC (In)

Description

This leaf function copies a quadword/doubleword from an EPC page belonging to a debug enclave into the RBX register. Eight bytes are read in 64-bit mode, four bytes are read in non-64-bit modes. The size of data read cannot be overridden.

The effective address of the source location inside the EPC is provided in the register RCX.

EDBGRD Memory Parameter Semantics

EPCQW
Read access permitted by Enclave

The error codes are:

Table 38-17. EDBGRD Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EDBGRD successful.
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state.

The instruction faults if any of the following:

EDBGRD Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is beyond the architectural size of the TCS (SGX_TCS_LIMIT).
An operand causing any segment violation.	May page fault.
CPL > 0.	

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDBGRD does not result in a #GP.

Concurrency Restrictions

Table 38-18. Base Concurrency Restrictions of EDBGD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDBGD	Target [DS:RCX]	Shared	#GP	

Table 38-19. Additional Concurrency Restrictions of EDBGD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGD Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1))
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.

TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ((TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned))
THEN #GP(0); FI;

IF ((TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned))
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing the same EPCM entry *)

IF (Another instruction modifying the same EPCM entry is executing)
THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 0)
THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (SOURCE) is pointing to a PT_REG or PT_TCS or PT_VA or PT_SS_FIRST or PT_SS_REST *)

IF ((EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_VA)
and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX points to an accessible EPC page *)

IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0))
THEN
RFLAGS.ZF := 1;

```

    RAX := SGX_PAGE_NOT_DEBUGGABLE;
    GOTO DONE;
FI;

(* If source is a TCS, then make sure that the offset into the page is not beyond the TCS size*)
IF ( ( EPCM(DS:RCX).PT = PT_TCS) and ((DS:RCX) & FFFH ≥ SGX_TCS_LIMIT) )
    THEN #GP(0); FI;

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF ( (EPCM(DS:RCX).PT = PT_REG) or (EPCM(DS:RCX).PT = PT_TCS) )
    THEN
        TMP_SECS := GET_SECS_ADDRESS;
        IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
            THEN #GP(0); FI;
        IF ( (TMP_MODE64 = 1) )
            THEN RBX[63:0] := (DS:RCX)[63:0];
            ELSE EBX[31:0] := (DS:RCX)[31:0];
        FI;
    ELSE
        TMP_64BIT_VAL[63:0] := (DS:RCX)[63:0] & (~07H); // Read contents from VA slot
        IF (TMP_MODE64 = 1)
            THEN
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN RBX[63:0] := 0FFFFFFFFFFFFFFFFH;
                    ELSE RBX[63:0] := 0H;
                FI;
            ELSE
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN EBX[31:0] := 0FFFFFFFFH;
                    ELSE EBX[31:0] := 0H;
                FI;
            FI;
    FI;

(* clear EAX and ZF to indicate successful completion *)
RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0;

```

Flags Affected

ZF is set if the page is MODIFIED or PENDING; RAX contains the error code. Otherwise ZF is cleared and RAX is set to 0. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights. If DS segment is unusable. If RCX points to a memory location not 4Byte-aligned. If the address in RCX points to a page belonging to a non-debug enclave. If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA. If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.
--------	---

#PF(error code) If a page fault occurs in accessing memory operands.
If the address in RCX points to a non-EPC page.
If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0) If RCX is non-canonical form.
If RCX points to a memory location not 8Byte-aligned.
If the address in RCX points to a page belonging to a non-debug enclave.
If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA.
If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.

#PF(error code) If a page fault occurs in accessing memory operands.
If the address in RCX points to a non-EPC page.
If the address in RCX points to an invalid EPC page.

EDBGWR—Write to a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLS[EDBGWR]	IR	V/V	SGX1	This leaf function writes a dword/quadword to a debug enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDBGWR (In)	Return error code (Out)	Data to be written to a debug enclave (In)	Address of Target memory in the EPC (In)

Description

This leaf function copies the content in EBX/RBX to an EPC page belonging to a debug enclave. Eight bytes are written in 64-bit mode, four bytes are written in non-64-bit modes. The size of data cannot be overridden. The effective address of the target location inside the EPC is provided in the register RCX.

EDBGWR Memory Parameter Semantics

EPCQW
Write access permitted by Enclave

The instruction faults if any of the following:

EDBGWR Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is not the FLAGS word.
An operand causing any segment violation.	May page fault.
CPL > 0.	

The error codes are:

Table 38-20. EDBGWR Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EDBGWR successful.
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state.

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDBGWR does not result in a #GP.

Concurrency Restrictions

Table 38-21. Base Concurrency Restrictions of EDBGWR

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDBGWR	Target [DS:RCX]	Shared	#GP	

Table 38-22. Additional Concurrency Restrictions of EDBGWR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGWR	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGWR Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1)).
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.

TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ((TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned))
THEN #GP(0); FI;

IF ((TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned))
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing the same EPCM entry *)

IF (Another instruction modifying the same EPCM entry is executing)
THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 0)
THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS or PT_SS_FIRST or PT_SS_REST *)

IF ((EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS)
and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX points to an accessible EPC page *)

IF ((EPCM(DS:RCX).PENDING is not 0) or (EPCM(DS:RCX).MODIFIED is not 0))
THEN
RFLAGS.ZF := 1;

INTEL® SGX INSTRUCTION REFERENCES

```
RAX := SGX_PAGE_NOT_DEBUGGABLE;
GOTO DONE;
FI;

(* If destination is a TCS, then make sure that the offset into the page can only point to the FLAGS field*)
IF ( ( EPCM(DS:RCX).PT = PT_TCS) and ((DS:RCX) & FF8H ≠ offset_of_FLAGS & OFF8H) )
    THEN #GP(0); FI;

(* Locate the SECS for the enclave to which the DS:RCX page belongs *)
TMP_SECS := GET_SECS_PHYS_ADDRESS(EPCM(DS:RCX).ENCLAVESECS);

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 1) )
    THEN (DS:RCX)[63:0] := RBX[63:0];
    ELSE (DS:RCX)[31:0] := EBX[31:0];
FI;

(* clear EAX and ZF to indicate successful completion *)
RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0
```

Flags Affected

ZF is set if the page is MODIFIED or PENDING; RAX contains the error code. Otherwise ZF is cleared and RAX is set to 0. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights. If DS segment is unusable. If RCX points to a memory location not 4Byte-aligned. If the address in RCX points to a page belonging to a non-debug enclave. If the address in RCX points to a page which is not PT_TCS or PT_REG. If the address in RCX points to a location inside TCS that is not the FLAGS word.
#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form. If RCX points to a memory location not 8Byte-aligned. If the address in RCX points to a page belonging to a non-debug enclave. If the address in RCX points to a page which is not PT_TCS or PT_REG. If the address in RCX points to a location inside TCS that is not the FLAGS word.
--------	--

#PF(error code) If a page fault occurs in accessing memory operands.
 If the address in RCX points to a non-EPC page.
 If the address in RCX points to an invalid EPC page.

EEXTEND—Extend Uninitialized Enclave Measurement by 256 Bytes

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLS[EEXTEND]	IR	V/V	SGX1	This leaf function measures 256 bytes of an uninitialized enclave page.

Instruction Operand Encoding

Op/En	EAX	EBX	RCX
IR	EEXTEND (In)	Effective address of the SECS of the data chunk (In)	Effective address of a 256-byte chunk in the EPC (In)

Description

This leaf function updates the MRENCLAVE measurement register of an SECS with the measurement of an EXTEND string comprising of “EEXTEND” || ENCLAVEOFFSET || PADDING || 256 bytes of the enclave page. This instruction can only be executed when current privilege level is 0 and the enclave is uninitialized.

RBX contains the effective address of the SECS of the region to be measured. The address must be the same as the one used to add the page into the enclave.

RCX contains the effective address of the 256 byte region of an EPC page to be measured. The DS segment is used to create linear addresses. Segment override is not supported.

EEXTEND Memory Parameter Semantics

EPC[RCX]
Read access by Enclave

The instruction faults if any of the following:

EEXTEND Faulting Conditions

RBX points to an address not 4KBytes aligned.	RBX does not resolve to an SECS.
RBX does not point to an SECS page.	RBX does not point to the SECS page of the data chunk.
RCX points to an address not 256B aligned.	RCX points to an unused page or a SECS.
RCX does not resolve in an EPC page.	If SECS is locked.
If the SECS is already initialized.	May page fault.
CPL > 0.	

Concurrency Restrictions

Table 38-23. Base Concurrency Restrictions of EEXTEND

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EEXTEND	Target [DS:RCX]	Shared	#GP	
	SECS [DS:RBX]	Concurrent		

Table 38-24. Additional Concurrency Restrictions of EEXTEND

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EEXTEND	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EEXTEND Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.
TMP_ENCLAVEOFFS ET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));
```

```
IF (DS:RBX is not 4096 Byte Aligned)
  THEN #GP(0); FI;
```

```
IF (DS:RBX does not resolve to an EPC page)
  THEN #PF(DS:RBX); FI;
```

```
IF (DS:RCX is not 256Byte Aligned)
  THEN #GP(0); FI;
```

```
IF (DS:RCX does not resolve within an EPC)
  THEN #PF(DS:RCX); FI;
```

```
(* make sure no other Intel SGX instruction is accessing EPCM *)
IF (Other instructions accessing EPCM)
  THEN #GP(0); FI;
```

```
IF (EPCM(DS:RCX). VALID = 0)
  THEN #PF(DS:RCX); FI;
```

```
(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS)
  and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
  THEN #PF(DS:RCX); FI;
```

```
TMP_SECS := Get_SECS_ADDRESS();
```

```
IF (DS:RBX does not resolve to TMP_SECS)
  THEN #GP(0); FI;
```

```
(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)
IF ( (Other instruction accessing MRENCLAVE) or (Other instructions checking or updating the initialized state of the SECS))
```

```
THEN #GP(0); FI;
```

```
(* Calculate enclave offset *)
```

```
TMP_ENCLAVEOFFSET := EPCM(DS:RCX).ENCLAVEADDRESS - TMP_SECS.BASEADDR;
```

```
TMP_ENCLAVEOFFSET := TMP_ENCLAVEOFFSET + (DS:RCX & 0FFFH)
```

```
(* Add EEXTEND message and offset to MRENCLAVE *)
```

```
TMPUPDATEFIELD[63:0] := 00444E4554584545H; // "EEXTEND"
```

```
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
```

```
TMPUPDATEFIELD[511:128] := 0; // 48 bytes
```

```
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
```

```
INC enclave's MRENCLAVE update counter;
```

```
(*Add 256 bytes to MRENCLAVE, 64 byte at a time *)
```

```
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[511:0] );
```

```
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1023: 512] );
```

```
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1535: 1024] );
```

```
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[2047: 1536] );
```

```
INC enclave's MRENCLAVE update counter by 4;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	<p>If the address in RBX is outside the DS segment limit.</p> <p>If RBX points to an SECS page which is not the SECS of the data chunk.</p> <p>If the address in RCX is outside the DS segment limit.</p> <p>If RCX points to a memory location not 256Byte-aligned.</p> <p>If another instruction is accessing MRENCLAVE.</p> <p>If another instruction is checking or updating the SECS.</p> <p>If the enclave is already initialized.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory operands.</p> <p>If the address in RBX points to a non-EPC page.</p> <p>If the address in RCX points to a page which is not PT_TCS or PT_REG.</p> <p>If the address in RCX points to a non-EPC page.</p> <p>If the address in RCX points to an invalid EPC page.</p>

64-Bit Mode Exceptions

#GP(0)	<p>If RBX is non-canonical form.</p> <p>If RBX points to an SECS page which is not the SECS of the data chunk.</p> <p>If RCX is non-canonical form.</p> <p>If RCX points to a memory location not 256 Byte-aligned.</p> <p>If another instruction is accessing MRENCLAVE.</p> <p>If another instruction is checking or updating the SECS.</p> <p>If the enclave is already initialized.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory operands.</p> <p>If the address in RBX points to a non-EPC page.</p> <p>If the address in RCX points to a page which is not PT_TCS or PT_REG.</p> <p>If the address in RCX points to a non-EPC page.</p> <p>If the address in RCX points to an invalid EPC page.</p>

EINIT—Initialize an Enclave for Execution

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLS[EINIT]	IR	V/V	SGX1	This leaf function initializes the enclave and makes it ready to execute enclave code.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EINIT (In)	Error code (Out)	Address of SIGSTRUCT (In)	Address of SECS (In)	Address of EINITOKEN (In)

Description

This leaf function is the final instruction executed in the enclave build process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to start user code execution using the EENTER instruction.

EINIT takes the effective address of a SIGSTRUCT and EINITOKEN. The SIGSTRUCT describes the enclave including MRENCLAVE, ATTRIBUTES, ISVSVN, a 3072 bit RSA key, and a signature using the included key. SIGSTRUCT must be populated with two values, q1 and q2. These are calculated using the formulas shown below:

$$q1 = \text{floor}(\text{Signature}^2 / \text{Modulus});$$

$$q2 = \text{floor}((\text{Signature}^3 - q1 * \text{Signature} * \text{Modulus}) / \text{Modulus});$$

The EINITOKEN contains the MRENCLAVE, MRSIGNER, and ATTRIBUTES. These values must match the corresponding values in the SECS. If the EINITOKEN was created with a debug launch key, the enclave must be in debug mode as well.

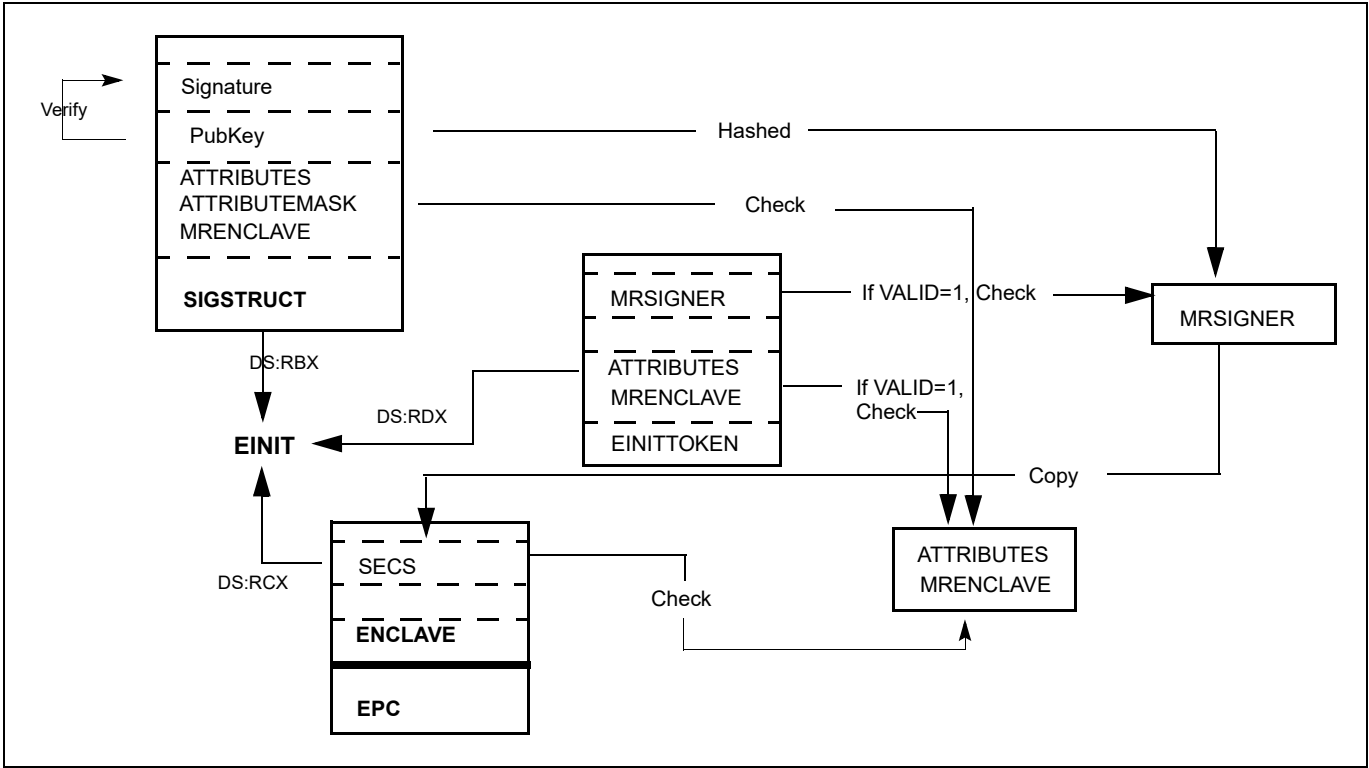


Figure 38-1. Relationships Between SECS, SIGSTRUCT, and EINITOKEN

EINIT Memory Parameter Semantics

SIGSTRUCT	SECS	EINITTOKEN
Access by non-Enclave	Read/Write access by Enclave	Access by non-Enclave

EINIT performs the following steps, which can be seen in Figure 38-1:

1. Validates that SIGSTRUCT is signed using the enclosed public key.
2. Checks that the completed computation of SECS.MRENCLAVE equals SIGSTRUCT.HASHENCLAVE.
3. Checks that no controlled ATTRIBUTES bits are set in SIGSTRUCT.ATTRIBUTES unless the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
4. Checks that the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SIGSTRUCT.ATTRIBUTES equals the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.
5. If EINITTOKEN.VALID is 0, checks that the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
6. If EINITTOKEN.VALID is 1, checks the validity of EINITTOKEN.
7. If EINITTOKEN.VALID is 1, checks that EINITTOKEN.MRENCLAVE equals SECS.MRENCLAVE.
8. If EINITTOKEN.VALID is 1 and EINITTOKEN.ATTRIBUTES.DEBUG is 1, SECS.ATTRIBUTES.DEBUG must be 1.
9. Commits SECS.MRENCLAVE, and sets SECS.MRSIGNER, SECS.ISVSVN, and SECS.ISVPRODID based on SIGSTRUCT.
10. Update the SECS as Initialized.

Periodically, EINIT polls for certain asynchronous events. If such an event is detected, it completes with failure code (ZF=1 and RAX = SGX_UNMASKED_EVENT), and RIP is incremented to point to the next instruction. These events includes external interrupts, non-maskable interrupts, system-management interrupts, machine checks, INIT signals, and the VMX-preemption timer. EINIT does not fail if the pending event is inhibited (e.g., external interrupts could be inhibited due to blocking by MOV SS blocking or by STI).

The following bits in RFLAGS are cleared: CF, PF, AF, OF, and SF. When the instruction completes with an error, RFLAGS.ZF is set to 1, and the corresponding error bit is set in RAX. If no error occurs, RFLAGS.ZF is cleared and RAX is set to 0.

The error codes are:

Table 38-25. EINIT Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EINIT successful.
SGX_INVALID_SIG_STRUCT	If SIGSTRUCT contained an invalid value.
SGX_INVALID_ATTRIBUTE	If SIGSTRUCT contains an unauthorized attributes mask.
SGX_INVALID_MEASUREMENT	If SIGSTRUCT contains an incorrect measurement. If EINITTOKEN contains an incorrect measurement.
SGX_INVALID_SIGNATURE	If signature does not validate with enclosed public key.
SGX_INVALID_LICENSE	If license is invalid.
SGX_INVALID_CPUSVN	If license SVN is unsupported.
SGX_UNMASKED_EVENT	If an unmasked event is received before the instruction completes its operation.

Concurrency Restrictions

Table 38-26. Base Concurrency Restrictions of EINIT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EINIT	SECS [DS:RCX]	Shared	#GP	

Table 38-27. Additional Concurrency Restrictions of EINIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EINIT	SECS [DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EINIT Operational Flow

Name	Type	Size	Description
TMP_SIG	SIGSTRUCT	1808Bytes	Temp space for SIGSTRUCT.
TMP_TOKEN	EINITTOKEN	304Bytes	Temp space for EINITTOKEN.
TMP_MRENCLAVE		32Bytes	Temp space for calculating MRENCLAVE.
TMP_MRSIGNER		32Bytes	Temp space for calculating MRSIGNER.
CONTROLLED_ATTRIBUTES	ATTRIBUTES	16Bytes	Constant mask of all ATTRIBUTE bits that can only be set for authorized enclaves.
TMP_KEYDEPENDENCIES	Buffer	224Bytes	Temp space for key derivation.
TMP_EINITTOKENKEY		16Bytes	Temp space for the derived EINITTOKEN Key.
TMP_SIG_PADDING	PKCS Padding Buffer	352Bytes	The value of the top 352 bytes from the computation of Signature ³ modulo MRSIGNER.

(* make sure SIGSTRUCT and SECS are aligned *)

IF ((DS:RBX is not 4KByte Aligned) or (DS:RCX is not 4KByte Aligned))
THEN #GP(0); FI;

(* make sure the EINITTOKEN is aligned *)

IF (DS:RDX is not 512Byte Aligned)
THEN #GP(0); FI;

(* make sure the SECS is inside the EPC *)

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

TMP_SIG[14463:0] := DS:RBX[14463:0]; // 1808 bytes

TMP_TOKEN[2423:0] := DS:RDX[2423:0]; // 304 bytes

(* Verify SIGSTRUCT Header. *)

```
IF ( (TMP_SIG.HEADER ≠ 06000000E10000000000010000000000h) or
    ((TMP_SIG.VENDOR ≠ 0) and (TMP_SIG.VENDOR ≠ 00008086h) ) or
    (TMP_SIG.HEADER2 ≠ 01010000600000006000000001000000h) or
    (TMP_SIG.EXPONENT ≠ 00000003h) or (Reserved space is not 0's) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_SIG_STRUCT;
        GOTO EXIT;
```

FI;

(* Open “Event Window” Check for Interrupts. Verify signature using embedded public key, q1, and q2. Save upper 352 bytes of the PKCS1.5 encoded message into the TMP_SIG_PADDING*)

```
IF (interrupt was pending) THEN
    RFLAGS.ZF := 1;
    RAX := SGX_UNMASKED_EVENT;
    GOTO EXIT;
```

FI

```
IF (signature failed to verify) THEN
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_SIGNATURE;
    GOTO EXIT;
```

FI;

(*Close “Event Window” *)

(* make sure no other Intel SGX instruction is modifying SECS*)

```
IF (Other instructions modifying SECS)
    THEN #GP(0); FI;
```

```
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PT ≠ PT_SECS) )
    THEN #PF(DS:RCX); FI;
```

(* Verify ISVFAMILYID is not used on an enclave with KSS disabled *)

```
IF ((TMP_SIG.ISVFAMILYID != 0) AND (DS:RCX.ATTRIBUTES.KSS == 0))
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_SIG_STRUCT;
        GOTO EXIT;
```

FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)

```
IF ( (Other instruction modifying MRENCLAVE) or (Other instructions modifying the SECS's Initialized state))
    THEN #GP(0); FI;
```

(* Calculate finalized version of MRENCLAVE *)

(* SHA256 algorithm requires one last update that compresses the length of the hashed message into the output SHA256 digest *)

```
TMP_ENCLAVE := SHA256FINAL( (DS:RCX).MRENCLAVE, enclave's MRENCLAVE update count *512);
```

(* Verify MRENCLAVE from SIGSTRUCT *)

```
IF (TMP_SIG.ENCLAVEHASH ≠ TMP_MRENCLAVE)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_MEASUREMENT;
    GOTO EXIT;
```

FI;

```
TMP_MRSIGNER := SHA256(TMP_SIG.MODULUS)
```

```
(* if controlled ATTRIBUTES are set, SIGSTRUCT must be signed using an authorized key *)
```

```
CONTROLLED_ATTRIBUTES := 0000000000000020H;
```

```
IF ( (DS:RCX.ATTRIBUTES & CONTROLLED_ATTRIBUTES) ≠ 0) and (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH) )
```

```
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;
```

```
FI;
```

```
(* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
```

```
IF ( (DS:RCX.ATTRIBUTES & TMP_SIG.ATTRIBUTEMASK) ≠ (TMP_SIG.ATTRIBUTE & TMP_SIG.ATTRIBUTEMASK) )
```

```
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;
```

```
FI;
```

```
(*Verify SIGSTRUCT.MISCSELECT requirements are met *)
```

```
IF ( (DS:RCX.MISCSELECT & TMP_SIG.MISCMASK) ≠ (TMP_SIG.MISCSELECT & TMP_SIG.MISCMASK) )
```

```
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT
```

```
FI;
```

```
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
```

```
    IF ( DS:RCX.CET_ATTRIBUTES & TMP_SIG.CET_ATTRIBUTES_MASK ≠ TMP_SIG.CET_ATTRIBUTES &
        TMP_SIG.CET_ATTRIBUTES_MASK )
```

```
        THEN
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_ATTRIBUTE;
            GOTO EXIT
```

```
    FI;
```

```
FI;
```

```
(* If EINITTOKEN.VALID[0] is 0, verify the enclave is signed by an authorized key *)
```

```
IF (TMP_TOKEN.VALID[0] = 0)
```

```
    IF (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH)
```

```
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_EINITTOKEN;
        GOTO EXIT;
```

```
    FI;
```

```
    GOTO COMMIT;
```

```
FI;
```

```
(* Debug Launch Enclave cannot launch Production Enclaves *)
```

```
IF ( (DS:RDX.MASKEDATTRIBUTESLE.DEBUG = 1) and (DS:RCX.ATTRIBUTES.DEBUG = 0) )
```

```
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
```

```
FI;
```


(* Check reserve space in EINIT token includes reserved regions and upper bits in valid field *)

IF (TMP_TOKEN.reserved space is not clear)

```
RFLAGS.ZF := 1;
RAX := SGX_INVALID_EINITTOKEN;
GOTO EXIT;
```

FI;

(* EINIT token must not have been created by a configuration beyond the current CPU configuration *)

IF (TMP_TOKEN.CPUSVN must not be a configuration beyond CR_CPUSVN)

```
RFLAGS.ZF := 1;
RAX := SGX_INVALID_CPUSVN;
GOTO EXIT;
```

FI;

(* Derive Launch key used to calculate EINITTOKEN.MAC *)

```
HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;
HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;
```

```
TMP_KEYDEPENDENCIES.KEYNAME := EINITTOKEN_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := TMP_TOKEN.ISVPRODIDLE;
TMP_KEYDEPENDENCIES.ISVSVN := TMP_TOKEN.ISVSVNLE;
TMP_KEYDEPENDENCIES.SGXOWNERPOUCH := CR_SGXOWNERPOUCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_TOKEN.MASKEDATTRIBUTESLE;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
TMP_KEYDEPENDENCIES.MRSIGNER := IA32_SGXLEPUBKEYHASH;
TMP_KEYDEPENDENCIES.KEYID := TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := TMP_TOKEN.CPUSVNLE;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_TOKEN.MASKEDMISCSELECTLE;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := 0;
TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_TOKEN.CET_MASKED_ATTRIBUTES_LE;
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
```

FI;

(* Calculate the derived key*)

```
TMP_EINITTOKENKEY := derivekey(TMP_KEYDEPENDENCIES);
```

(* Verify EINITTOKEN was generated using this CPU's Launch key and that it has not been modified since issuing by the Launch Enclave. Only 192 bytes of EINITTOKEN are CMACed *)

IF (TMP_TOKEN.MAC ≠ CMAC(TMP_EINITTOKENKEY, TMP_TOKEN[1535:0]))

```
RFLAGS.ZF := 1;
RAX := SGX_INVALID_EINITTOKEN;
GOTO EXIT;
```

FI;

```

(* Verify EINITOKEN (RDX) is for this enclave *)
IF ( (TMP_TOKEN.MRENCLAVE ≠ TMP_MRENCLAVE) or (TMP_TOKEN.MRSIGNER ≠ TMP_MRSIGNER) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_MEASUREMENT;
    GOTO EXIT;
FI;

(* Verify ATTRIBUTES in EINITOKEN are the same as the enclave's *)
IF (TMP_TOKEN.ATTRIBUTES ≠ DS:RCX.ATTRIBUTES)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINIT_ATTRIBUTE;
    GOTO EXIT;
FI;

COMMIT:
(* Commit changes to the SECS; Set ISVPRODID, ISVSVN, MRSIGNER, INIT ATTRIBUTE fields in SECS (RCX) *)
DS:RCX.MRENCLAVE := TMP_MRENCLAVE;
(* MRSIGNER stores a SHA256 in little endian implemented natively on x86 *)
DS:RCX.MRSIGNER := TMP_MRSIGNER;
DS:RCX.ISVEXTPRODID := TMP_SIG.ISVEXTPRODID;
DS:RCX.ISVPRODID := TMP_SIG.ISVPRODID;
DS:RCX.ISVSVN := TMP_SIG.ISVSVN;
DS:RCX.ISVFAMILYID := TMP_SIG.ISVFAMILYID;
DS:RCX.PADDING := TMP_SIG.PADDING;

(* Mark the SECS as initialized *)
Update DS:RCX to initialized;

(* Set RAX and ZF for success*)
RFLAGS.ZF := 0;
RAX := 0;
EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;

```

Flags Affected

ZF is cleared if successful, otherwise ZF is set and RAX contains the error code. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory operand is not properly aligned. If another instruction is modifying the SECS. If the enclave is already initialized. If the SECS.MRENCLAVE is in use.
#PF(error code)	<ul style="list-style-type: none"> If a page fault occurs in accessing memory operands. If RCX does not resolve in an EPC page. If the memory address is not a valid, uninitialized SECS.

64-Bit Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory operand is not properly aligned. If another instruction is modifying the SECS. If the enclave is already initialized. If the SECS.MRENCLAVE is in use.
--------	---

INTEL® SGX INSTRUCTION REFERENCES

#PF(error code) If a page fault occurs in accessing memory operands.
 If RCX does not resolve in an EPC page.
 If the memory address is not a valid, uninitialized SECS.

ELDB/ELDU/ELDBC/ELDUC—Load an EPC Page and Mark its State

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLS[ELDB]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as blocked.
EAX = 08H ENCLS[ELDU]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as unblocked.
EAX = 12H ENCLS[ELDBC]	IR	V/V	EAX[6]	This leaf function behaves like ELDB but with improved conflict handling for oversubscription.
EAX = 13H ENCLS[ELDUC]	IR	V/V	EAX[6]	This leaf function behaves like ELDU but with improved conflict handling for oversubscription.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	ELDB/ELDU (In)	Return error code (Out)	Address of the PAGEINFO (In)	Address of the EPC page (In)	Address of the version- array slot (In)

Description

This leaf function copies a page from regular main memory to the EPC. As part of the copying process, the page is cryptographically authenticated and decrypted. This instruction can only be executed when current privilege level is 0.

The ELDB leaf function sets the BLOCK bit in the EPCM entry for the destination page in the EPC after copying. The ELDU leaf function clears the BLOCK bit in the EPCM entry for the destination page in the EPC after copying.

RBX contains the effective address of a PAGEINFO structure; RCX contains the effective address of the destination EPC page; RDX holds the effective address of the version array slot that holds the version of the page.

The ELDBC/ELDUC leafs are very similar to ELDB and ELDU. They provide an error code on the concurrency conflict for any of the pages which need to acquire a lock. These include the destination, SECS, and VA slot.

The table below provides additional information on the memory parameter of ELDB/ELDU leaf functions.

ELDB/ELDU/ELDBC/ELBUC Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	PAGEINFO.SECS	EPCPAGE	Version-Array Slot
Non-enclave read access	Non-enclave read access	Non-enclave read access	Enclave read/write access	Read/Write access permitted by Enclave	Read/Write access per- mitted by Enclave

The error codes are:

Table 38-28. ELDB/ELDU/ELDBC/ELBUC Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	ELDB/ELDU successful.
SGX_MAC_COMPARE_FAIL	If the MAC check fails.

Concurrency Restrictions

Table 38-29. Base Concurrency Restrictions of ELDB/ELDU/ELDBC/ELBUC

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ELDB/ELDU	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA [DS:RDX]	Shared	#GP	
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	
ELDBC/ELBUC	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
	VA [DS:RDX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS [DS:RBX]PAGEINFO.SECS	Shared	SGX_EPC_PAGE_CONFLICT	

Table 38-30. Additional Concurrency Restrictions of ELDB/ELDU/ELDBC/ELBUC

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ELDB/ELDU	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
ELDBC/ELBUC	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ELDB/ELDU/ELDBC/ELBUC Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Memory page	4KBytes	
TMP_SECS	Memory page	4KBytes	
TMP_PCMD	PCMD	128 Bytes	
TMP_HEADER	MACHEADER	128 Bytes	
TMP_VER	UINT64	64	
TMP_MAC	UINT128	128	
TMP_PK	UINT128	128	Page encryption/MAC key.
SCRATCH_PCMD	PCMD	128 Bytes	

(* Check PAGEINFO and EPCPAGE alignment *)
 IF ((DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned))
 THEN #GP(0); FI;

```
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;
```

```
(* Check VASLOT alignment *)
IF (DS:RDX is not 8Byte aligned)
    THEN #GP(0); FI;
```

```
IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;
```

```
TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECONDS;
TMP_PCMD := DS:RBX.PCMD;
```

```
(* Check alignment of PAGEINFO (RBX) linked parameters. Note: PCMD pointer is overlaid on top of PAGEINFO.SECINFO field *)
IF ( (DS:TMP_PCMD is not 128Byte aligned) or (DS:TMP_SRCPGE is not 4KByte aligned) )
    THEN #GP(0); FI;
```

```
(* Check concurrency of EPC by other Intel SGX instructions *)
IF (other instructions accessing EPC)
    THEN
        IF ((EAX==07h) OR (EAX==08h)) (* ELDB/ELDU *)
            THEN
                IF (<<VMX non-root operation>> AND
                    <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
                    THEN
                        VMCS.Exit_reason := SGX_CONFLICT;
                        VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                        VMCS.Exit_qualification.error := 0;
                        VMCS.Guest-physical_address :=
                            << translation of DS:RCX produced by paging >>;
                        VMCS.Guest-linear_address := DS:RCX;
                        Deliver VMEXIT;
                    ELSE
                        #GP(0);
                    FI;
                ELSE (* ELDBC/ELDUC *)
                    IF (<<VMX non-root operation>> AND
                        <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
                        THEN
                            VMCS.Exit_reason := SGX_CONFLICT;
                            VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_ERROR;
                            VMCS.Exit_qualification.error := SGX_EPC_PAGE_CONFLICT;
                            VMCS.Guest-physical_address :=
                                << translation of DS:RCX produced by paging >>;
                            VMCS.Guest-linear_address := DS:RCX;
                            Deliver VMEXIT;
                        ELSE
                            RFLAGS.ZF := 1;
                            RFLAGS.CF := 0;
                            RAX := SGX_EPC_PAGE_CONFLICT;
                            GOTO ERROR_EXIT;
                        FI;
```

```

    FI;
FI;

(* Check concurrency of EPC and VASLOT by other Intel SGX instructions *)
IF (Other instructions modifying VA slot) THEN
    IF ((EAX==07h) OR (EAX==08h)) (* ELDB/ELDU *)
        THEN #GP(0);
    ELSE (* ELDBC/ELDUC *)
        RFLAGS.ZF := 1;
        RFLAGS.CF := 0;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO ERROR_EXIT;
    FI;
FI;

(* Verify EPCM attributes of EPC page, VA, and SECS *)
IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~OFFFH).VALID = 0) or (EPCM(DS:RDX & ~OFFFH).PT ≠ PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Copy PCMD into scratch buffer *)
SCRATCH_PCMD[1023: 0] := DS:TMP_PCMD[1023:0];

(* Zero out TMP_HEADER*)
TMP_HEADER[sizeof(TMP_HEADER)-1: 0] := 0;

TMP_HEADER.SECINFO := SCRATCH_PCMD.SECINFO;
TMP_HEADER.RSVD := SCRATCH_PCMD.RSVD;
TMP_HEADER.LINADDR := DS:RBX.LINADDR;

(* Verify various attributes of SECS parameter *)
IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
    (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) or
    (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
    (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) )
    THEN
        IF ( DS:TMP_SECS is not 4KByte aligned)
            THEN #GP(0) FI;
        IF (DS:TMP_SECS does not resolve within an EPC)
            THEN #PF(DS:TMP_SECS) FI;
        IF ( Another instruction is currently modifying the SECS) THEN
            IF ((EAX==07h) OR (EAX==08h)) (* ELDB/ELDU *)
                THEN #GP(0);
            ELSE (* ELDBC/ELDUC *)
                RFLAGS.ZF := 1;
                RFLAGS.CF := 0;
                RAX := SGX_EPC_PAGE_CONFLICT;
                GOTO ERROR_EXIT;
            FI;
        FI;
        TMP_HEADER.EID := DS:TMP_SECS.EID;
    ELSE

```

```

(* TMP_HEADER.SECINFO.FLAGS.PT is PT_SECS or PT_VA which do not have a parent SECS, and hence no EID binding *)
TMP_HEADER.EID := 0;
IF (DS:TMP_SECS ≠ 0)
    THEN #GP(0) FI;
FI;

(* Copy 4KBytes SRCPGE to secure location *)
DS:RCX[32767: 0] := DS:TMP_SRCPGE[32767: 0];
TMP_VER := DS:RDX[63:0];

(* Decrypt and MAC page. AES_GCM_DEC has 2 outputs, {plain text, MAC} *)
(* Parameters for AES_GCM_DEC {Key, Counter, ..} *)
{DS:RCX, TMP_MAC} := AES_GCM_DEC(CR_BASE_PK, TMP_VER << 32, TMP_HEADER, 128, DS:RCX, 4096);

IF ( (TMP_MAC ≠ DS:TMP_PCMD.MAC) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_MAC_COMPARE_FAIL;
        GOTO ERROR_EXIT;
FI;

(* Clear VA Slot *)
DS:RDX := 0

(* Commit EPCM changes *)
EPCM(DS:RCX).PT := TMP_HEADER.SECINFO.FLAGS.PT;
EPCM(DS:RCX).RWX := TMP_HEADER.SECINFO.FLAGS.RWX;
EPCM(DS:RCX).PENDING := TMP_HEADER.SECINFO.FLAGS.PENDING;
EPCM(DS:RCX).MODIFIED := TMP_HEADER.SECINFO.FLAGS.MODIFIED;
EPCM(DS:RCX).PR := TMP_HEADER.SECINFO.FLAGS.PR;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_HEADER.LINADDR;

IF ( ((EAX = 07H) or (EAX = 12H)) and (TMP_HEADER.SECINFO.FLAGS.PT is NOT PT_SECS or PT_VA) )
    THEN
        EPCM(DS:RCX).BLOCKED := 1;
    ELSE
        EPCM(DS:RCX).BLOCKED := 0;
FI;

IF (TMP_HEADER.SECINFO.FLAGS.PT is PT_SECS)
    << store translation of DS:RCX produced by paging in SECS(DS:RCX).ENCLAVECONTEXT >>
FI;

EPCM(DS:RCX). VALID := 1;

RAX := 0;
RFLAGS.ZF := 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;

```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

- #GP(0) If a memory operand effective address is outside the DS segment limit.
 If a memory operand is not properly aligned.
 If the instruction's EPC resource is in use by others.
 If the instruction fails to verify MAC.
 If the version-array slot is in use.
 If the parameters fail consistency checks.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If a memory operand expected to be in EPC does not resolve to an EPC page.
 If one of the EPC memory operands has incorrect page type.
 If the destination EPC page is already valid.

64-Bit Mode Exceptions

- #GP(0) If a memory operand is non-canonical form.
 If a memory operand is not properly aligned.
 If the instruction's EPC resource is in use by others.
 If the instruction fails to verify MAC.
 If the version-array slot is in use.
 If the parameters fail consistency checks.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If a memory operand expected to be in EPC does not resolve to an EPC page.
 If one of the EPC memory operands has incorrect page type.
 If the destination EPC page is already valid.

EMODPR—Restrict the Permissions of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0EH ENCLS[EMODPR]	IR	V/V	SGX2	This leaf function restricts the access rights associated with a EPC page in an initialized enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODPR (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function restricts the access rights associated with an EPC page in an initialized enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not restrict the page permissions will have no effect. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPR leaf function.

EMODPR Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODPR Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 38-31. EMODPR Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EMODPR successful.
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB.

Concurrency Restrictions

Table 38-32. Base Concurrency Restrictions of EMODPR

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODPR	Target [DS:RCX]	Shared	#GP	

Table 38-33. Additional Concurrency Restrictions of EMODPR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODPR	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	

Operation

Temp Variables in EMODPR Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

IF (DS:RBX is not 64Byte Aligned)
 THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
 THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
 THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
 IF ((SCRATCH_SECINFO reserved fields are not zero) or
 (SCRATCH_SECINFO.FLAGS.R is 0 and SCRATCH_SECINFO.FLAGS.W is not 0))
 THEN #GP(0); FI;

(* Check concurrency with SGX1 or SGX2 instructions on the EPC page *)
 IF (SGX1 or other SGX2 instructions accessing EPC page)
 THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID is 0)
 THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
 IF (EPC page in use by another SGX2 instruction)
 THEN
 RFLAGS.ZF := 1;
 RAX := SGX_EPC_PAGE_CONFLICT;
 GOTO DONE;

FI;

IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0))
 THEN
 RFLAGS.ZF := 1;
 RAX := SGX_PAGE_NOT_MODIFIABLE;

```

    GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT is not PT_REG)
    THEN #PF(DS:RCX); FI;

TMP_SECS := GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
    THEN #GP(0); FI;

(* Set the PR bit to indicate that permission restriction is in progress *)
EPCM(DS:RCX).PR := 1;

(* Update EPCM permissions *)
EPCM(DS:RCX).R := EPCM(DS:RCX).R & SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := EPCM(DS:RCX).W & SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := EPCM(DS:RCX).X & SCRATCH_SECINFO.FLAGS.X;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;

```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form. If a memory operand is not properly aligned. If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

EMODT—Change the Type of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0FH ENCLS[EMODT]	IR	V/V	SGX2	This leaf function changes the type of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function modifies the type of an EPC page. The security attributes are configured to prevent access to the EPC page at its new type until a corresponding invocation of the EACCEPT leaf confirms the modification. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODT leaf function.

EMODT Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODT Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 38-34. EMODT Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EMODT successful.
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODPR, or EWB.

Concurrency Restrictions

Table 38-35. Base Concurrency Restrictions of EMODT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODT	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR

Table 38-36. Additional Concurrency Restrictions of EMODT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODT	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	

Operation**Temp Variables in EMODT Operational Flow**

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

IF (DS:RBX is not 64Byte Aligned)
THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)

IF ((SCRATCH_SECINFO reserved fields are not zero) or
!(SCRATCH_SECINFO.FLAGS.PT is PT_TCS or SCRATCH_SECINFO.FLAGS.PT is PT_TRIM))
THEN #GP(0); FI;

(* Check concurrency with SGX1 instructions on the EPC page *)

IF (other SGX1 instructions accessing EPC page)
THEN
RFLAGS.ZF := 1;
RAX := SGX_EPC_PAGE_CONFLICT;
GOTO DONE;

FI;

IF (EPCM(DS:RCX).VALID is 0)
THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)

IF (EPC page in use by another SGX2 instruction)
THEN
RFLAGS.ZF := 1;
RAX := SGX_EPC_PAGE_CONFLICT;
GOTO DONE;

```

FI;

IF (!(EPCM(DS:RCX).PT is PT_REG or
    ((EPCM(DS:RCX).PT is PT_TCS or PT_SS_FIRST or PT_SS_REST) and SCRATCH_SECINFO.FLAGS.PT is PT_TRIM)))
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PAGE_NOT_MODIFIABLE;
        GOTO DONE;
FI;

TMP_SECS := GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
    THEN #GP(0); FI;

(* Update EPCM fields *)
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).MODIFIED := 1;
EPCM(DS:RCX).R := 0;
EPCM(DS:RCX).W := 0;
EPCM(DS:RCX).X := 0;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;

```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form. If a memory operand is not properly aligned. If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

EPA—Add Version Array

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0AH ENCLS[EPA]	IR	V/V	SGX1	This leaf function adds a Version Array to the EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EPA (In)	PT_VA (In, Constant)	Effective address of the EPC page (In)

Description

This leaf function creates an empty version array in the EPC page whose logical address is given by DS:RCX, and sets up EPCM attributes for that page. At the time of execution of this instruction, the register RBX must be set to PT_VA.

The table below provides additional information on the memory parameter of EPA leaf function.

EPA Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

Concurrency Restrictions

Table 38-37. Base Concurrency Restrictions of EPA

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EPA	VA [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 38-38. Additional Concurrency Restrictions of EPA

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EPA	VA [DS:RCX]	Concurrent	L	Concurrent		Concurrent	

Operation

IF (RBX ≠ PT_VA or DS:RCX is not 4KByte Aligned)
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)

IF (Other Intel SGX instructions accessing the page)
THEN

IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)


```

    THEN
        VMCS.Exit_reason := SGX_CONFLICT;
        VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
        VMCS.Guest-linear_address := DS:RCX;
    Deliver VMEXIT;
    ELSE
        #GP(0);
FI;
FI;

```

(* Check EPC page must be empty *)

```

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

```

(* Clears EPC page *)

```

DS:RCX[32767:0] := 0;

```

```

EPCM(DS:RCX).PT := PT_VA;
EPCM(DS:RCX).ENCLAVEADDRESS := 0;
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).RWX := 0;
EPCM(DS:RCX).VALID := 1;

```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If another Intel SGX instruction is accessing the EPC page. If RBX is not set to PT_VA.
#PF(error code)	<ul style="list-style-type: none"> If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page. If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory operand is non-canonical form. If a memory operand is not properly aligned. If another Intel SGX instruction is accessing the EPC page. If RBX is not set to PT_VA.
#PF(error code)	<ul style="list-style-type: none"> If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page. If the EPC page is valid.

ERDINFO—Read Type and Status Information About an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 10H ENCLS[ERDINFO]	IR	V/V	EAX[6]	This leaf function returns type and status information about an EPC page.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	ERDINFO (In)	Return error code (Out)	Address of a RDINFO structure (In)	Address of the destination EPC page (In)

Description

This instruction reads type and status information about an EPC page and returns it in a RDINFO structure. The STATUS field of the structure describes the status of the page and determines the validity of the remaining fields. The FLAGS field returns the EPCM permissions of the page; the page type; and the BLOCKED, PENDING, MODIFIED, and PR status of the page. For enclave pages, the ENCLAVECONTEXT field of the structure returns the value of SECS.ENCLAVECONTEXT. For non-enclave pages (e.g., VA) ENCLAVECONTEXT returns 0.

For invalid or non-EPC pages, the instruction returns an information code indicating the page's status, in addition to populating the STATUS field.

ERDINFO returns an error code if the destination EPC page is being modified by a concurrent SGX instruction.

RBX contains the effective address of a RDINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of ERDINFO leaf function.

ERDINFO Memory Parameter Semantics

RDINFO	EPCPAGE
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

ERDINFO Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A memory operand is not properly aligned.
DS segment is unusable (32b mode).	A page fault occurs in accessing memory operands.
A memory address is in a non-canonical form (64b mode).	

The error codes are:

Table 38-39. ERDINFO Return Value in RAX

Error Code	Value	Description
No Error	0	ERDINFO successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.
SGX_PG_INVLD		Target page is not a valid EPC page.
SGX_PG_NONEPC		Page is not an EPC page.

Concurrency Restrictions

Table 38-40. Base Concurrency Restrictions of ERDINFO

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ERDINFO	Target [DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	

Table 38-41. Additional Concurrency Restrictions of ERDINFO

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ERDINFO	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ERDINFO Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_RDINFO	Linear Address	64	Address of the RDINFO structure.

(* check alignment of RDINFO structure (RBX) *)
 IF (DS:RBX is not 32Byte Aligned) THEN
 #GP(0); FI;

(* check alignment of the EPCPAGE (RCX) *)
 IF (DS:RCX is not 4KByte Aligned) THEN
 #GP(0); FI;

(* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
 IF (DS:RCX does not resolve within EPC) THEN
 RFLAGS.CF := 1;
 RFLAGS.ZF := 0;
 RAX := SGX_PG_NONEPC;
 goto DONE;
 FI;

(* Check the EPC page for concurrency *)
 IF (EPC page is being modified) THEN
 RFLAGS.ZF = 1;
 RFLAGS.CF = 0;
 RAX = SGX_EPC_PAGE_CONFLICT;
 goto DONE;
 FI;

(* check page validity *)
 IF (EPCM(DS:RCX).VALID = 0) THEN
 RFLAGS.CF = 1;

```

RFLAGS.ZF = 0;
RAX = SGX_PG_INVLD;
goto DONE;
FI;

(* clear the fields of the RDINFO structure *)
TMP_RDINFO := DS:RBX;
TMP_RDINFO.STATUS := 0;
TMP_RDINFO.FLAGS := 0;
TMP_RDINFO.ENCLAVECONTEXT := 0;

(* store page info in RDINFO structure *)
TMP_RDINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
TMP_RDINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
TMP_RDINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
TMP_RDINFO.FLAGS.PR := EPCM(DS:RCX).PR;
TMP_RDINFO.FLAGS.PAGE_TYPE := EPCM(DS:RCX).PAGE_TYPE;
TMP_RDINFO.FLAGS.BLOCKED := EPCM(DS:RCX).BLOCKED;

(* read SECS.ENCLAVECONTEXT for enclave child pages *)
IF ((EPCM(DS:RCX).PAGE_TYPE = PT_REG) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_TCS) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_TRIM) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_SS_FIRST) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_SS_REST)
    ) THEN
    TMP_SECS := Address of SECS for (DS:RCX);
    TMP_RDINFO.ENCLAVECONTEXT := SECS(TMP_SECS).ENCLAVECONTEXT;
FI;

(* populate enclave information for SECS pages *)
IF (EPCM(DS:RCX).PAGE_TYPE = PT_SECS) THEN
    IF ((VMX non-root mode) and
        (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)
        ) THEN
        TMP_RDINFO.STATUS.CHILDPRESENT :=
            ((SECS(DS:RCX).CHLDCNT ≠ 0) or
             SECS(DS:RCX).VIRTCHILDCNT ≠ 0);
    ELSE
        TMP_RDINFO.STATUS.CHILDPRESENT := (SECS(DS:RCX).CHLDCNT ≠ 0);
        TMP_RDINFO.STATUS.VIRTCHILDPRESENT :=
            (SECS(DS:RCX).VIRTCHILDCNT ≠ 0);
        TMP_RDINFO.ENCLAVECONTEXT := SECS(DS:RCX).ENCLAVECONTEXT;
    FI;
FI;

RAX := 0;
RFLAGS.ZF := 0;
RFLAGS.CF := 0;

DONE:
(* clear flags *)
RFLAGS.PF := 0;
RFLAGS.AF := 0;

```

RFLAGS.OF := 0;
RFLAGS.SF := 70;

Flags Affected

ZF is set if ERDINFO fails due to concurrent operation with another SGX instruction; otherwise cleared.

CF is set if page is not a valid EPC page or not an EPC page; otherwise cleared.

PF, AF, OF, and SF are cleared.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
 If DS segment is unusable.

 If a memory operand is not properly aligned.

#PF(error code) If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0) If the memory address is in a non-canonical form.

 If a memory operand is not properly aligned.

#PF(error code) If a page fault occurs in accessing memory operands.

EREMOVE—Remove a page from the EPC

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLS[EREMOVE]	IR	V/V	SGX1	This leaf function removes a page from the EPC.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	EREMOVE (In)	Return error code (Out)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be un-associated with its SECS and be marked as unused. This instruction leaf can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

The instruction fails if the operand is not properly aligned or does not refer to an EPC page or the page is in use by another thread, or other threads are running in the enclave to which the page belongs. In addition the instruction fails if the operand refers to an SECS with associations.

EREMOVE Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

The instruction faults if any of the following:

EREMOVE Faulting Conditions

The memory operand is not properly aligned.	The memory operand does not resolve in an EPC page.
Refers to an invalid SECS.	Refers to an EPC page that is locked by another thread.
Another Intel SGX instruction is accessing the EPC page.	RCX does not contain an effective address of an EPC page.
the EPC page refers to an SECS with associations.	

The error codes are:

Table 38-42. EREMOVE Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EREMOVE successful.
SGX_CHILD_PRESENT	If the SECS still have enclave pages loaded into EPC.
SGX_ENCLAVE_ACT	If there are still logical processors executing inside the enclave.

Concurrency Restrictions

Table 38-43. Base Concurrency Restrictions of EREMOVE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EREMOVE	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 38-44. Additional Concurrency Restrictions of EREMOVE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREMOVE	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREMOVE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.

IF (DS:RCX is not 4KByte Aligned)
 THEN #GP(0); FI;

IF (DS:RCX does not resolve to an EPC page)
 THEN #PF(DS:RCX); FI;

TMP_SECS := Get_SECS_ADDRESS();

(* Check the EPC page for concurrency *)

IF (EPC page being referenced by another Intel SGX instruction)
 THEN
 IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
 THEN
 VMCS.Exit_reason := SGX_CONFLICT;
 VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
 VMCS.Exit_qualification.error := 0;
 VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
 VMCS.Guest-linear_address := DS:RCX;
 Deliver VMEXIT;
 ELSE
 #GP(0);
 FI;

FI;

(* if DS:RCX is already unused, nothing to do*)

IF ((EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PT = PT_TRIM AND EPCM(DS:RCX).MODIFIED = 0))
 THEN GOTO DONE;

FI;

```

IF ( (EPCM(DS:RCX).PT = PT_VA) OR
      ((EPCM(DS:RCX).PT = PT_TRIM) AND (EPCM(DS:RCX).MODIFIED = 0)) )
  THEN
    EPCM(DS:RCX).VALID := 0;
    GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT = PT_SECS)
  THEN
    IF (DS:RCX has an EPC page associated with it)
      THEN
        RFLAGS.ZF := 1;
        RAX := SGX_CHILD_PRESENT;
        GOTO ERROR_EXIT;
      FI;
    (* treat SECS as having a child page when VIRTCHILDCNT is non-zero *)
    IF (<<in VMX non-root operation>> AND
        <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>> AND
        (SECS(DS:RCX).VIRTCHILDCNT ≠ 0))
      THEN
        RFLAGS.ZF := 1;
        RAX := SGX_CHILD_PRESENT;
        GOTO ERROR_EXIT;
      FI;
    EPCM(DS:RCX).VALID := 0;
    GOTO DONE;
FI;

IF (Other threads active using SECS)
  THEN
    RFLAGS.ZF := 1;
    RAX := SGX_ENCLAVE_ACT;
    GOTO ERROR_EXIT;
FI;

IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM) or
      (EPCM(DS:RCX).PT is PT_SS_FIRST) or (EPCM(DS:RCX).PT is PT_SS_REST))
  THEN
    EPCM(DS:RCX).VALID := 0;
    GOTO DONE;
FI;

DONE:
RAX := 0;
RFLAGS.ZF := 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;

```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

- #GP(0) If a memory operand effective address is outside the DS segment limit.
 If a memory operand is not properly aligned.
 If another Intel SGX instruction is accessing the page.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If the memory operand is not an EPC page.

64-Bit Mode Exceptions

- #GP(0) If the memory operand is non-canonical form.
 If a memory operand is not properly aligned.
 If another Intel SGX instruction is accessing the page.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If the memory operand is not an EPC page.

ETRAK—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0CH ENCLS[ETRAK]	IR	V/V	SGX1	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	ETRAK (In)	Return error code (Out)	Pointer to the SECS of the EPC page (In)

Description

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

The table below provides additional information on the memory parameter of ETRAK leaf function.

ETRAK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 38-45. ETRAK Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	ETRAK successful.
SGX_PREV_TRK_INCMPL	All processors did not complete the previous shoot-down sequence.

Concurrency Restrictions

Table 38-46. Base Concurrency Restrictions of ETRAK

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ETRAK	SECS [DS:RCX]	Shared	#GP	

Table 38-47. Additional Concurrency Restrictions of ETRAK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRAK, ETRAKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ETRAK	SECS [DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT

Operation

IF (DS:RCX is not 4KByte Aligned)
THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)

IF (Other Intel SGX instructions using tracking facility on this SECS)
THEN
IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
THEN
VMCS.Exit_reason := SGX_CONFLICT;
VMCS.Exit_qualification.code := TRACKING_RESOURCE_CONFLICT;
VMCS.Exit_qualification.error := 0;
VMCS.Guest-physical_address := SECS(TMP_SECS).ENCLAVECONTEXT;
VMCS.Guest-linear_address := 0;
Deliver VMEXIT;
ELSE
#GP(0);
FI;

FI;

IF (EPCM(DS:RCX).VALID = 0)
THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).PT ≠ PT_SECS)
THEN #PF(DS:RCX); FI;

(* All processors must have completed the previous tracking cycle*)

IF ((DS:RCX).TRACKING ≠ 0)
THEN
IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
THEN
VMCS.Exit_reason := SGX_CONFLICT;
VMCS.Exit_qualification.code := TRACKING_REFERENCE_CONFLICT;
VMCS.Exit_qualification.error := 0;
VMCS.Guest-physical_address := SECS(TMP_SECS).ENCLAVECONTEXT;
VMCS.Guest-linear_address := 0;
Deliver VMEXIT;
FI;
RFLAGS.ZF := 1;
RAX := SGX_PREV_TRK_INCMPL;
GOTO DONE;
ELSE
RAX := 0;
RFLAGS.ZF := 0;
FI;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

- #GP(0) If a memory operand effective address is outside the DS segment limit.
 If a memory operand is not properly aligned.
 If another thread is concurrently using the tracking facility on this SECS.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If a memory operand is not an EPC page.

64-Bit Mode Exceptions

- #GP(0) If a memory operand is non-canonical form.
 If a memory operand is not properly aligned.
 If the specified EPC resource is in use.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If a memory operand is not an EPC page.

ETRACKC—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 11H ENCLS[ETRACKC]	IR	V/V	EAX[6]	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX	
IR	ETRACK (In)	Return error code (Out)	Address of the destination EPC page (In, EA)	Address of the SECS page (In, EA)

Description

The ETRACKC instruction is thread safe variant of ETRACK leaf and can be executed concurrently with other CPU threads operating on the same SECS.

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

The table below provides additional information on the memory parameter of ETRACK leaf function.

ETRACKC Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 38-48. ETRACKC Return Value in RAX

Error Code	Value	Description
No Error	0	ETRACKC successful.
SGX_EPC_PAGE_CONFLICT	7	Failure due to concurrent operation of another SGX instruction.
SGX_PG_INVLD	6	Target page is not a VALID EPC page.
SGX_PREV_TRK_INCMPL	17	All processors did not complete the previous tracking sequence.
SGX_TRACK_NOT_REQUIRED	27	Target page type does not require tracking.

Concurrency Restrictions

Table 38-49. Base Concurrency Restrictions of ETRACKC

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ETRACKC	Target [DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS implicit	Concurrent		

Table 38-50. Additional Concurrency Restrictions of ETRACKC

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ETRACKC	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS implicit	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT

Operation

Temp Variables in ETRACKC Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.

```
(* check alignment of EPCPAGE (RCX) *)
IF (DS:RCX is not 4KByte Aligned) THEN
#GP(0); FI;
```

```
(* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
#PF(DS:RCX, PFEC.SGX); FI;
```

```
(* Check the EPC page for concurrency *)
IF (EPC page is being modified) THEN
RFLAGS.ZF := 1;
RFLAGS.CF := 0;
RAX := SGX_EPC_PAGE_CONFLICT;
goto DONE_POST_LOCK_RELEASE;
FI;
```

```
(* check to make sure the page is valid *)
IF (EPCM(DS:RCX).VALID = 0) THEN
RFLAGS.ZF := 1;
RFLAGS.CF := 0;
RAX := SGX_PG_INVLD;
GOTO DONE;
FI;
```

```
(* find out the target SECS page *)
IF (EPCM(DS:RCX).PT is PT_REG or PT_TCS or PT_TRIM or PT_SS_FIRST or PT_SS_REST) THEN
TMP_SECS := Obtain SECS through EPCM(DS:RCX).ENCLAVESECS;
ELSE IF (EPCM(DS:RCX).PT is PT_SECS) THEN
TMP_SECS := Obtain SECS through (DS:RCX);
ELSE
RFLAGS.ZF := 0;
RFLAGS.CF := 1;
RAX := SGX_TRACK_NOT_REQUIRED;
GOTO DONE;
FI;
```

```

(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions using tracking facility on this SECS) THEN
  IF ((VMX non-root mode) and
    (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)) THEN
    VMCS.Exit_reason := SGX_CONFLICT;
    VMCS.Exit_qualification.code := TRACKING_RESOURCE_CONFLICT;
    VMCS.Exit_qualification.error := 0;
    VMCS.Guest-physical_address :=
      SECS(TMP_SECS).ENCLAVECONTEXT;
    VMCS.Guest-linear_address := 0;
    Deliver VMEXIT;
  FI;

  RFLAGS.ZF := 1;
  RFLAGS.CF := 0;
  RAX := SGX_EPC_PAGE_CONFLICT;
  GOTO DONE;
FI;

(* All processors must have completed the previous tracking cycle*)
IF ((TMP_SECS).TRACKING ≠ 0) )
THEN
  IF ((VMX non-root mode) and
    (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)) THEN
    VMCS.Exit_reason := SGX_CONFLICT;
    VMCS.Exit_qualification.code := TRACKING_REFERENCE_CONFLICT;
    VMCS.Exit_qualification.error := 0;
    VMCS.Guest-physical_address :=
      SECS(TMP_SECS).ENCLAVECONTEXT;
    VMCS.Guest-linear_address := 0;
    Deliver VMEXIT;
  FI;

  RFLAGS.ZF := 1;
  RFLAGS.CF := 0;
  RAX := SGX_PREV_TRK_INCMPL;
  GOTO DONE;
FI;

RFLAGS.ZF := 0;
RFLAGS.CF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.PF,AF,OF,SF := 0;

```

Flags Affected

ZF is set if ETRACKC fails due to concurrent operations with another SGX instructions or target page is an invalid EPC page or tracking is not completed on SECS page; otherwise cleared.

CF is set if target page is not of a type that requires tracking; otherwise cleared.

PF, AF, OF, and SF are cleared.

Protected Mode Exceptions

- #GP(0) If the memory operand violates access-control policies of DS segment.
 If DS segment is unusable.
- #PF(error code) If the memory operand is not properly aligned.
 If the memory operand expected to be in EPC does not resolve to an EPC page.
 If a page fault occurs in access memory operand.

64-Bit Mode Exceptions

- #GP(0) If a memory address is in a non-canonical form.
 If a memory operand is not properly aligned.
- #PF(error code) If the memory operand expected to be in EPC does not resolve to an EPC page.
 If a page fault occurs in access memory operand.

EWB—Invalidate an EPC Page and Write out to Main Memory

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0BH ENCLS[EWB]	IR	V/V	SGX1	This leaf function invalidates an EPC page and writes it out to main memory.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EWB (In)	Error code (Out)	Address of an PAGEINFO (In)	Address of the EPC page (In)	Address of a VA slot (In)

Description

This leaf function copies a page from the EPC to regular main memory. As part of the copying process, the page is cryptographically protected. This instruction can only be executed when current privilege level is 0.

The table below provides additional information on the memory parameter of EPA leaf function.

EWB Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	EPCPAGE	VASLOT
Non-EPC R/W access	Non-EPC R/W access	Non-EPC R/W access	EPC R/W access	EPC R/W access

The error codes are:

Table 38-51. EWB Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EWB successful.
SGX_PAGE_NOT_BLOCKED	If page is not marked as blocked.
SGX_NOT_TRACKED	If EWB is racing with ETRACK instruction.
SGX_VA_SLOT_OCCUPIED	Version array slot contained valid entry.
SGX_CHILD_PRESENT	Child page present while attempting to page out enclave.

Concurrency Restrictions

Table 38-52. Base Concurrency Restrictions of EWB

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EWB	Source [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA [DS:RDX]	Shared	#GP	

Table 38-53. Additional Concurrency Restrictions of EWB

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EWB	Source [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Exclusive	

Operation

Temp Variables in EWB Operational Flow

Name	Type	Size (Bytes)	Description
TMP_SRCPGE	Memory page	4096	
TMP_PCMD	PCMD	128	
TMP_SECS	SECS	4096	
TMP_BPEPOCH	UINT64	8	
TMP_BPREFCOUNT	UINT64	8	
TMP_HEADER	MAC Header	128	
TMP_PCMD_ENCLAVEID	UINT64	8	
TMP_VER	UINT64	8	
TMP_PK	UINT128	16	

```
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
  THEN #GP(0); FI;
```

```
IF (DS:RCX does not resolve within an EPC)
  THEN #PF(DS:RCX); FI;
```

```
IF (DS:RDX is not 8Byte Aligned)
  THEN #GP(0); FI;
```

```
IF (DS:RDX does not resolve within an EPC)
  THEN #PF(DS:RDX); FI;
```

```
(* EPCPAGE and VASLOT should not resolve to the same EPC page*)
IF (DS:RCX and DS:RDX resolve to the same EPC page)
  THEN #GP(0); FI;
```

```
TMP_SRCPGE := DS:RBX.SRCPGE;
(* Note PAGEINFO.PCMD is overlaid on top of PAGEINFO.SECINFO *)
TMP_PCMD := DS:RBX.PCMD;
```

```
If (DS:RBX.LINADDR ≠ 0) OR (DS:RBX.SECS ≠ 0)
  THEN #GP(0); FI;
```

```
IF ( (DS:TMP_PCMD is not 128Byte Aligned) or (DS:TMP_SRCPGE is not 4KByte Aligned) )
  THEN #GP(0); FI;
```

```
(* Check for concurrent Intel SGX instruction access to the page *)
IF (Other Intel SGX instruction is accessing page)
  THEN
    IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
      THEN
        VMCS.Exit_reason := SGX_CONFLICT;
        VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
```

```

        VMCS.Guest-linear_address := DS:RCX;
    Deliver VMEXIT;
    ELSE
        #GP(0);
    FI;
FI;

(*Check if the VA Page is being removed or changed*)
IF (VA Page is being modified)
    THEN #GP(0); FI;

(* Verify that EPCPAGE and VASLOT page are valid EPC pages and DS:RDX is VA *)
IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~OFFFH).VALID = 0) or (EPCM(DS:RDX & ~FFFH).PT is not PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Perform page-type-specific exception checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM) or
    (EPCM(DS:RCX).PT is PT_SS_FIRST) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        TMP_SECS = Obtain SECS through EPCM(DS:RCX)
        (* Check that EBLOCK has occurred correctly *)
        IF (EBLOCK is not correct)
            THEN #GP(0); FI;
    FI;

RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

(* Zero out TMP_HEADER*)
TMP_HEADER[ sizeof(TMP_HEADER) - 1 : 0 ] := 0;

(* Perform page-type-specific checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM) or
    (EPCM(DS:RCX).PT is PT_SS_FIRST) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        (* check to see if the page is evictable *)
        IF (EPCM(DS:RCX).BLOCKED = 0)
            THEN
                RAX := SGX_PAGE NOT_BLOCKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
            FI;
        (* Check if tracking done correctly *)
        IF (Tracking not correct)
            THEN
                RAX := SGX_NOT_TRACKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
            FI;

        (* Obtain EID to establish cryptographic binding between the paged-out page and the enclave *)

```

```
TMP_HEADER.EID := TMP_SECS.EID;
```

```
(* Obtain EID as an enclave handle for software *)
```

```
TMP_PCMD_ENCLAVEID := TMP_SECS.EID;
```

```
ELSE IF (EPCM(DS:RCX).PT is PT_SECS)
```

```
(*check that there are no child pages inside the enclave *)
```

```
IF (DS:RCX has an EPC page associated with it)
```

```
THEN
```

```
    RAX := SGX_CHILD_PRESENT;
```

```
    RFLAGS.ZF := 1;
```

```
    GOTO ERROR_EXIT;
```

```
FI;
```

```
(* treat SECS as having a child page when VIRTCHILDCNT is non-zero *)
```

```
IF (<<in VMX non-root operation>> AND
```

```
<<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>> AND
```

```
(SECS(DS:RCX).VIRTCHILDCNT ≠ 0))
```

```
THEN
```

```
    RFLAGS.ZF := 1;
```

```
    RAX := SGX_CHILD_PRESENT;
```

```
    GOTO ERROR_EXIT;
```

```
FI;
```

```
TMP_HEADER.EID := 0;
```

```
(* Obtain EID as an enclave handle for software *)
```

```
TMP_PCMD_ENCLAVEID := (DS:RCX).EID;
```

```
ELSE IF (EPCM(DS:RCX).PT is PT_VA)
```

```
TMP_HEADER.EID := 0; // Zero is not a special value
```

```
(* No enclave handle for VA pages*)
```

```
TMP_PCMD_ENCLAVEID := 0;
```

```
FI;
```

```
TMP_HEADER.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;
```

```
TMP_HEADER.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
```

```
TMP_HEADER.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
```

```
TMP_HEADER.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
```

```
TMP_HEADER.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
```

```
TMP_HEADER.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;
```

```
(* Encrypt the page, DS:RCX could be encrypted in place. AES-GCM produces 2 values, {ciphertext, MAC}. *)
```

```
(* AES-GCM input parameters: key, GCM Counter, MAC_HDR, MAC_HDR_SIZE, SRC, SRC_SIZE*)
```

```
{DS:TMP_SRCPGE, DS:TMP_PCMD.MAC} := AES_GCM_ENC(CR_BASE_PK), (TMP_VER << 32),
```

```
    TMP_HEADER, 128, DS:RCX, 4096);
```

```
(* Write the output *)
```

```
Zero out DS:TMP_PCMD.SECINFO
```

```
DS:TMP_PCMD.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
```

```
DS:TMP_PCMD.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
```

```
DS:TMP_PCMD.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
```

```
DS:TMP_PCMD.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
```

```
DS:TMP_PCMD.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;
```

```
DS:TMP_PCMD.RESERVED := 0;
```

```
DS:TMP_PCMD.ENCLAVEID := TMP_PCMD_ENCLAVEID;
```

```
DS:RBX.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;
```

```
(*Check if version array slot was empty *)
```

```

IF ([DS.RDX])
  THEN
    RAX := SGX_VA_SLOT_OCCUPIED
    RFLAGS.CF := 1;

```

```

FI;

```

(* Write version to Version Array slot *)

```

[DS.RDX] := TMP_VER;

```

(* Free up EPCM Entry *)

```

EPCM.(DS:RCX).VALID := 0;

```

```

ERROR_EXIT:

```

Flags Affected

ZF is set if page is not blocked, not tracked, or a child is present. Otherwise cleared.

CF is set if VA slot is previously occupied, Otherwise cleared.

Protected Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If the EPC page and VASLOT resolve to the same EPC page. If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages. If the tracking resource is in use. If the EPC page or the version array page is invalid. If the parameters fail consistency checks.
#PF(error code)	<ul style="list-style-type: none"> If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page. If one of the EPC memory operands has incorrect page type.

64-Bit Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If a memory operand is non-canonical form. If a memory operand is not properly aligned. If the EPC page and VASLOT resolve to the same EPC page. If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages. If the tracking resource is in use. If the EPC page or the version array page in invalid. If the parameters fail consistency checks.
#PF(error code)	<ul style="list-style-type: none"> If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page. If one of the EPC memory operands has incorrect page type.

38.4 INTEL® SGX USER LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLU instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of the implicitly-encoded register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EACCEPT—Accept Changes to an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLU[EACCEPT]	IR	V/V	SGX2	This leaf function accepts changes made by system software to an EPC page in the running enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EACCEPT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function accepts changes to a page in the running enclave by verifying that the security attributes specified in the SECINFO match the security attributes of the page in the EPCM. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPT leaf function.

EACCEPT Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPT Faulting Conditions

The operands are not properly aligned.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	Page type is PT_REG and MODIFIED bit is 0.
SECINFO contains an invalid request.	Page type is PT_TCS or PT_TRIM and PENDING bit is 0 and MODIFIED bit is 1.
If security attributes of the SECINFO page make the page inaccessible.	

The error codes are:

Table 38-54. EACCEPT Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EACCEPT successful.
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values.
SGX_NOT_TRACKED	The OS did not complete an ETRACK on the target page.

Concurrency Restrictions

Table 38-55. Base Concurrency Restrictions of EACCEPT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPT	Target [DS:RCX]	Shared	#GP	
	SECINFO [DS:RBX]	Concurrent		

Table 38-56. Additional Concurrency Restrictions of EACCEPT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPT	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EACCEPT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operands belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

IF (DS:RBX is not 64Byte Aligned)
THEN #GP(0); FI;

IF (DS:RBX is not within CR_ELRANGE)
THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
(EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or
(EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
(EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ (DS:RBX & FFFH)))
THEN #PF(DS:RBX); FI;

(* Copy 64 bytes of contents *)
SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero)
THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
THEN #GP(0); FI;


```
IF (DS:RCX is not within CR_ELRANGE)
    THEN #GP(0); FI;
```

```
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;
```

(* Check that the combination of requested PT, PENDING, and MODIFIED is legal *)

```
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 0)
    THEN
        IF (NOT (((SCRATCH_SECINFO.FLAGS.PT is PT_REG) and
            ((SCRATCH_SECINFO.FLAGS.PR is 1) or
            (SCRATCH_SECINFO.FLAGS.PENDING is 1)) and
            (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) or
            ((SCRATCH_SECINFO.FLAGS.PT is PT_TCS or PT_TRIM) and
            (SCRATCH_SECINFO.FLAGS.PR is 0) and
            (SCRATCH_SECINFO.FLAGS.PENDING is 0) and
            (SCRATCH_SECINFO.FLAGS.MODIFIED is 1) )))
            THEN #GP(0); FI
        ELSE
            IF (NOT (((SCRATCH_SECINFO.FLAGS.PT is PT_REG) AND
            ((SCRATCH_SECINFO.FLAGS.PR is 1) OR
            (SCRATCH_SECINFO.FLAGS.PENDING is 1)) AND
            (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) OR
            ((SCRATCH_SECINFO.FLAGS.PT is PT_TCS OR PT_TRIM) AND
            (SCRATCH_SECINFO.FLAGS.PENDING is 0) AND
            (SCRATCH_SECINFO.FLAGS.MODIFIED is 1) AND
            (SCRATCH_SECINFO.FLAGS.PR is 0)) OR
            ((SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST or PT_SS_REST) AND
            (SCRATCH_SECINFO.FLAGS.PENDING is 1) AND
            (SCRATCH_SECINFO.FLAGS.MODIFIED is 0) AND
            (SCRATCH_SECINFO.FLAGS.PR is 0))))
                THEN #GP(0); FI;
            FI;
```

(* Check security attributes of the destination EPC page *)

```
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).BLOCKED is not 0) or
    ((EPCM(DS:RCX).PT is not PT_REG) and (EPCM(DS:RCX).PT is not PT_TCS) and (EPCM(DS:RCX).PT is not PT_TRIM)
    and (EPCM(DS:RCX).PT is not PT_SS_FIRST) and (EPCM(DS:RCX).PT is not PT_SS_REST)) or
    (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS))
    THEN #PF(DS:RCX); FI;
```

(* Check the destination EPC page for concurrency *)

```
IF ( EPC page in use )
    THEN #GP(0); FI;
```

(* Re-Check security attributes of the destination EPC page *)

```
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN #PF(DS:RCX); FI;
```

(* Verify that accept request matches current EPC page settings *)

```
IF ( (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX) or (EPCM(DS:RCX).PENDING ≠ SCRATCH_SECINFO.FLAGS.PENDING) or
    (EPCM(DS:RCX).MODIFIED ≠ SCRATCH_SECINFO.FLAGS.MODIFIED) or (EPCM(DS:RCX).R ≠ SCRATCH_SECINFO.FLAGS.R) or
    (EPCM(DS:RCX).W ≠ SCRATCH_SECINFO.FLAGS.W) or (EPCM(DS:RCX).X ≠ SCRATCH_SECINFO.FLAGS.X) or
    (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) )
```

```

THEN
    RFLAGS.ZF := 1;
    RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
    GOTO DONE;
FI;
(* Check that all required threads have left enclave *)
IF (Tracking not correct)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_NOT_TRACKED;
        GOTO DONE;
FI;

(* Get pointer to the SECS to which the EPC page belongs *)
TMP_SECS = << Obtain physical address of SECS through EPCM(DS:RCX)>>
(* For TCS pages, perform additional checks *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;

        (* Check that TCS.FLAGS.DBGOPTIN, TCS stack, and TCS status are correctly initialized *)
        (* check that TCS.PREVSSP is 0 *)
        IF ( ((DS:RCX).FLAGS.DBGOPTIN is not 0) or ((DS:RCX).CSSA ≥ (DS:RCX).NSSA) or ((DS:RCX).AEP is not 0) or ((DS:RCX).STATE is not 0)
        or ((CPUID.(EAX=07H, ECX=0H):ECX[CET_SS] = 1) AND ((DS:RCX).PREVSSP != 0)))
            THEN #GP(0); FI;

        (* Check consistency of FS & GS Limit *)
        IF ( (TMP_SECS.ATTRIBUTES.MODE64BIT is 0) and ((DS:RCX.FSLIMIT & FFFH ≠ FFFH) or (DS:RCX.GSLIMIT & FFFH ≠ FFFH)) )
            THEN #GP(0); FI;
FI;

(* Clear PENDING/MODIFIED flags to mark accept operation complete *)
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;

(* Clear EAX and ZF to indicate successful completion *)
RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;

```

Flags Affected

Sets ZF if page cannot be accepted, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

- #GP(0) If executed outside an enclave.
 If a memory operand effective address is outside the DS segment limit.
 If a memory operand is not properly aligned.
 If a memory operand is locked.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If a memory operand is not an EPC page.
 If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

- #GP(0) If executed outside an enclave.
 If a memory operand is non-canonical form.
 If a memory operand is not properly aligned.
 If a memory operand is locked.
- #PF(error code) If a page fault occurs in accessing memory operands.
 If a memory operand is not an EPC page.
 If EPC page has incorrect page type or security attributes.

EACCEPTCOPY—Initialize a Pending Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLU[EACCEPTCOPY]	IR	V/V	SGX2	This leaf function initializes a dynamically allocated EPC page from another page in the EPC.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EACCEPTCOPY (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)	Address of the source EPC page (In)

Description

This leaf function copies the contents of an existing EPC page into an uninitialized EPC page (created by EAUG). After initialization, the instruction may also modify the access rights associated with the destination EPC page. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX and RDX each contain the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPTCOPY leaf function.

EACCEPTCOPY Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)	EPCPAGE (Source)
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPTCOPY Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	If security attributes of the source EPC page make the page inaccessible.
The EPC page is not valid.	RBX does not contain an effective address in an EPC page in the running enclave.
SECINFO contains an invalid request.	RCX/RDX does not contain an effective address of an EPC page in the running enclave.

The error codes are:

Table 38-57. EACCEPTCOPY Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EACCEPTCOPY successful.
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values.

Concurrency Restrictions

Table 38-58. Base Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPTCOPY	Target [DS:RCX]	Concurrent		
	Source [DS:RDX]	Concurrent		
	SECINFO [DS:RBX]	Concurrent		

Table 38-59. Additional Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPTCOPY	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	Source [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EACCEPTCOPY Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

IF (DS:RBX is not 64Byte Aligned)
 THEN #GP(0); FI;

IF ((DS:RCX is not 4KByte Aligned) or (DS:RDX is not 4KByte Aligned))
 THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) or (DS:RDX is not within CR_ELRANGE))
 THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
 THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
 THEN #PF(DS:RCX); FI;

IF (DS:RDX does not resolve within an EPC)
 THEN #PF(DS:RDX); FI;

IF ((EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
 (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or (EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or
 (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
 (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ DS:RBX))
 THEN #PF(DS:RBX); FI;

```
(* Copy 64 bytes of contents *)
SCRATCH_SECINFO := DS:RBX;
```

```
(* Check for misconfigured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or (SCRATCH_SECINFO.FLAGS.R=0) AND(SCRATCH_SECINFO.FLAGS.W≠0) or
  (SCRATCH_SECINFO.FLAGS.PT is not PT_REG) )
  THEN #GP(0); FI;
```

```
(* Check security attributes of the source EPC page *)
IF ( (EPCM(DS:RDX).VALID = 0) or (EPCM(DS:RCX).R = 0) or (EPCM(DS:RDX).PENDING ≠ 0) or (EPCM(DS:RDX).MODIFIED ≠ 0) or
  (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
  (EPCM(DS:RDX).ENCLAVEADDRESS ≠ DS:RDX))
  THEN #PF(DS:RDX); FI;
```

```
(* Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
  (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
  THEN
    RFLAGS.ZF := 1;
    RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
    GOTO DONE;
FI;
```

```
(* Check the destination EPC page for concurrency *)
IF (destination EPC page in use )
  THEN #GP(0); FI;
```

```
(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
  (EPCM(DS:RCX).R ≠ 1) or (EPCM(DS:RCX).W ≠ 1) or (EPCM(DS:RCX).X ≠ 0) or
  (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
  (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
  THEN
    RFLAGS.ZF := 1;
    RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
    GOTO DONE;
FI;
```

```
(* Copy 4Kbytes form the source to destination EPC page*)
DS:RCX[32767:0] := DS:RDX[32767:0];
```

```
(* Update EPCM permissions *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PENDING := 0;
```

```
RFLAGS.ZF := 0;
RAX := 0;
```

```
DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page is not modifiable, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

- #GP(0) If executed outside an enclave.
If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If a memory operand is locked.
- #PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

- #GP(0) If executed outside an enclave.
If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If a memory operand is locked.
- #PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If EPC page has incorrect page type or security attributes.

EDECCSSA—Decrements TCS.CSSA

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLU[EDECCSSA]	IR	V/V	EDECCSSA	This leaf function decrements TCS.CSSA.

Instruction Operand Encoding

Op/En	EAX
IR	EDECCSSA (In)

Description

This leaf function changes the current SSA frame by decrementing TCS.CSSA for the current enclave thread. If the enclave has enabled CET shadow stacks or indirect branch tracking, then EDECCSSA also changes the current CET state save frame. This instruction leaf can only be executed inside an enclave.

EDECCSSA Memory Parameter Semantics

TCS
Read/Write access by Enclave

The instruction faults if any of the following:

EDECCSSA Faulting Conditions

TCS.CSSA is 0.	TCS is not valid or available or locked.
The SSA frame is not valid or in use.	

Concurrency Restrictions**Table 38-60. Base Concurrency Restrictions of EDECCSSA**

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDECCSSA	TCS [CR_TCS_PA]	Shared	#GP	

Table 38-61. Additional Concurrency Restrictions of EDECCSSA

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDECCSSA	TCS [CR_TCS_PA]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDECCSSA Operational Flow

Name	Type	Size (bits)	Description
TMP_SSA	Effective Address	32/64	Address of current SSA frame.
TMP_XSIZE	Integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the target frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the target SSA frame.
TMP_XSAVE_PAGE_PA_n	Physical Address	32/64	Physical address of the nth page within the target SSA frame.
TMP_CET_SAVE_AREA	Effective Address	32/64	Address of the current CET save area.
TMP_CET_SAVE_PAGE	Effective Address	32/64	Address of the current CET save area page.

```
IF (CR_TCS_PA.CSSA = 0)
  THEN #GP(0); FI;
```

```
(* Compute linear address of SSA frame *)
```

```
TMP_SSA := CR_TCS_PA.OSSA + CR_ACTIVE_SECS.BASEADDR + 4096 * CR_ACTIVE_SECS.SSAFRAMESIZE * (CR_TCS_PA.CSSA - 1);
```

```
TMP_XSIZE := compute_XSAVE_frame_size(CR_ACTIVE_SECS.ATTRIBUTES.XFRM);
```

```
FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
```

```
  (* Check page is read/write accessible *)
```

```
  Check that DS:TMP_SSA_PAGE is read/write accessible;
```

```
  If a fault occurs, release locks, abort and deliver that fault;
```

```
  IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
  IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
  IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
  IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
  IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMP_SSA_PAGE) or
```

```
  (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
```

```
  (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(CR_TCS_PA).ENCLAVESECS) or
```

```
  (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0))
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
  TMP_XSAVE_PAGE_PA_n := Physical_Address(DS:TMP_SSA_PAGE);
```

```
ENDFOR
```

```
(* Compute address of GPR area*)
```

```
TMP_GPR := TMP_SSA + 4096 * CR_ACTIVE_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
```

```
Check that DS:TMP_SSA_PAGE is read/write accessible;
```

```
If a fault occurs, release locks, abort and deliver that fault;
```

```
IF (DS:TMP_GPR does not resolve to EPC page)
```

```
  THEN #PF(DS:TMP_GPR); FI;
```

```

IF (EPCM(DS:TMP_GPR).VALID = 0)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or
    (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS ≠ EPCM(CR_TCS_PA).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
    THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)
    THEN
        IF (TMP_GPR + (sizeof(GPRSGX_AREA) - 1) is not in DS segment)
            THEN #GP(0); FI;
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        IF ((CR_ACTIVE_SECS.CET_ATTRIBUTES.SH_STK_EN == 1) OR (CR_ACTIVE_SECS.CET_ATTRIBUTES.ENDBR_EN == 1))
            THEN
                (* Compute linear address of what will become new CET state save area and cache its PA *)
                TMP_CET_SAVE_AREA := CR_TCS_PA.OCETSSA + CR_ACTIVE_SECS.BASEADDR + (CR_TCS_PA.CSSA - 1) * 16;
                TMP_CET_SAVE_PAGE := TMP_CET_SAVE_AREA & ~0xFFF;
                Check the TMP_CET_SAVE_PAGE page is read/write accessible
                If fault occurs release locks, abort and deliver fault

                (* read the EPCM VALID, PENDING, MODIFIED, BLOCKED and PT fields atomically *)
                IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR
                    (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(CR_TCS_PA).ENCLAVESECS))
                    THEN #PF(DS:TMP_CET_SAVE_PAGE); FI;
            FI;
        FI;

(* At this point, the instruction is guaranteed to complete *)
CR_TCS_PA.CSSA := CR_TCS_PA.CSSA - 1;

CR_GPR_PA := Physical_Address(DS:TMP_GPR);

FOR EACH TMP_XSAVE_PAGE_n
    CR_XSAVE_PAGE_n := TMP_XSAVE_PAGE_PA_n;
ENDFOR

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN

```

```

IF ((TMP_SECS.CET_ATTRIBUTES.SH_STK_EN == 1) OR
(TMP_SECS.CET_ATTRIBUTES.ENDBR_EN == 1))
  THEN
    CR_CET_SAVE_AREA_PA := Physical_Address(DS:TMP_CET_SAVE_AREA);
  FI;
FI;

```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave. If CR_TCS_PA.CSSA = 0.
#PF(error code)	If a page fault occurs in accessing memory. If one or more pages of the target SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page. If CET is enabled for the enclave and the target CET SSA frame is not readable/writable, or does not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave. If CR_TCS_PA.CSSA = 0.
#PF(error code)	If a page fault occurs in accessing memory. If one or more pages of the target SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page. If CET is enabled for the enclave and the target CET SSA frame is not readable/writable, or does not resolve to a valid PT_REG EPC page.

EENTER—Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLU[EENTER]	IR	V/V	SGX1	This leaf function is used to enter an enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	
IR	EENTER (In)	Content of RBX.CSSA (Out)	Address of a TCS (In)	Address of AEP (In)	Address of IP following EENTER (Out)

Description

The ENCLU[EENTER] instruction transfers execution to an enclave. At the end of the instruction, the logical processor is executing in enclave mode at the RIP computed as EnclaveBase + TCS.OENTRY. If the target address is not within the CS segment (32-bit) or is not canonical (64-bit), a #GP(0) results.

EENTER Memory Parameter Semantics

TCS
Enclave access

EENTER is a serializing instruction. The instruction faults if any of the following occurs:

Address in RBX is not properly aligned.	Any TCS.FLAGS's must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use.	Either of TCS-specified FS and GS segment is not a subsets of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCRO.
If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.	

The following operations are performed by EENTER:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or interrupt.
- The AEP contained in RCX is stored into the TCS for use by AEXs. FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCRO is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 40.1.2):
 - On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 40.2.5).
 - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 40.2.2).

- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.
- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 40.2.3):
 - All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
 - PEBS is suppressed.
 - AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set
 - If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 38-62. Base Concurrency Restrictions of EENTER

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EENTER	TCS [DS:RBX]	Shared	#GP	

Table 38-63. Additional Concurrency Restrictions of EENTER

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EENTER	TCS [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EENTER Operational Flow

Name	Type	Size (Bits)	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for GS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.

TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)

IF (TMP_MODE64 = 0 and (DS not usable or ((DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1)))
 THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)

IF (TMP_MODE64 = 0)
 THEN

```

    IF(CS.base ≠ 0 or DS.base ≠ 0) #GP(0); FI;
    IF(ES usable and ES.base ≠ 0) #GP(0); FI;
    IF(SS usable and SS.base ≠ 0) #GP(0); FI;
    IF(SS usable and SS.B = 0) #GP(0); FI;
FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (CS:RCX is not canonical) )
    THEN #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions are operating on TCS)
    THEN #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) or (EPCM(DS:RBX).PT ≠ PT_TCS) )
    THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS := Address of SECS for TCS;

(* Ensure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & FFFFFFFFCH) ≠ 0)
    THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    THEN #GP(0); FI;

(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
    THEN #GP(0); FI;

```

```
IF (CR4.OSFXSR = 0)
    THEN #GP(0); FI;
```

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)

```
IF (CR4.OSXSAVE = 0)
    THEN
        IF (TMP_SECS.ATTRIBUTES.XFRM ≠ 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUTES.XFRM & XCRO) ≠ TMP_SECS.ATTRIBUTES.XFRM) THEN #GP(0); FI;
FI;
```

```
IF ((DS:RBX).CSSA.FLAGS.DBGOPTIN = 0) and (DS:RBX).CSSA.FLAGS.AEXNOTIFY ≠ TMP_SECS.ATTRIBUTES.AEXNOTIFY))
    THEN #GP(0); FI;
```

(* Make sure the SSA contains at least one more frame *)

```
IF ( (DS:RBX).CSSA ≥ (DS:RBX).NSSA)
    THEN #GP(0); FI;
```

(* Compute linear address of SSA frame *)

```
TMP_SSA := (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * (DS:RBX).CSSA;
TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);
```

```
FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
```

(* Check page is read/write accessible *)

Check that DS:TMP_SSA_PAGE is read/write accessible;
If a fault occurs, release locks, abort, and deliver that fault;

```
IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMP_SSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
```

```
    (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
```

```
    (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0) )
```

```
    THEN #PF(DS:TMP_SSA_PAGE); FI;
```

```
    CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);
```

```
ENDFOR
```

(* Compute address of GPR area*)

```
TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
```

If a fault occurs; release locks, abort, and deliver that fault;

```
IF (DS:TMP_GPR does not resolve to EPC page)
```

```
    THEN #PF(DS:TMP_GPR); FI;
```

```
IF (EPCM(DS:TMP_GPR).VALID = 0)
```

```
    THEN #PF(DS:TMP_GPR); FI;
```

```
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
```

```
    THEN #PF(DS:TMP_GPR); FI;
```

```
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
```

```
    THEN #PF(DS:TMP_GPR); FI;
```

```

IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
(EPCM(DS:TMP_GPR).ENCLAVESECS EPCM(DS:RBX).ENCLAVESECS) or
(EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
THEN #PF(DS:TMP_GPR); FI;

```

```

IF (TMP_MODE64 = 0)
THEN
    IF (TMP_GPR + (GPR_SIZE - 1) is not in DS segment) THEN #GP(0); FI;
FI;

```

```

CR_GPR_PA := Physical_Address (DS: TMP_GPR);

```

```

(* Validate TCS.OENTRY *)
TMP_TARGET := (DS:RBX).OENTRY + TMP_SECS.BASEADDR;
IF (TMP_MODE64 = 1)
THEN
    IF (TMP_TARGET is not canonical) THEN #GP(0); FI;
ELSE
    IF (TMP_TARGET > CS limit) THEN #GP(0); FI;
FI;

```

```

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
THEN
    TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
    TMP_FSLIMIT := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
    TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
    TMP_GSLIMIT := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
    (* if FS wrap-around, make sure DS has no holes*)
    IF (TMP_FSLIMIT < TMP_FSBASE)
    THEN
        IF (DS.limit < 4GB) THEN #GP(0); FI;
    ELSE
        IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
    FI;
    (* if GS wrap-around, make sure DS has no holes*)
    IF (TMP_GSLIMIT < TMP_GSBASE)
    THEN
        IF (DS.limit < 4GB) THEN #GP(0); FI;
    ELSE
        IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
    FI;
ELSE
    TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
    TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
    IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
    THEN #GP(0); FI;
FI;

```

```

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE)
THEN #GP(0); FI;

```

```

TMP_IA32_U_CET := 0

```


TMP_SSP := 0

IF CPUID.(EAX=12H, ECX=1):EAX[6] = 1

THEN

IF (CR4.CET = 0)

THEN

(* If part does not support CET or CET has not been enabled and enclave requires CET then fail *)

IF (TMP_SECS.CET_ATTRIBUTES ≠ 0 OR TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) #GP(0); FI;

FI;

(* If indirect branch tracking or shadow stacks enabled but CET state save area is not 16B aligned then fail #EENTER *)

IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN = 1 OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN = 1)

THEN

IF (DS:RBX.OCETSSA is not 16B aligned) #GP(0); FI;

FI;

IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN)

THEN

(* Setup CET state from SECS, note tracker goes to IDLE *)

TMP_IA32_U_CET = TMP_SECS.CET_ATTRIBUTES;

IF (TMP_IA32_U_CET.LEG_IW_EN = 1 AND TMP_IA32_U_CET.ENDBR_EN = 1)

THEN

TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.BASEADDR;

TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.CET_LEG_BITMAP_BASE;

FI;

(* Compute linear address of what will become new CET state save area and cache its PA *)

TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA) * 16

TMP_CET_SAVE_PAGE = TMP_CET_SAVE_AREA & ~0xFFF;

Check the TMP_CET_SAVE_PAGE page is read/write accessible

If fault occurs release locks, abort, and deliver fault

(* Read the EPCM VALID, PENDING, MODIFIED, BLOCKED, and PT fields atomically *)

IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR

(EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS))

THEN

#PF(DS:TMP_CET_SAVE_PAGE);

FI;

CR_CET_SAVE_AREA_PA := Physical address(DS:TMP_CET_SAVE_AREA)

IF TMP_IA32_U_CET.SH_STK_EN = 1

THEN

TMP_SSP = TCS.PREVSSP;

FI;

FI;

```

FI;

CR_ENCLAVE_MODE := 1;
CR_ACTIVE_SECS := TMP_SECS;
CR_ELRANGE := (TMPSECS.BASEADDR, TMP_SECS.SIZE);

(* Save state for possible AEXs *)
CR_TCS_PA := Physical_Address (DS:RBX);
CR_TCS_LA := RBX;
CR_TCS_LA.AEP := RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector := FS.selector;
CR_SAVE_FS_base := FS.base;
CR_SAVE_FS_limit := FS.limit;
CR_SAVE_FS_access_rights := FS.access_rights;
CR_SAVE_GS_selector := GS.selector;
CR_SAVE_GS_base := GS.base;
CR_SAVE_GS_limit := GS.limit;
CR_SAVE_GS_access_rights := GS.access_rights;

(* If XSAVE is enabled, save XCRO and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCRO := XCRO;
    XCRO := TMP_SECS.ATTRIBUTES.XFRM;
FI;

RCX := RIP;
RIP := TMP_TARGET;
RAX := (DS:RBX).CSSA;
(* Save the outside RSP and RBP so they can be restored on interrupt or EEXIT *)
DS:TMP_SSA.U_RSP := RSP;
DS:TMP_SSA.U_RBP := RBP;

(* Do the FS/GS swap *)
FS.base := TMP_FSBASE;
FS.limit := DS:RBX.FSLIMIT;
FS.type := 0001b;
FS.W := DS.W;
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS.L;
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS.W;
GS.S := 1;

```

```
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS.L;
GS.unusable := 0;
GS.selector := 0BH;
```

```
CR_DBGOPTIN := TCS.FLAGS.DBGOPTIN;
Suppress_all_code_breakpoints_that_are_outside_ELRANGE;
```

```
IF (CR_DBGOPTIN = 0)
  THEN
    Suppress_all_code_breakpoints_that_overlap_with_ELRANGE;
    CR_SAVE_TF := RFLAGS.TF;
    RFLAGS.TF := 0;
    Suppress_monitor_trap_flag for the source of the execution of the enclave;
    Suppress any pending debug exceptions;
    Suppress any pending MTF VM exit;
  ELSE
    IF RFLAGS.TF = 1
      THEN pend a single-step #DB at the end of EENTER; FI;
    IF the "monitor trap flag" VM-execution control is set
      THEN pend an MTF VM exit at the end of EENTER; FI;
  FI;
```

```
IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7H, ECX=0):ECX[CET_SS] = 1))
  THEN
    (* Save enclosing application CET state into save registers *)
    CR_SAVE_IA32_U_CET := IA32_U_CET
    (* Setup enclave CET state *)
    IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
      THEN
        CR_SAVE_SSP := SSP
        SSP := TMP_SSP
      FI;

    IA32_U_CET := TMP_IA32_U_CET;
```

```
FI;
```

```
Flush_linear_context;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry.

Protected Mode Exceptions

#GP(0)	<p>If DS:RBX is not page aligned.</p> <p>If the enclave is not initialized.</p> <p>If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.</p> <p>If the thread is not in the INACTIVE state.</p> <p>If CS, DS, ES or SS bases are not all zero.</p> <p>If executed in enclave mode.</p> <p>If any reserved field in the TCS FLAG is set.</p> <p>If the target address is not within the CS segment.</p> <p>If CR4.OSFXSR = 0.</p> <p>If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.</p> <p>If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.</p> <p>If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory.</p> <p>If DS:RBX does not point to a valid TCS.</p> <p>If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.</p>

64-Bit Mode Exceptions

#GP(0)	<p>If DS:RBX is not page aligned.</p> <p>If the enclave is not initialized.</p> <p>If the thread is not in the INACTIVE state.</p> <p>If CS, DS, ES or SS bases are not all zero.</p> <p>If executed in enclave mode.</p> <p>If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.</p> <p>If the target address is not canonical.</p> <p>If CR4.OSFXSR = 0.</p> <p>If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.</p> <p>If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.</p> <p>If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory operands.</p> <p>If DS:RBX does not point to a valid TCS.</p> <p>If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.</p>

EEXIT—Exits an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLU[EEXIT]	IR	V/V	SGX1	This leaf function is used to exit an enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EEXIT (In)	Target address outside the enclave (In)	Address of the current AEP (Out)

Description

The ENCLU[EEXIT] instruction exits the currently executing enclave and branches to the location specified in RBX. RCX receives the current AEP. If RBX is not within the CS (32-bit mode) or is not canonical (64-bit mode) a #GP(0) results.

EEXIT Memory Parameter Semantics

Target Address
Non-Enclave read and execute access

If RBX specifies an address that is inside the enclave, the instruction will complete normally. The fetch of the next instruction will occur in non-enclave mode, but will attempt to fetch from inside the enclave. This fetch returns a fixed data pattern.

If secrets are contained in any registers, it is responsibility of enclave software to clear those registers.

If XCR0 was modified on enclave entry, it is restored to the value it had at the time of the most recent EENTER or ERESUME.

If the enclave is opt-out, RFLAGS.TF is loaded from the value previously saved on EENTER.

Code and data breakpoints are unsuppressed.

Performance monitoring counters are unsuppressed.

Concurrency Restrictions

Table 38-64. Base Concurrency Restrictions of EEXIT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EEXIT		Concurrent		

Table 38-65. Additional Concurrency Restrictions of EEXIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EEXIT		Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EEXIT Operational Flow

Name	Type	Size (Bits)	Description
TMP_RIP	Effective Address	32/64	Saved copy of CRIP for use when creating LBR.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));
```

```
IF (TMP_MODE64 = 1)
  THEN
    IF (RBX is not canonical) THEN #GP(0); FI;
  ELSE
    IF (RBX > CS limit) THEN #GP(0); FI;
FI;
```

```
TMP_RIP := CRIP;
RIP := RBX;
```

```
(* Return current AEP in RCX *)
RCX := CR_TCS_PA.AEP;
```

```
(* Do the FS/GS swap *)
FS.selector := CR_SAVE_FS.selector;
FS.base := CR_SAVE_FS.base;
FS.limit := CR_SAVE_FS.limit;
FS.access_rights := CR_SAVE_FS.access_rights;
GS.selector := CR_SAVE_GS.selector;
GS.base := CR_SAVE_GS.base;
GS.limit := CR_SAVE_GS.limit;
GS.access_rights := CR_SAVE_GS.access_rights;
```

```
(* Restore XCRO if needed *)
IF (CR4.OSXSAVE = 1)
  XCRO := CR_SAVE__XCRO;
FI;
```

```
Unsuppress_all_code_breakpoints_that_are_outside_ELRANGE;
```

```
IF (CR_DBGOPTIN = 0)
  THEN
    UnSuppress_all_code_breakpoints_that_overlap_with_ELRANGE;
    Restore suppressed breakpoint matches;
    RFLAGS.TF := CR_SAVE_TF;
    UnSuppress_montior_trap_flag;
    UnSuppress_LBR_Generation;
    UnSuppress_performance_monitoring_activity;
    Restore performance monitoring counter AnyThread demotion to MyThread in enclave back to AnyThread
  FI;
```

```
IF RFLAGS.TF = 1
  THEN Pend Single-Step #DB at the end of EEXIT;
FI;
```

```

IF the "monitor trap flag" VM-execution control is set
  THEN pend a MTF VM exit at the end of EEXIT;
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
  THEN
    (* Record PREVSSP *)
    IF (IA32_U_CET.SH_STK_EN == 1)
      THEN CR_TCS_PA.PREVSSP = SSP; FI;
  FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 1)
  THEN
    (* Restore enclosing app's CET state from the save registers *)
    IA32_U_CET := CR_SAVE_IA32_U_CET;
    IF (CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1)
      THEN SSP := CR_SAVE_SSP; FI;

    (* Update enclosing app's TRACKER if enclosing app has indirect branch tracking enabled *)
    IF (CR4.CET = 1 AND IA32_U_CET.ENDBR_EN = 1)
      THEN
        IA32_U_CET.TRACKER := WAIT_FOR_ENDBRANCH;
        IA32_U_CET.SUPPRESS := 0;
      FI;
  FI;

CR_ENCLAVE_MODE := 0;
CR_TCS_PA.STATE := INACTIVE;

(* Assure consistent translations *)
Flush_linear_context;

```

Flags Affected

RFLAGS.TF is restored from the value previously saved in EENTER or ERESUME.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave. If RBX is outside the CS segment.
#PF(error code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave. If RBX is not canonical.
#PF(error code)	If a page fault occurs in accessing memory operands.

EGETKEY—Retrieves a Cryptographic Key

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLU[EGETKEY]	IR	V/V	SGX1	This leaf function retrieves a cryptographic key.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EGETKEY (In)	Return error code (Out)	Address to a KEYREQUEST (In)	Address of the OUTPUTDATA (In)

Description

The ENCLU[EGETKEY] instruction returns a 128-bit secret key from the processor specific key hierarchy. The register RBX contains the effective address of a KEYREQUEST structure, which the instruction interprets to determine the key being requested. The Requesting Keys section below provides a description of the keys that can be requested. The RCX register contains the effective address where the key will be returned. Both the addresses in RBX & RCX should be locations inside the enclave.

EGETKEY derives keys using a processor unique value to create a specific key based on a number of possible inputs. This instruction leaf can only be executed inside an enclave.

EGETKEY Memory Parameter Semantics

KEYREQUEST	OUTPUTDATA
Enclave read access	Enclave write access

After validating the operands, the instruction determines which key is to be produced and performs the following actions:

- The instruction assembles the derivation data for the key based on the Table 38-66.
- Computes derived key using the derivation data and package specific value.
- Outputs the calculated key to the address in RCX.

The instruction fails with #GP(0) if the operands are not properly aligned. Successful completion of the instruction will clear RFLAGS.{ZF, CF, AF, OF, SF, PF}. The instruction returns an error code if the user tries to request a key based on an invalid CPUSVN or ISVSVN (when the user request is accepted, see the table below), requests a key for which it has not been granted the attribute to request, or requests a key that is not supported by the hardware. These checks may be performed in any order. Thus, an indication by error number of one cause (for example, invalid attribute) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same Enclave. The correctness of software should not rely on the order resulting from the checks documented in this section. In such cases the ZF flag is set and the corresponding error bit (SGX_INVALID_SVN, SGX_INVALID_ATTRIBUTE, SGX_INVALID_KEYNAME) is set in RAX and the data at the address specified by RCX is unmodified.

Requesting Keys

The KEYREQUEST structure (see Section 35.18.1) identifies the key to be provided. The Keyrequest.KeyName field identifies which type of key is requested.

Deriving Keys

Key derivation is based on a combination of the enclave specific values (see Table 38-66) and a processor key. Depending on the key being requested a field may either be included by definition or the value may be included from the KeyRequest. A “yes” in Table 38-66 indicates the value for the field is included from its default location, identified in the source row, and a “request” indicates the values for the field is included from its corresponding KeyRequest field.

Table 38-66. Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODIG	ISVEXT PRODIG	ISVFAM ILYID	MRENCLAVE	MRSIGNER	CONFIG ID	CONFIGS VN	RAND
Source	Key Dependent Constant	Y := SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;	CR_SGX_OWNER EPOCH	Y := CPUSVN Register;	R := Req.ISV SVN;	SECS.ISVID	SECS.IS VEXTPR ODID	SECS.IS VFAMIL YID	SECS.MRENCLAVE	SECS.MRSIGNER	SECS.CO NFIGID	SECS.CO NFIGSVN	Req. KEYID
		R := AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;		R := Req.CPU SVN;									
EINITTOKEN	Yes	Request	Yes	Request	Request	Yes	No	No	No	Yes	No	No	Request
Report	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	Yes	Yes	Request
Seal	Yes	Request	Yes	Request	Request	Request	Request	Request	Request	Request	Request	Request	Request
Provisioning	Yes	Request	No	Request	Request	Yes	No	No	No	Yes	No	No	Yes
Provisioning Seal	Yes	Request	No	Request	Request	Request	Request	Request	No	Yes	Request	Request	Yes

Keys that permit the specification of a CPU or ISV's code's, or enclave configuration's SVNs have additional requirements. The caller may not request a key for an SVN beyond the current CPU, ISV or enclave configuration's SVN, respectively.

Several keys are access controlled. Access to the Provisioning Key and Provisioning Seal key requires the enclave's ATTRIBUTES.PROVISIONKEY be set. The EINITTOKEN Key requires ATTRIBUTES.EINITTOKEN_KEY be set and SECS.MRSIGNER equal IA32_SGXLEPUBKEYHASH.

Some keys are derived based on a hardcoded PKCS padding constant (352 byte string):

HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;

HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH

HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;

The error codes are:

Table 38-67. EGETKEY Return Value in RAX

Error Code (see Table 38-4)	Value	Description
No Error	0	EGETKEY successful.
SGX_INVALID_ATTRIBUTE		The KEYREQUEST contains a KEYNAME for which the enclave is not authorized.
SGX_INVALID_CPUSVN		If KEYREQUEST.CPUSVN is an unsupported platforms CPUSVN value.
SGX_INVALID_ISVSVN		If KEYREQUEST software SVN (ISVSVN or CONFIGSVN) is greater than the enclave's corresponding SVN.
SGX_INVALID_KEYNAME		If KEYREQUEST.KEYNAME is an unsupported value.

Concurrency Restrictions

Table 38-68. Base Concurrency Restrictions of EGETKEY

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent		
	OUTPUTDATA [DS:RCX]	Concurrent		

Table 38-69. Additional Concurrency Restrictions of EGETKEY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EGETKEY Operational Flow

Name	Type	Size (Bits)	Description
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_ATTRIBUTES		128	Temp Space for the calculation of the sealable Attributes.
TMP_ISVEXTPRODID		16 bytes	Temp Space for ISVEXTPRODID.
TMP_ISVPRODID		2 bytes	Temp Space for ISVPRODID.
TMP_ISVFAMILYID		16 bytes	Temp Space for ISVFAMILYID.
TMP_CONFIGID		64 bytes	Temp Space for CONFIGID.
TMP_CONFIGSVN		2 bytes	Temp Space for CONFIGSVN.
TMP_OUTPUTKEY		128	Temp Space for the calculation of the key.

(* Make sure KEYREQUEST is properly aligned and inside the current enclave *)
 IF ((DS:RBX is not 512Byte aligned) or (DS:RBX is not within CR_ELRANGE))
 THEN #GP(0); FI;

(* Make sure DS:RBX is an EPC address and the EPC page is valid *)
 IF ((DS:RBX does not resolve to an EPC address) or (EPCM(DS:RBX).VALID = 0))
 THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
 THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
 IF ((EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
 (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH)) or (EPCM(DS:RBX).R = 0))
 THEN #PF(DS:RBX);
 FI;

(* Make sure OUTPUTDATA is properly aligned and inside the current enclave *)
 IF ((DS:RCX is not 16Byte aligned) or (DS:RCX is not within CR_ELRANGE))
 THEN #GP(0); FI;

(* Make sure DS:RCX is an EPC address and the EPC page is valid *)
 IF ((DS:RCX does not resolve to an EPC address) or (EPCM(DS:RCX).VALID = 0))

```

THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
  THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
  (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).W = 0) )
  THEN #PF(DS:RCX);
FI;

(* Verify RESERVED spaces in KEYREQUEST are valid *)
IF ( (DS:RBX).RESERVED ≠ 0) or (DS:RBX.KEYPOLICY.RESERVED ≠ 0) )
  THEN #GP(0); FI;

TMP_CURRENTSECS := CR_ACTIVE_SECS;

(* Verify that CONFIGSVN & New Policy bits are not used if KSS is not enabled *)
IF ((TMP_CURRENTSECS.ATTRIBUTES.KSS == 0) AND ((DS:RBX.KEYPOLICY & 0x003C ≠ 0) OR (DS:RBX.CONFIGSVN > 0)))
  THEN #GP(0); FI;

(* Determine which enclave attributes that must be included in the key. Attributes that must always be include INIT & DEBUG *)
REQUIRED_SEALING_MASK[127:0] := 00000000 00000000 00000000 00000003H;
TMP_ATTRIBUTES := (DS:RBX.ATTRIBUTEMASK | REQUIRED_SEALING_MASK) & TMP_CURRENTSECS.ATTRIBUTES;

(* Compute MISCSELECT fields to be included *)
TMP_MISCSELECT := DS:RBX.MISCMASK & TMP_CURRENTSECS.MISCSELECT

(* Compute CET_ATTRIBUTES fields to be included *)
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
  THEN TMP_CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES_MASK & TMP_CURRENTSECS.CET_ATTRIBUTES; FI;
TMP_KEYDEPENDENCIES := 0;

CASE (DS:RBX.KEYNAME)
  SEAL_KEY:
    IF (DS:RBX.CPUSVN is beyond current CPU configuration)
      THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_CPUSVN;
        GOTO EXIT;
    FI;
    IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
      THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ISVSVN;
        GOTO EXIT;
    FI;
    IF (DS:RBX.CONFIGSVN > TMP_CURRENTSECS.CONFIGSVN)
      THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ISVSVN;
        GOTO EXIT;
    FI;

    (*Include enclave identity?*)

```

```

TMP_MRENCLAVE := 0;
IF (DS:RBX.KEYPOLICY.MRENCLAVE = 1)
    THEN TMP_MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
FI;
(*Include enclave author?*)
TMP_MRSIGNER := 0;
IF (DS:RBX.KEYPOLICY.MRSIGNER = 1)
    THEN TMP_MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
FI;
(* Include enclave product family ID? *)
TMP_ISVFAMILYID := 0;
IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
    THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
FI;

(* Include enclave product ID? *)
TMP_ISVPRODID := 0;
IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
    THEN TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
FI;

(* Include enclave Config ID? *)
TMP_CONFIGID := 0;
TMP_CONFIGSVN := 0;
IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
    THEN TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;
    TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
FI;

(* Include enclave extended product ID? *)
TMP_ISVEXTPRODID := 0;
IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1)
    THEN TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
FI;

//Determine values key is based on
TMP_KEYDEPENDENCIES.KEYNAME := SEAL_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTESMASK;
TMP_KEYDEPENDENCIES.MRENCLAVE := TMP_MRENCLAVE;
TMP_KEYDEPENDENCIES.MRSIGNER := TMP_MRSIGNER;
TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;

```

```

TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
IF CPUID.(EAX=12H, ECX=1):EAX[6] = 1
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := DS:RBX.CET_ATTRIBUTES_MASK;
    FI;
BREAK;
REPORT_KEY:
//Determine values key is based on
TMP_KEYDEPENDENCIES.KEYNAME := REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := 0;
TMP_KEYDEPENDENCIES.ISVSVN := 0;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
TMP_KEYDEPENDENCIES.MRSIGNER := 0;
TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := TMP_CURRENTSECS.CONFIGID;
TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
    FI;
BREAK;
EINITTOKEN_KEY:
(* Check ENCLAVE has EINITTOKEN Key capability *)
IF (TMP_CURRENTSECS.ATTRIBUTES.EINITTOKEN_KEY = 0)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ATTRIBUTE;
        GOTO EXIT;
    FI;
IF (DS:RBX.CPUSVN is beyond current CPU configuration)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_CPUSVN;
        GOTO EXIT;
    FI;
IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ISVSVN;
        GOTO EXIT;
    FI;
FI;

```

```

(* Determine values key is based on *)
TMP_KEYDEPENDENCIES.KEYNAME := EINITTOKEN_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := 0;
TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
FI;
BREAK;
PROVISION_KEY:
(* Check ENCLAVE has PROVISIONING capability *)
IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ATTRIBUTE;
        GOTO EXIT;
FI;
IF (DS:RBX.CPUSVN is beyond current CPU configuration)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_CPUSVN;
        GOTO EXIT;
FI;
IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ISVSVN;
        GOTO EXIT;
FI;
(* Determine values key is based on *)
TMP_KEYDEPENDENCIES.KEYNAME := PROVISION_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := 0;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;

```

```

TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTESMASK;
TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
TMP_KEYDEPENDENCIES.KEYID := 0;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := 0;
TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := 0;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
FI;
BREAK;
PROVISION_SEAL_KEY:
(* Check ENCLAVE has PROVISIONING capability *)
IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ATTRIBUTE;
        GOTO EXIT;
FI;
IF (DS:RBX.CPUSVN is beyond current CPU configuration)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_CPUSVN;
        GOTO EXIT;
FI;
IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ISVSVN;
        GOTO EXIT;
FI;
(* Include enclave product family ID? *)
TMP_ISVFAMILYID := 0;
IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
    THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
FI;

(* Include enclave product ID? *)
TMP_ISVPRODID := 0;
IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
    THEN TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
FI;

(* Include enclave Config ID? *)
TMP_CONFIGID := 0;
TMP_CONFIGSVN := 0;
IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
    THEN TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;

```

```

TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
FI;

(* Include enclave extended product ID? *)
TMP_ISVEXTPRODID := 0;
IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1)
    TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
FI;

    (* Determine values key is based on *)
    TMP_KEYDEPENDENCIES.KEYNAME := PROVISION_SEAL_KEY;
    TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
    TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
    TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
    TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
    TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := 0;
    TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
    TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
    TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
    TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
    TMP_KEYDEPENDENCIES.KEYID := 0;
    TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
    TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
    TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
    TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
    TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
    TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
    TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;
    TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
    IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
        THEN
            TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
            TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
        FI;
    BREAK;
DEFAULT:
    (* The value of KEYNAME is invalid *)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_KEYNAME;
    GOTO EXIT;
ESAC;

(* Calculate the final derived key and output to the address in RCX *)
TMP_OUTPUTKEY := derivekey(TMP_KEYDEPENDENCIES);
DS:RCX[15:0] := TMP_OUTPUTKEY;
RAX := 0;
RFLAGS.ZF := 0;

EXIT:
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;

```


Flags Affected

ZF is cleared if successful, otherwise ZF is set. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

- #GP(0) If executed outside an enclave.
If a memory operand effective address is outside the current enclave.
If an effective address is not properly aligned.
If an effective address is outside the DS segment limit.
If KEYREQUEST format is invalid.
- #PF(error code) If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

- #GP(0) If executed outside an enclave.
If a memory operand effective address is outside the current enclave.
If an effective address is not properly aligned.
If an effective address is not canonical.
If KEYREQUEST format is invalid.
- #PF(error code) If a page fault occurs in accessing memory operands.

EMODPE—Extend an EPC Page Permissions

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLU[EMODPE]	IR	V/V	SGX2	This leaf function extends the access rights of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EMODPE (In)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function extends the access rights associated with an existing EPC page in the running enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not extend the page permissions will have no effect. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPE leaf function.

EMODPE Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EMODPE Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is not valid.	RCX does not contain an effective address of an EPC page in the running enclave.
SECINFO contains an invalid request.	

Concurrency Restrictions

Table 38-70. Base Concurrency Restrictions of EMODPE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODPE	Target [DS:RCX]	Concurrent		
	SECINFO [DS:RBX]	Concurrent		

Table 38-71. Additional Concurrency Restrictions of EMODPE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODPE	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EMODPE Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

IF (DS:RBX is not 64Byte Aligned)
 THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
 THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE))
 THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
 THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
 THEN #PF(DS:RCX); FI;

IF ((EPCM(DS:RBX).VALID = 0) or (EPCM(DS:RBX).R = 0) or (EPCM(DS:RBX).PENDING ≠ 0) or (EPCM(DS:RBX).MODIFIED ≠ 0) or
 (EPCM(DS:RBX).BLOCKED ≠ 0) or (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
 (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0xFFF)))
 THEN #PF(DS:RBX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
 IF (SCRATCH_SECINFO reserved fields are not zero)
 THEN #GP(0); FI;

(* Check security attributes of the EPC page *)
 IF ((EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
 (EPCM(DS:RCX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS))
 THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
 IF (EPC page in use by another SGX2 instruction)
 THEN #GP(0); FI;

(* Re-Check security attributes of the EPC page *)
 IF ((EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
 (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
 (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
 THEN #PF(DS:RCX); FI;

(* Check for misconfigured SECINFO flags*)
 IF ((EPCM(DS:RCX).R = 0) and (SCRATCH_SECINFO.FLAGS.R = 0) and (SCRATCH_SECINFO.FLAGS.W ≠ 0))
 THEN #GP(0); FI;

(* Update EPCM permissions *)

EPCM(DS:RCX).R := EPCM(DS:RCX).R | SCRATCH_SECINFO.FLAGS.R;

EPCM(DS:RCX).W := EPCM(DS:RCX).W | SCRATCH_SECINFO.FLAGS.W;

EPCM(DS:RCX).X := EPCM(DS:RCX).X | SCRATCH_SECINFO.FLAGS.X;

Flags Affected

None

Protected Mode Exceptions

#GP(0)	<p>If executed outside an enclave.</p> <p>If a memory operand effective address is outside the DS segment limit.</p> <p>If a memory operand is not properly aligned.</p> <p>If a memory operand is locked.</p>
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	<p>If executed outside an enclave.</p> <p>If a memory operand is non-canonical form.</p> <p>If a memory operand is not properly aligned.</p> <p>If a memory operand is locked.</p>
#PF(error code)	If a page fault occurs in accessing memory operands.

EReport—Create a Cryptographic Report of the Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLU[EReport]	IR	V/V	SGX1	This leaf function creates a cryptographic report of the enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX	RDX
IR	EReport (In)	Address of TARGETINFO (In)	Address of REPORTDATA (In)	Address where the REPORT is written to in an OUTPUTDATA (In)

Description

This leaf function creates a cryptographic REPORT that describes the contents of the enclave. This instruction leaf can only be executed when inside the enclave. The cryptographic report can be used by other enclaves to determine that the enclave is running on the same platform.

RBX contains the effective address of the MRENCLAVE value of the enclave that will authenticate the REPORT output, using the REPORT key delivered by EGETKEY command for that enclave. RCX contains the effective address of a 64-byte REPORTDATA structure, which allows the caller of the instruction to associate data with the enclave from which the instruction is called. RDX contains the address where the REPORT will be output by the instruction.

EReport Memory Parameter Semantics

TARGETINFO	REPORTDATA	OUTPUTDATA
Read access by Enclave	Read access by Enclave	Read/Write access by Enclave

This instruction leaf perform the following:

1. Validate the 3 operands (RBX, RCX, RDX) are inside the enclave.
2. Compute a report key for the target enclave, as indicated by the value located in RBX(TARGETINFO).
3. Assemble the enclave SECS data to complete the REPORT structure (including the data provided using the RCX (REPORTDATA) operand).
4. Computes a cryptographic hash over REPORT structure.
5. Add the computed hash to the REPORT structure.
6. Output the completed REPORT structure to the address in RDX (OUTPUTDATA).

The instruction fails if the operands are not properly aligned.

CR_REPORT_KEYID, used to provide key wearout protection, is populated with a statistically unique value on boot of the platform by a trusted entity within the SGX TCB.

The instruction faults if any of the following:

EReport Faulting Conditions

An effective address not properly aligned.	An memory address does not resolve in an EPC page.
If accessing an invalid EPC page.	If the EPC page is blocked.
May page fault.	

Concurrency Restrictions

Table 38-72. Base Concurrency Restrictions of EREPORT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EREPORT	TARGETINFO [DS:RBX]	Concurrent		
	REPORTDATA [DS:RCX]	Concurrent		
	OUTPUTDATA [DS:RDX]	Concurrent		

Table 38-73. Additional Concurrency Restrictions of EREPORT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREPORT	TARGETINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RDX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREPORT Operational Flow

Name	Type	Size (bits)	Description
TMP_ATTRIBUTES		32	Physical address of SECS of the enclave to which source operand belongs.
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_REPORTKEY		128	REPORTKEY generated by the instruction.
TMP_REPORT		3712	

TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Address verification for TARGETINFO (RBX) *)

IF ((DS:RBX is not 512Byte Aligned) or (DS:RBX is not within CR_ELRange))
THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).VALID = 0)
THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)

IF ((EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
(EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH)) or (EPCM(DS:RBX).R = 0))

INTEL® SGX INSTRUCTION REFERENCES

```
    THEN #PF(DS:RBX);  
FI;
```

(* Verify RESERVED spaces in TARGETINFO are valid *)

```
IF (DS:RBX.RESERVED != 0)  
    THEN #GP(0); FI;
```

(* Address verification for REPORTDATA (RCX) *)

```
IF ( (DS:RCX is not 128Byte Aligned) or (DS:RCX is not within CR_ELRANGE) )  
    THEN #GP(0); FI;
```

IF (DS:RCX does not resolve within an EPC)

```
    THEN #PF(DS:RCX); FI;
```

IF (EPCM(DS:RCX).VALID = 0)

```
    THEN #PF(DS:RCX); FI;
```

IF (EPCM(DS:RCX).BLOCKED = 1)

```
    THEN #PF(DS:RCX); FI;
```

(* Check page parameters for correctness *)

```
IF ( (EPCM(DS:RCX).PT != PT_REG) or (EPCM(DS:RCX).ENCLAVESECS != CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or  
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS != (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).R = 0) )  
    THEN #PF(DS:RCX);
```

```
FI;
```

(* Address verification for OUTPUTDATA (RDX) *)

```
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RDX is not within CR_ELRANGE) )  
    THEN #GP(0); FI;
```

IF (DS:RDX does not resolve within an EPC)

```
    THEN #PF(DS:RDX); FI;
```

IF (EPCM(DS:RDX).VALID = 0)

```
    THEN #PF(DS:RDX); FI;
```

IF (EPCM(DS:RDX).BLOCKED = 1)

```
    THEN #PF(DS:RDX); FI;
```

(* Check page parameters for correctness *)

```
IF ( (EPCM(DS:RDX).PT != PT_REG) or (EPCM(DS:RDX).ENCLAVESECS != CR_ACTIVE_SECS) or (EPCM(DS:RDX).PENDING = 1) or  
    (EPCM(DS:RDX).MODIFIED = 1) or (EPCM(DS:RDX).ENCLAVEADDRESS != (DS:RDX & ~0FFFH) ) or (EPCM(DS:RDX).W = 0) )  
    THEN #PF(DS:RDX);
```

```
FI;
```

(* REPORT MAC needs to be computed over data which cannot be modified *)

```
TMP_REPORT.CPUSVN := CR_CPUSVN;  
TMP_REPORT.ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;  
TMP_REPORT.ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;  
TMP_REPORT.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;  
TMP_REPORT.ISVSVN := TMP_CURRENTSECS.ISVSVN;  
TMP_REPORT.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;  
TMP_REPORT.REPORTDATA := DS:RCX[511:0];  
TMP_REPORT.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
```

```

TMP_REPORT.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
TMP_REPORT.MRRESERVED := 0;
TMP_REPORT.KEYID[255:0] := CR_REPORT_KEYID;
TMP_REPORT.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
TMP_REPORT.CONFIGID := TMP_CURRENTSECS.CONFIGID;
TMP_REPORT.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN TMP_REPORT.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES; FI;

```

(* Derive the report key *)

```

TMP_KEYDEPENDENCIES.KEYNAME := REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := 0;
TMP_KEYDEPENDENCIES.ISVSVN := 0;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := DS:RBX.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := DS:RBX.MEASUREMENT;
TMP_KEYDEPENDENCIES.MRSIGNER := 0;
TMP_KEYDEPENDENCIES.KEYID := TMP_REPORT.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := DS:RBX.MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := DS:RBX.CONFIGID;
TMP_KEYDEPENDENCIES.CONFIGSVN := DS:RBX.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
FI;

```

(* Calculate the derived key*)

```

TMP_REPORTKEY := derivekey(TMP_KEYDEPENDENCIES);

```

(* call cryptographic CMAC function, CMAC data are not including MAC&KEYID *)

```

TMP_REPORT.MAC := cmac(TMP_REPORTKEY, TMP_REPORT[3071:0] );
DS:RDX[3455: 0] := TMP_REPORT;

```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave. If the address in RCS is outside the DS segment limit. If a memory operand is not properly aligned. If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

- #GP(0) If executed outside an enclave.
 If RCX is non-canonical form.
 If a memory operand is not properly aligned.
 If a memory operand is not in the current enclave.
- #PF(error code) If a page fault occurs in accessing memory operands.

ERESUME—Re-Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLU[ERESUME]	IR	V/V	SGX1	This leaf function is used to re-enter an enclave after an interrupt.

Instruction Operand Encoding

Op/En	RAX	RBX	RCX
IR	ERESUME (In)	Address of a TCS (In)	Address of AEP (In)

Description

The ENCLU[ERESUME] instruction resumes execution of an enclave that was interrupted due to an exception or interrupt, using the machine state previously stored in the SSA.

ERESUME Memory Parameter Semantics

TCS
Enclave read/write access

The instruction faults if any of the following occurs:

Address in RBX is not properly aligned.	Any TCS.FLAGS's must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use by another enclave.	Either of TCS-specified FS and GS segment is not a subset of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCRO.
Offsets 520-535 of the XSAVE area not 0.	The bit vector stored at offset 512 of the XSAVE area must be a subset of SECS.ATTRIBUTES.XFRM.
The SSA frame is not valid or in use.	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.

The following operations are performed by ERESUME:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or an asynchronous exit due to any Interrupt event.
- The AEP contained in RCX is stored into the TCS for use by AEXs. FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCRO is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 40.1.2):
 - On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 40.2.5).
 - On opt-in entry, a single-step debug exception is pending on the instruction boundary immediately after EENTER (see Section 40.2.3).
- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.

- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 40.2.3):
 - All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
 - PEBS is suppressed.
 - AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set.
 - If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 38-74. Base Concurrency Restrictions of ERESUME

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ERESUME	TCS [DS:RBX]	Shared	#GP	

Table 38-75. Additional Concurrency Restrictions of ERESUME

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ERESUME	TCS [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ERESUME Operational Flow

Name	Type	Size	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for GS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_TARGET	Effective Address	32/64	Address of first instruction inside enclave at which execution is to resume.
TMP_SECS	Effective Address	32/64	Physical address of SECS for this enclave.
TMP_SSA	Effective Address	32/64	Address of current SSA frame.
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.
TMP_BRANCH_RECORD	LBR Record		From/to addresses to be pushed onto the LBR stack.
TMP_NOTIFY	Boolean	1	When set to 1, deliver an AEX notification.

TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)

IF (TMP_MODE64 = 0 and (DS not usable or ((DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1))))

THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)

IF (TMP_MODE64 = 0)

THEN

IF(CS.base \neq 0 or DS.base \neq 0) #GP(0); FI;

IF(ES usable and ES.base \neq 0) #GP(0); FI;

IF(SS usable and SS.base \neq 0) #GP(0); FI;

IF(SS usable and SS.B = 0) #GP(0); FI;

FI;

IF (DS:RBX is not 4KByte Aligned)

THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)

THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)

IF (TMP_MODE64 = 1 and (CS:RCX is not canonical))

THEN #GP(0); FI;

(* Check concurrency of TCS operation*)

IF (Other Intel SGX instructions are operating on TCS)

THEN #GP(0); FI;

(* TCS verification *)

IF (EPCM(DS:RBX).VALID = 0)

THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)

THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))

THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).ENCLAVEADDRESS \neq DS:RBX) or (EPCM(DS:RBX).PT \neq PT_TCS))

THEN #PF(DS:RBX); FI;

IF ((DS:RBX).OSSA is not 4KByte Aligned)

THEN #GP(0); FI;

(* Check proposed FS and GS *)

IF (((DS:RBX).OFSBASE is not 4KByte Aligned) or ((DS:RBX).OGSBASE is not 4KByte Aligned))

THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)

TMP_SECS := Address of SECS for TCS;

(* Make sure that the FLAGS field in the TCS does not have any reserved bits set *)

IF (((DS:RBX).FLAGS & FFFFFFFFCH) \neq 0)

THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)

IF (the enclave is not already initialized)

THEN #GP(0); FI;

(* make sure the logical processor's operating mode matches the enclave *)

IF ((TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT))

THEN #GP(0); FI;

IF (CR4.OSFXSR = 0)

THEN #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)

IF (CR4.OSXSAVE = 0)

THEN

IF (TMP_SECS.ATTRIBUTES.XFRM ≠ 03H) THEN #GP(0); FI;

ELSE

IF ((TMP_SECS.ATTRIBUTES.XFRM & XCRO) ≠ TMP_SECS.ATTRIBUTES.XFRM) THEN #GP(0); FI;

FI;

IF ((DS:RBX).CSSA.FLAGS.DBGOPTIN = 0) and (DS:RBX).CSSA.FLAGS.AEXNOTIFY ≠ TMP_SECS.ATTRIBUTES.AEXNOTIFY)

THEN #GP(0); FI;

(* Make sure the SSA contains at least one active frame *)

IF ((DS:RBX).CSSA = 0)

THEN #GP(0); FI;

(* Compute linear address of SSA frame *)

TMP_SSA := (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * ((DS:RBX).CSSA - 1);

TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE

(* Check page is read/write accessible *)

Check that DS:TMP_SSA_PAGE is read/write accessible;

If a fault occurs, release locks, abort and deliver that fault;

IF (DS:TMP_SSA_PAGE does not resolve to EPC page)

THEN #PF(DS:TMP_SSA_PAGE); FI;

IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)

THEN #PF(DS:TMP_SSA_PAGE); FI;

IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)

THEN #PF(DS:TMP_SSA_PAGE); FI;

IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))

THEN #PF(DS:TMP_SSA_PAGE); FI;

IF ((EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMP_SSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or

(EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or

(EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0))

THEN #PF(DS:TMP_SSA_PAGE); FI;

CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);

ENDFOR

(* Compute address of GPR area*)

TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);

Check that DS:TMP_SSA_PAGE is read/write accessible;

If a fault occurs, release locks, abort and deliver that fault;

IF (DS:TMP_GPR does not resolve to EPC page)

THEN #PF(DS:TMP_GPR); FI;

IF (EPCM(DS:TMP_GPR).VALID = 0)

```

    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0))
    THEN #PF(DS:TMP_GPR); FI;

```

```

IF (TMP_MODE64 = 0)
    THEN
        IF (TMP_GPR + (GPR_SIZE - 1) is not in DS segment) THEN #GP(0); FI;
FI;

```

```

CR_GPR_PA := Physical_Address (DS: TMP_GPR);

```

```

IF ((DS:RBX).FLAGS.AEXNOTIFY = 1) and (DS:TMP_GPR.AEXNOTIFY[0] = 1))
    THEN
        TMP_NOTIFY := 1;
    ELSE
        TMP_NOTIFY := 0;
FI;

```

```

IF (TMP_NOTIFY = 1)
    THEN
        (* Make sure the SSA contains at least one more frame *)
        IF ((DS:RBX).CSSA ≥ (DS:RBX).NSSA)
            THEN #GP(0); FI;

        TMP_SSA := TMP_SSA + 4096 * TMP_SECS.SSAFRAMESIZE;
        TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

```

```

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;

    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or
        (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMP_SSA_PAGE) or
        (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
        (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

```

(* Compute address of GPR area*)

TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);

If a fault occurs; release locks, abort and deliver that fault;

IF (DS:TMP_GPR does not resolve to EPC page)

THEN #PF(DS:TMP_GPR); FI;

IF (EPCM(DS:TMP_GPR).VALID = 0)

THEN #PF(DS:TMP_GPR); FI;

IF (EPCM(DS:TMP_GPR).BLOCKED = 1)

THEN #PF(DS:TMP_GPR); FI;

IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))

THEN #PF(DS:TMP_GPR); FI;

IF ((EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or

(EPCM(DS:TMP_GPR).PT ≠ PT_REG) or

(EPCM(DS:TMP_GPR).ENCLAVESECS EPCM(DS:RBX).ENCLAVESECS) or

(EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0))

THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)

THEN

IF (TMP_GPR + (GPR_SIZE - 1) is not in DS segment) THEN #GP(0); FI;

FI;

CR_GPR_PA := Physical_Address (DS: TMP_GPR);

TMP_TARGET := (DS:RBX).OENTRY + TMP_SECS.BASEADDR;

ELSE

TMP_TARGET := (DS:TMP_GPR).RIP;

FI;

IF (TMP_MODE64 = 1)

THEN

IF (TMP_TARGET is not canonical) THEN #GP(0); FI;

ELSE

IF (TMP_TARGET > CS limit) THEN #GP(0); FI;

FI;

(* Check proposed FS/GS segments fall within DS *)

IF (TMP_MODE64 = 0)

THEN

TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;

TMP_FSLIMIT := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;

TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;

TMP_GSLIMIT := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;

(* if FS wrap-around, make sure DS has no holes*)

IF (TMP_FSLIMIT < TMP_FSBASE)

THEN

IF (DS.limit < 4GB) THEN #GP(0); FI;

ELSE

IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;

FI;

(* if GS wrap-around, make sure DS has no holes*)

IF (TMP_GSLIMIT < TMP_GSBASE)

THEN

```

        IF (DS.limit < 4GB) THEN #GP(0); FI;
    ELSE
        IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
    FI;
ELSE
    IF (TMP_NOTIFY = 1)
        THEN
            TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
            TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        ELSE
            TMP_FSBASE := DS:TMP_GPR.FSBASE;
            TMP_GSBASE := DS:TMP_GPR.GSBASE;
        FI;
    IF ((TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
        THEN #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE)
    THEN #GP(0); FI;

TMP_IA32_U_CET := 0
TMP_SSP := 0

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        IF ( CR4.CET = 0 )
            THEN
                (* If part does not support CET or CET has not been enabled and enclave requires CET then fail *)
                IF (TMP_SECS.CET_ATTRIBUTES ≠ 0 OR TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) #GP(0); FI;
            FI;
        (* If indirect branch tracking or shadow stacks enabled but CET state save area is not 16B aligned then fail ERESUME *)
        IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN = 1 OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN = 1)
            THEN
                IF (DS:RBX.OCETSSA is not 16B aligned) #GP(0); FI;
            FI;
    FI;

IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN)
    THEN
        (* Setup CET state from SECS, note tracker goes to IDLE *)
        TMP_IA32_U_CET = TMP_SECS.CET_ATTRIBUTES;
        IF (TMP_IA32_U_CET.LEG_IW_EN = 1 AND TMP_IA32_U_CET.ENDBR_EN = 1)
            THEN
                TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.BASEADDR;
                TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.CET_LEG_BITMAP_BASE;
            FI;

        (* Compute linear address of what will become new CET state save area and cache its PA *)
        IF (TMP_NOTIFY = 1)
            THEN
                TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA) * 16;
            ELSE
                TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA - 1) * 16;
            FI;
    FI;

```



```
TMP_CET_SAVE_PAGE = TMP_CET_SAVE_AREA & ~0xFFF;
```

Check the TMP_CET_SAVE_PAGE page is read/write accessible
If fault occurs release locks, abort and deliver fault

```
(* read the EPCM VALID, PENDING, MODIFIED, BLOCKED and PT fields atomically *)
IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR
(EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS))
THEN
    #PF(DS:TMP_CET_SAVE_PAGE);
FI;
```

```
CR_CET_SAVE_AREA_PA := Physical address(DS:TMP_CET_SAVE_AREA)
```

```
IF (TMP_NOTIFY = 1)
THEN
    IF TMP_IA32_U_CET.SH_STK_EN = 1
    THEN TMP_SSP = TCS.PREVSSP; FI;
ELSE
    TMP_SSP = CR_CET_SAVE_AREA_PA.SSP
    TMP_IA32_U_CET.TRACKER = CR_CET_SAVE_AREA_PA.TRACKER;
    TMP_IA32_U_CET.SUPPRESS = CR_CET_SAVE_AREA_PA.SUPPRESS;
    IF ( (TMP_MODE64 = 1 AND TMP_SSP is not canonical) OR
        (TMP_MODE64 = 0 AND (TMP_SSP & 0xFFFFFFFF00000000) ≠ 0) OR
        (TMP_SSP is not 4 byte aligned) OR
        (TMP_IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH AND TMP_IA32_U_CET.SUPPRESS = 1) OR
        (CR_CET_SAVE_AREA_PA.Reserved ≠ 0) ) #GP(0); FI;
FI;
FI;
```

```
IF (TMP_NOTIFY = 0)
```

```
THEN
```

```
(* SECS.ATTRIBUTES.XFRM selects the features to be saved. *)
(* CR_XSAVE_PAGE_n: A list of 1 or more physical address of pages that contain the XSAVE area. *)
XRSTOR(TMP_MODE64, SECS.ATTRIBUTES.XFRM, CR_XSAVE_PAGE_n);
```

```
IF (XRSTOR failed with #GP)
```

```
THEN
```

```
DS:RBX.STATE := INACTIVE;
#GP(0);
```

```
FI;
```

```
FI;
```

```
CR_ENCLAVE_MODE := 1;
```

```
CR_ACTIVE_SECS := TMP_SECS;
```

```
CR_ELRange := (TMP_SECS.BASEADDR, TMP_SECS.SIZE);
```

(* Save state for possible AEXs *)

```
CR_TCS_PA := Physical_Address (DS:RBX);
CR_TCS_LA := RBX;
CR_TCS_LA.AEP := RCX;
```

(* Save the hidden portions of FS and GS *)

```
CR_SAVE_FS_selector := FS.selector;
CR_SAVE_FS_base := FS.base;
CR_SAVE_FS_limit := FS.limit;
CR_SAVE_FS_access_rights := FS.access_rights;
CR_SAVE_GS_selector := GS.selector;
CR_SAVE_GS_base := GS.base;
CR_SAVE_GS_limit := GS.limit;
CR_SAVE_GS_access_rights := GS.access_rights;
```

IF (TMP_NOTIFY = 1)

THEN

(* If XSAVE is enabled, save XCRO and replace it with SECS.ATTRIBUTES.XFRM*)

IF (CR4.OSXSAVE = 1)

THEN

```
CR_SAVE_XCRO := XCRO;
XCRO := TMP_SECS.ATTRIBUTES.XFRM;
```

FI;

FI;

RIP := TMP_TARGET;

IF (TMP_NOTIFY = 1)

THEN

RCX := RIP;

RAX := (DS:RBX).CSSA;

(* Save the outside RSP and RBP so they can be restored on interrupt or EEXIT *)

DS:TMP_SSA.U_RSP := RSP;

DS:TMP_SSA.U_RBP := RBP;

ELSE

Restore_GPRs from DS:TMP_GPR;

(*Restore the RFLAGS values from SSA*)

RFLAGS.CF := DS:TMP_GPR.RFLAGS.CF;

RFLAGS.PF := DS:TMP_GPR.RFLAGS.PF;

RFLAGS.AF := DS:TMP_GPR.RFLAGS.AF;

RFLAGS.ZF := DS:TMP_GPR.RFLAGS.ZF;

RFLAGS.SF := DS:TMP_GPR.RFLAGS.SF;

RFLAGS.DF := DS:TMP_GPR.RFLAGS.DF;

RFLAGS.OF := DS:TMP_GPR.RFLAGS.OF;

RFLAGS.NT := DS:TMP_GPR.RFLAGS.NT;

RFLAGS.AC := DS:TMP_GPR.RFLAGS.AC;

RFLAGS.ID := DS:TMP_GPR.RFLAGS.ID;

RFLAGS.RF := DS:TMP_GPR.RFLAGS.RF;

RFLAGS.VM := 0;

IF (RFLAGS.IOPL = 3)

THEN RFLAGS.IF := DS:TMP_GPR.RFLAGS.IF; FI;

IF (TCS.FLAGS.OPTIN = 0)

```
THEN RFLAGS.TF := 0; FI;
```

```
(* If XSAVE is enabled, save XCRO and replace it with SECS.ATTRIBUTES.XFRM*)
```

```
IF (CR4.OSXSAVE = 1)
```

```
THEN
```

```
CR_SAVE_XCRO := XCRO;
```

```
XCRO := TMP_SECS.ATTRIBUTES.XFRM;
```

```
FI;
```

```
(* Pop the SSA stack*)
```

```
(DS:RBX).CSSA := (DS:RBX).CSSA - 1;
```

```
FI;
```

```
(* Do the FS/GS swap *)
```

```
FS.base := TMP_FSBASE;
```

```
FS.limit := DS:RBX.FSLIMIT;
```

```
FS.type := 0001b;
```

```
FS.W := DS.W;
```

```
FS.S := 1;
```

```
FS.DPL := DS.DPL;
```

```
FS.G := 1;
```

```
FS.B := 1;
```

```
FS.P := 1;
```

```
FS.AVL := DS.AVL;
```

```
FS.L := DS.L;
```

```
FS.unusable := 0;
```

```
FS.selector := 0BH;
```

```
GS.base := TMP_GSBASE;
```

```
GS.limit := DS:RBX.GSLIMIT;
```

```
GS.type := 0001b;
```

```
GS.W := DS.W;
```

```
GS.S := 1;
```

```
GS.DPL := DS.DPL;
```

```
GS.G := 1;
```

```
GS.B := 1;
```

```
GS.P := 1;
```

```
GS.AVL := DS.AVL;
```

```
GS.L := DS.L;
```

```
GS.unusable := 0;
```

```
GS.selector := 0BH;
```

```
CR_DBGOPTIN := TCS.FLAGS.DBGOPTIN;
```

```
Suppress all code breakpoints that are outside ELRANGE;
```

```
IF (CR_DBGOPTIN = 0)
```

```
THEN
```

```
Suppress all code breakpoints that overlap with ELRANGE;
```

```
CR_SAVE_TF := RFLAGS.TF;
```

```
RFLAGS.TF := 0;
```

```
Suppress any MTF VM exits during execution of the enclave;
```

```
Clear all pending debug exceptions;
```

```
Clear any pending MTF VM exit;
```

```
ELSE
```

```

    IF (TMP_NOTIFY = 1)
        THEN
            IF RFLAGS.TF = 1
                THEN pend a single-step #DB at the end of ERESUME; FI;
            IF the "monitor trap flag" VM-execution control is set
                THEN pend an MTF VM exit at the end of ERESUME; FI;
        ELSE
            Clear all pending debug exceptions;
            Clear pending MTF VM exits;
        FI;
FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* Save enclosing application CET state into save registers *)
        CR_SAVE_IA32_U_CET := IA32_U_CET
        (* Setup enclave CET state *)
        IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
            THEN
                CR_SAVE_SSP := SSP
                SSP := TMP_SSP;
            FI;
        IA32_U_CET := TMP_IA32_U_CET;
    FI;

(* Assure consistent translations *)
Flush_linear_context;
Clear_Monitor_FSM;
Allow_front_end_to_begin_fetch_at_new_RIP;

```

Flags Affected

RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0)	<ul style="list-style-type: none"> If DS:RBX is not page aligned. If the enclave is not initialized. If the thread is not in the INACTIVE state. If CS, DS, ES or SS bases are not all zero. If executed in enclave mode. If part or all of the FS or GS segment specified by TCS is outside the DS segment. If any reserved field in the TCS FLAG is set. If the target address is not within the CS segment. If CR4.OSFXSR = 0. If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3. If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0. If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	<ul style="list-style-type: none"> If a page fault occurs in accessing memory. If DS:RBX does not point to a valid TCS. If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	<p>If DS:RBX is not page aligned.</p> <p>If the enclave is not initialized.</p> <p>If the thread is not in the INACTIVE state.</p> <p>If CS, DS, ES or SS bases are not all zero.</p> <p>If executed in enclave mode.</p> <p>If part or all of the FS or GS segment specified by TCS is outside the DS segment.</p> <p>If any reserved field in the TCS FLAG is set.</p> <p>If the target address is not canonical.</p> <p>If CR4.OSFXSR = 0.</p> <p>If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.</p> <p>If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.</p> <p>If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory operands.</p> <p>If DS:RBX does not point to a valid TCS.</p> <p>If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.</p>

38.5 INTEL® SGX VIRTUALIZATION LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLV instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EDECVIRTCHILD—Decrement VIRTCHILDCNT in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLV[EDECVIRTCHILD]	IR	V/V	EAX[5]	This leaf function decrements the SECS VIRTCHILDCNT field.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDECVIRTCHILD (In)	Return error code (Out)	Address of an enclave page (In)	Address of an SECS page (In)

Description

This instruction decrements the SECS VIRTCHILDCNT field. This instruction can only be executed when current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

EDECVIRTCHILD Memory Parameter Semantics

EPCPAGE	SECS
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EDECVIRTCHILD Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A page fault occurs in accessing memory operands.
DS segment is unusable (32b mode).	RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
A memory address is in a non-canonical form (64b mode).	RCX does not refer to an SECS page.
A memory operand is not properly aligned.	RBX does not refer to an enclave page associated with SECS referenced in RCX.

Concurrency Restrictions

Table 38-76. Base Concurrency Restrictions of EDECVIRTCHILD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDECVIRTCHILD	Target [DS:RBX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS [DS:RCX]	Concurrent		

Table 38-77. Additional Concurrency Restrictions of EDECVIRTCHILD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDECVIRTCHILD	Target [DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation**Temp Variables in EDECVIRTCHILD Operational Flow**

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_VIRTCHILDCNT	Integer	64	Number of virtual child pages.

EDECVIRTCHILD Return Value in RAX

Error	Value	Description
No Error	0	EDECVIRTCHILD Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.
SGX_INVALID_COUNTER		Attempt to decrement counter that is already zero.

(* check alignment of DS:RBX *)

```
IF (DS:RBX is not 4K aligned) THEN
  #GP(0); FI;
```

(* check DS:RBX is a linear address of an EPC page *)

```
IF (DS:RBX does not resolve within an EPC) THEN
  #PF(DS:RBX, PFEC.SGX); FI;
```

(* check DS:RCX is a linear address of an EPC page *)

```
IF (DS:RCX does not resolve within an EPC) THEN
  #PF(DS:RCX, PFEC.SGX); FI;
```

(* Check the EPCPAGE for concurrency *)

```
IF (EPCPAGE is being modified) THEN
  RFLAGS.ZF = 1;
  RAX = SGX_EPC_PAGE_CONFLICT;
  goto DONE;
FI;
```

(* check that the EPC page is valid *)

```
IF (EPCM(DS:RBX).VALID = 0) THEN
  #PF(DS:RBX, PFEC.SGX); FI;
```

(* check that the EPC page has the correct type and that the back pointer matches the pointer passed as the pointer to parent *)

```
IF ((EPCM(DS:RBX).PAGE_TYPE = PT_REG) or
    (EPCM(DS:RBX).PAGE_TYPE = PT_TCS) or
```



```

(EPCM(DS:RBX).PAGE_TYPE = PT_TRIM) or
(EPCM(DS:RBX).PAGE_TYPE = PT_SS_FIRST) or
(EPCM(DS:RBX).PAGE_TYPE = PT_SS_REST))
  THEN
    (* get the SECS of DS:RBX *)
    TMP_SECS := Address of SECS for (DS:RBX);
ELSE IF (EPCM(DS:RBX).PAGE_TYPE = PT_SECS) THEN
    (* get the physical address of DS:RBX *)
    TMP_SECS := Physical_Address(DS:RBX);
ELSE
    (* EDECVIRTUALD called on page of incorrect type *)
    #PF(DS:RBX, PFEC.SGX); FI;

IF (TMP_SECS ≠ Physical_Address(DS:RCX)) THEN
    #GP(0); FI;

(* Atomically decrement virtchild counter and check for underflow *)
Locked_Decrement(SECS(TMP_SECS).VIRTCHILDCNT);
IF (There was an underflow) THEN
    Locked_Increment(SECS(TMP_SECS).VIRTCHILDCNT);
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_COUNTER;
    goto DONE;
FI;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;

```

Flags Affected

ZF is set if EDECVIRTUALD fails due to concurrent operation with another SGX instruction, or if there is a VIRTCHILDCNT underflow. Otherwise cleared.

Protected Mode Exceptions

#GP(0)	<p>If a memory operand effective address is outside the DS segment limit.</p> <p>If DS segment is unusable.</p> <p>If a memory operand is not properly aligned.</p> <p>RBX does not refer to an enclave page associated with SECS referenced in RCX.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory operands.</p> <p>If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).</p> <p>If RCX does not refer to an SECS page.</p>

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form. If a memory operand is not properly aligned. RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands. If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS). If RCX does not refer to an SECS page.

EINCVIRTCHILD—Increment VIRTCHILDCNT in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLV[EINCVIRTCHILD]	IR	V/V	EAX[5]	This leaf function increments the SECS VIRTCHILDCNT field.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EINCVIRTCHILD (In)	Return error code (Out)	Address of an enclave page (In)	Address of an SECS page (In)

Description

This instruction increments the SECS VIRTCHILDCNT field. This instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create a linear address. Segment override is not supported.

EINCVIRTCHILD Memory Parameter Semantics

EPCPAGE	SECS
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EINCVIRTCHILD Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A page fault occurs in accessing memory operands.
DS segment is unusable (32b mode).	RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
A memory address is in a non-canonical form (64b mode).	RCX does not refer to an SECS page.
A memory operand is not properly aligned.	RBX does not refer to an enclave page associated with SECS referenced in RCX.

Concurrency Restrictions

Table 38-78. Base Concurrency Restrictions of EINCVIRTCHILD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EINCVIRTCHILD	Target [DS:RBX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS [DS:RCX]	Concurrent		

Table 38-79. Additional Concurrency Restrictions of EINCVRTCHILD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EINCVRTCHILD	Target [DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation**Temp Variables in EINCVRTCHILD Operational Flow**

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.

EINCVRTCHILD Return Value in RAX

Error	Value	Description
No Error	0	EINCVRTCHILD Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.

(* check alignment of DS:RBX *)
IF (DS:RBX is not 4K aligned) THEN
 #GP(0); FI;

(* check DS:RBX is an linear address of an EPC page *)
IF (DS:RBX does not resolve within an EPC) THEN
 #PF(DS:RBX, PFEC.SGX); FI;

(* check DS:RCX is an linear address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
 #PF(DS:RCX, PFEC.SGX); FI;

(* Check the EPCPAGE for concurrency *)
IF (EPCPAGE is being modified) THEN
 RFLAGS.ZF = 1;
 RAX = SGX_EPC_PAGE_CONFLICT;
 goto DONE;
FI;

(* check that the EPC page is valid *)
IF (EPCM(DS:RBX).VALID = 0) THEN
 #PF(DS:RBX, PFEC.SGX); FI;

(* check that the EPC page has the correct type and that the back pointer matches the pointer passed as the pointer to parent *)
IF ((EPCM(DS:RBX).PAGE_TYPE = PT_REG) or
(EPCM(DS:RBX).PAGE_TYPE = PT_TCS) or
(EPCM(DS:RBX).PAGE_TYPE = PT_TRIM) or
(EPCM(DS:RBX).PAGE_TYPE = PT_SS_FIRST) or
(EPCM(DS:RBX).PAGE_TYPE = PT_SS_REST))

```

THEN
  (* get the SECS of DS:RBX *)
  TMP_SECS := Address of SECS for (DS:RBX);
ELSE IF (EPCM(DS:RBX).PAGE_TYPE = PT_SECS) THEN
  (* get the physical address of DS:RBX *)
  TMP_SECS := Physical_Address(DS:RBX);
ELSE
  (* EINCVIRTCHILD called on page of incorrect type *)
  #PF(DS:RBX, PFEC.SGX); FI;

IF (TMP_SECS ≠ Physical_Address(DS:RCX)) THEN
  #GP(0); FI;

(* Atomically increment vrtchild counter *)
Locked_Increment(SECS(TMP_SECS).VIRTCHILDCNT);

```

```

RFLAGS.ZF := 0;
RAX := 0;

```

```

DONE:
(* clear flags *)
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;

```

Flags Affected

ZF is set if EINCVIRTCHILD fails due to concurrent operation with another SGX instruction; otherwise cleared.

Protected Mode Exceptions

#GP(0)	<p>If a memory operand effective address is outside the DS segment limit.</p> <p>If DS segment is unusable.</p> <p>If a memory operand is not properly aligned.</p> <p>RBX does not refer to an enclave page associated with SECS referenced in RCX.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory operands.</p> <p>If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).</p> <p>If RCX does not refer to an SECS page.</p>

64-Bit Mode Exceptions

#GP(0)	<p>If a memory address is in a non-canonical form.</p> <p>If a memory operand is not properly aligned.</p> <p>RBX does not refer to an enclave page associated with SECS referenced in RCX.</p>
#PF(error code)	<p>If a page fault occurs in accessing memory operands.</p> <p>If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).</p> <p>If RCX does not refer to an SECS page.</p>

ESETCONTEXT—Set the ENCLAVECONTEXT Field in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLV[ESETCONTEXT]	IR	V/V	EAX[5]	This leaf function sets the ENCLAVECONTEXT field in SECS.

Instruction Operand Encoding

Op/En	EAX		RCX	RDX
IR	ESETCONTEXT (In)	Return error code (Out)	Address of the destination EPC page (In, EA)	Context Value (In, EA)

Description

The ESETCONTEXT leaf overwrites the ENCLAVECONTEXT field in the SECS. ECREATE and ELD of an SECS set the ENCLAVECONTEXT field in the SECS to the address of the SECS (for access later in ERDINFO). The ESETCONTEXT instruction allows a VMM to overwrite the default context value if necessary, for example, if the VMM is emulating ECREATE or ELD on behalf of the guest.

The content of RCX is an effective address of the SECS page to be updated, RDX contains the address pointing to the value to be stored in the SECS. The DS segment is used to create linear address. Segment override is not supported.

The instruction fails if:

- The operand is not properly aligned.
- RCX does not refer to an SECS page.

ESETCONTEXT Memory Parameter Semantics

EPCPAGE	CONTEXT
Read access permitted by Enclave	Read/Write access permitted by Non Enclave

The instruction faults if any of the following:

ESETCONTEXT Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A memory operand is not properly aligned.
DS segment is unusable (32b mode).	A page fault occurs in accessing memory operands.
A memory address is in a non-canonical form (64b mode).	

Concurrency Restrictions

Table 38-80. Base Concurrency Restrictions of ESETCONTEXT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ESETCONTEXT	SECS [DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	

Table 38-81. Additional Concurrency Restrictions of ESETCONTEXT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ESETCONTEXT	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ESETCONTEXT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_CONTEXT	CONTEXT	64	Data Value of CONTEXT.

ESETCONTEXT Return Value in RAX

Error	Value	Description
No Error	0	ESETCONTEXT Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.

(* check alignment of the EPCPAGE (RCX) *)

```
IF (DS:RCX is not 4KByte Aligned) THEN
    #GP(0); FI;
```

(* check that EPCPAGE (DS:RCX) is the address of an EPC page *)

```
IF (DS:RCX does not resolve within an EPC) THEN
    #PF(DS:RCX, PFEC.SGX); FI;
```

(* check alignment of the CONTEXT field (RDX) *)

```
IF (DS:RDX is not 8Byte Aligned) THEN
    #GP(0); FI;
```

(* Load CONTEXT into local variable *)

```
TMP_CONTEXT := DS:RDX
```

(* Check the EPC page for concurrency *)

```
IF (EPC page is being modified) THEN
    RFLAGS.ZF := 1;
    RFLAGS.CF := 0;
    RAX := SGX_EPC_PAGE_CONFLICT;
    goto DONE;
FI;
```

(* check page validity *)

```
IF (EPCM(DS:RCX).VALID = 0) THEN
    #PF(DS:RCX, PFEC.SGX);
FI;
```

(* check EPC page is an SECS page *)

```
IF (EPCM(DS:RCX).PT is not PT_SECS) THEN
  #PF(DS:RCX, PFEC.SGX);
FI;
```

```
(* load the context value into SECS(DS:RCX).ENCLAVECONTEXT *)
SECS(DS:RCX).ENCLAVECONTEXT := TMP_CONTEXT;
```

```
RAX := 0;
RFLAGS.ZF := 0;
```

```
DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is set if ESETCONTEXT fails due to concurrent operation with another SGX instruction; otherwise cleared. CF, PF, AF, OF, and SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit. If DS segment is unusable. If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form. If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

20. Updates to Chapter 40, Volume 3D

Change bars and green text show changes to Chapter 40 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Updates throughout the chapter for the AEX Notify feature.

CHAPTER 40

ENCLAVE CODE DEBUG AND PROFILING

Intel® SGX is architected to provide protection for production enclaves and permit enclave code developers to use an SGX-aware debugger to effectively debug a non-production enclave (debug enclave). Intel SGX also allows a non-SGX-aware debugger to debug non-enclave portions of the application without getting confused by enclave instructions.

40.1 CONFIGURATION AND CONTROLS

40.1.1 Debug Enclave vs. Production Enclave

The SECS of each enclave provides a bit, SECS.ATTRIBUTES.DEBUG, indicating whether the enclave is a debug enclave (if set) or a production enclave (if 0). If this bit is set, software outside the enclave can use EDBGWR/EDBGWR to access the EPC memory of the enclave. The value of DEBUG is not included in the measurement of the enclave and therefore doesn't require an alternate SIGSTRUCT to be generated to debug the enclave.

The ATTRIBUTES field in the SECS is reported in the enclave's attestation, and is included in the key derivation. Enclave secrets that were protected by the enclave using Intel SGX keys when it ran as a production enclave will not be accessible by the debug enclave. A debugger needs to be aware that special debug content might be required for a debug enclave to run in a meaningful way.

EPC memory belonging to a debug enclave can be accessed via the EDBGWR/EDBGWR leaf functions (see Section 38.4), while that belonging to a non-debug enclave cannot be accessed by these leaf functions.

40.1.2 Tool-Chain Opt-in

The TCS.FLAGS.DBGOPTIN bit controls interactions of certain debug and profiling features with enclaves, including code/data breakpoints, TF, RF, monitor trap flag, BTF, LBRs, BTM, BTS, Intel Processor Trace, and performance monitoring. This bit is forced to zero when EPC pages are added via EADD. A debugger can set this bit via EDBGWR to the TCS of a debug enclave.

An enclave entry through a TCS with the TCS.FLAGS.DBGOPTIN set to 0 is called an **opt-out entry**. Conversely, an enclave entry through a TCS with TCS.FLAGS.DBGOPTIN set to 1 is called an **opt-in entry**.

40.1.3 Debugging an Enclave That Uses Asynchronous Enclave Exit Notify

Whenever an opt-in enclave entry is used to perform enclave code debugging or profiling, the debugger or profiling tool may clear TCS.FLAGS.AEXNOTIFY to prevent AEX notifications from being delivered at each interrupt, breakpoint, trap, or other exception.

40.2 SINGLE STEP DEBUG

40.2.1 Single Stepping ENCLS Instruction Leafs

If the RFLAGS.TF bit is set at the beginning of ENCLS, then a single-step debug exception is pending as a trap-class exception on the instruction boundary immediately after the ENCLS instruction. Additionally, if the instruction is executed in VMX non-root operation and the "monitor trap flag" VM-execution control is 1, an MTF VM exit is pending on the instruction boundary immediately after the instruction if the instruction does not fault.

40.2.2 Single Stepping ENCLU Instruction Leafs

The interactions of the unprivileged Intel SGX instruction ENCLU are leaf dependent.

An enclave entry via EENTER/ERESUME leaf functions of the ENCLU, in certain cases, may mask the RFLAGS.TF bit, and mask the setting of the “monitor trap flag” VM-execution control. In such situations, an exit from the enclave, either via the EEXIT leaf function or via an AEX unmasks the RFLAGS.TF bit and the “monitor trap flag” VM-execution control. The details of this masking/unmasking and the pending of single stepping events across EENTER/ERESUME/EEXIT/AEX are covered in detail in Section 40.2.3.

If the EFLAGS.TF bit is set at the beginning of EREPORT or EGETKEY leafs, and if the EFLAGS.TF is not masked by the preceding enclave entry, then a single-step debug exception is pending on the instruction boundary immediately after the ENCLU instruction. Additionally, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, and if the monitor trap flag is not masked by the preceding enclave entry, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

If the instruction under consideration results in a fault, then the control flow goes to the fault handler, and no single-step debug exception is asserted. In such a situation, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending after the delivery of the fault (or any nested exception). No MTF VM exit occurs if another VM exit occurs before reaching that boundary on which an MTF VM exit would be pending.

40.2.3 Single-Stepping Enclave Entry with Opt-out Entry

40.2.3.1 Single Stepping without AEX

Figure 40-1 shows the most common case for single-stepping after an opt-out entry.

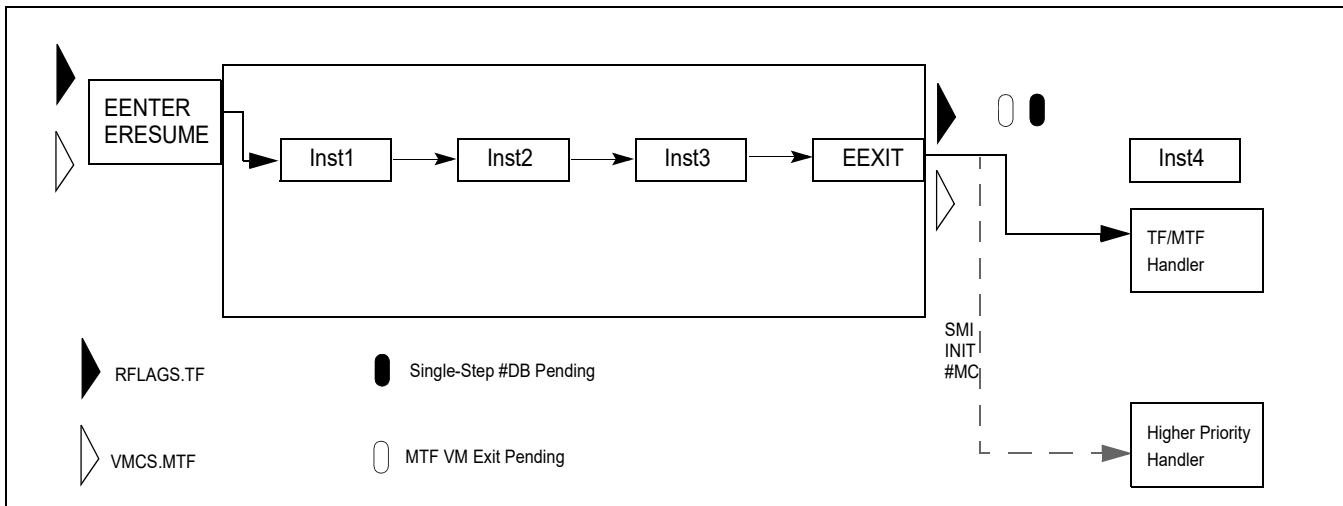


Figure 40-1. Single Stepping with Opt-out Entry - No AEX

In this scenario, if the RFLAGS.TF bit is set at the time of the enclave entry, then a single step debug exception is pending on the instruction boundary after EEXIT. Additionally, if the enclave is executing in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after EEXIT.

The value of the RFLAGS.TF bit at the end of EEXIT is the same as the value of RFLAGS.TF at the time of the enclave entry.

40.2.3.2 Single Step Preempted by AEX Due to Non-SMI Event

Figure 40-2 shows the interaction of single stepping with AEX due to a non-SMI event after an opt-out entry.

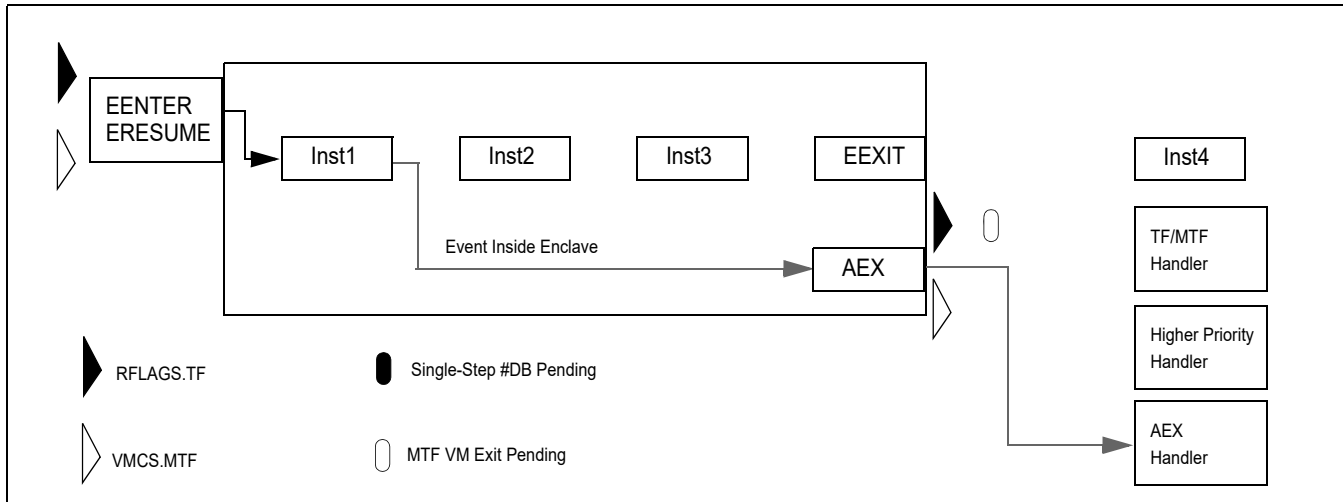


Figure 40-2. Single Stepping with Opt-out Entry -AEX Due to Non-SMI Event Before Single-Step Boundary

In this scenario, if the enclave is executing in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after the AEX. No MTF VM exit occurs if another VM exit happens before reaching that instruction boundary.

The value of the **RFLAGS.TF** bit at the end of AEX is the same as the value of **RFLAGS.TF** at the time of the enclave entry.

40.2.4 RFLAGS.TF Treatment on AEX

The value of **EFLAGS.TF** at the end of AEX from an opt-out enclave is same as the value of **EFLAGS.TF** at the time of the enclave entry. The value of **EFLAGS.TF** at the end of AEX from an opt-in enclave is unmodified. The **EFLAGS.TF** saved in GPR portion of the SSA on an AEX is 0. For more detail see **EENTER** and **ERESUME** in Chapter 5.

40.2.5 Restriction on Setting of TF after an Opt-Out Entry

Enclave entered through an opt-out entry is not allowed to set **EFLAGS.TF**. The **POPF** instruction forces **RFLAGS.TF** to 0 if the enclave was entered through opt-out entry.

40.2.6 Trampoline Code Considerations

Any AEX from the enclave which results in the **RFLAGS.TF = 1** on the reporting stack will result in a single-step #DB after the first instruction of the trampoline code if the trampoline is entered using the **IRET** instruction.

40.3 CODE AND DATA BREAKPOINTS

40.3.1 Breakpoint Suppression

Following an opt-out entry:

- Instruction breakpoints are suppressed during execution in an enclave.
- Data breakpoints are not triggered on accesses to the address range defined by **ELRANGE**.
- Data breakpoints are triggered on accesses to addresses outside the **ELRANGE**

Following an opt-in entry instruction and data breakpoints are not suppressed.

The processor does not report any matches on debug breakpoints that are suppressed on enclave entry. However, the processor does not clear any bits in DR6 that were already set at the time of the enclave entry.

40.3.2 Reporting of Instruction Breakpoint on Next Instruction on a Debug Trap

A debug exception caused by the single-step execution mode or when a data breakpoint condition was met causes the processor to perform an AEX. Following such an AEX, the processor reports in the debug status register (DR6) matches of the new instruction pointer (the AEP address) in a breakpoint address register setup to detect instruction execution.

40.3.3 RF Treatment on AEX

RF flag value saved in SSA is the same as what would have been pushed on stack if the exception or event causing the AEX occurred when executing outside an enclave (see Section 18.3.1.1). Following an AEX, the RF flag is 0 in the synthetic state.

40.3.4 Breakpoint Matching in Intel® SGX Instruction Flows

Implicit accesses made by Intel SGX instructions to EPC regions do not trigger data breakpoints. Explicit accesses made by ENCLS[ECREATE], ENCLS[EADD], ENCLS[EEXTEND], ENCLS[EINIT], ENCLS[EREMOVE], ENCLS[ETRACK], ENCLS[EBLOCK], ENCLS[EPA], ENCLS[EWB], ENCLS[ELD], ENCLS[EDBGGRD], ENCLS[EDBGWR], ENCLU[EENTER], and ENCLU[ERESUME] to the EPC operands do not trigger data breakpoints.

Explicit accesses made by the Intel SGX instructions (ENCLU[EGETKEY] and ENCLU[EREPORT]) executed by an enclave following an opt-in entry, trigger data breakpoints on accesses to their EPC operands. All Intel SGX instructions trigger data breakpoints on accesses to their non-EPC operands.

40.4 CONSIDERATION OF THE INT1 AND INT3 INSTRUCTIONS

This section considers the operation of the INT1 and INT3 instructions when executed inside an enclave. These are the instructions with opcodes F1 and CC, respectively, and not INT *n* (with opcode CD) with value 1 or 3 for *n*.

40.4.1 Behavior of INT1 and INT3 Inside an Enclave

An execution of either INT1 or INT3 inside an enclave results in a fault-class exception. Following an opt-out entry, execution of either instruction results in an invalid-opcode exception (#UD). Following opt-in entry, INT1 results in a debug exception (#DB) and INT3 delivers a breakpoint exception (#BP). The normal requirement for INT3 (that the CPL not be greater than the DPL of descriptor 3 in the IDT) is not enforced.

Because execution of INT1 or INT3 inside an enclave results in a fault, the RIP saved in the SSA on AEX references the INT1 or INT3 instruction (and not the following instruction). The RIP value saved on the stack (or in the TSS or VMCS) is that of the AEP.

If execution of INT1 or INT3 inside an enclave causes a VM exit, the event type in the VM-exit interruption information field indicates a hardware exception (type 3),¹ and the VM-exit instruction length field is saved as zero.

40.4.2 Debugger Considerations

A debugger using INT3 inside an enclave should account for the modified behavior described in Section 40.4.1. Because INT3 is fault-like inside an enclave, the RIP saved in the SSA on AEX is that of the INT3 instruction. Conse-

1. INT1 would normally indicate a privileged software exception (type 5), and INT3 would normally indicate a software exception (type 6).

quently, the debugger must not decrement SSA.RIP for #BP coming from an enclave to re-execute the instruction at the RIP of the INT3 instruction on a subsequent enclave entry.

40.4.3 VMM Considerations

As described in Section 40.4.1, execution of INT3 inside an enclave delivers #BP with “interruption type” of 3. A VMM that re-injects #BP into the guest should establish the VM-entry interruption information field using data saved into the appropriate VMCS fields by the VM exit incident to the #BP (as recommended in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C).

VMMs that create the VM-entry interruption information based solely on the exception vector should take care to use event type 3 (instead of 6) when they detect a VM exit incident to enclave mode that is due to an exception with vector 3.

40.5 BRANCH TRACING

40.5.1 BTF Treatment

When software enables single-stepping on branches then:

- Following an opt-in entry using EENTER the processor generates a single step debug exception.
- Following an EEXIT the processor generates a single-step debug exception

Enclave entry using ERESUME (opt-in or opt-out) and an AEX from the enclave do not cause generation of the single-step debug exception.

40.5.2 LBR Treatment

40.5.2.1 LBR Stack on Opt-in Entry

Following an opt-in entry into an enclave, last branch recording facilities if enabled continued to store branch records in the LBR stack MSRs as follows:

- On enclave entry using EENTER/ERESUME, the processor push the address of EENTER/ERESUME instruction into MSR_LASTBRANCH_n_FROM_IP, and the destination address of the EENTER/ERESUME into MSR_LASTBRANCH_n_TO_IP.
- On EEXIT, the processor pushes the address of EEXIT instruction into MSR_LASTBRANCH_n_FROM_IP, and the address of EEXIT destination into MSR_LASTBRANCH_n_TO_IP.
- On AEX, the processor pushes RIP saved in the SSA into MSR_LASTBRANCH_n_FROM_IP, and the address of AEP into MSR_LASTBRANCH_n_TO_IP.
- For every branch inside the enclave, a branch record is pushed on the LBR stack.

Figure 40-3 shows an example of LBR stack manipulation after an opt-in entry. Every arrow in this picture indicates a branch record pushed on the LBR stack. The “From IP” of the branch record contains the linear address of the instruction located at the start of the arrow, while the “To IP” of the branch record contains the linear address of the instruction at the end of the arrow.

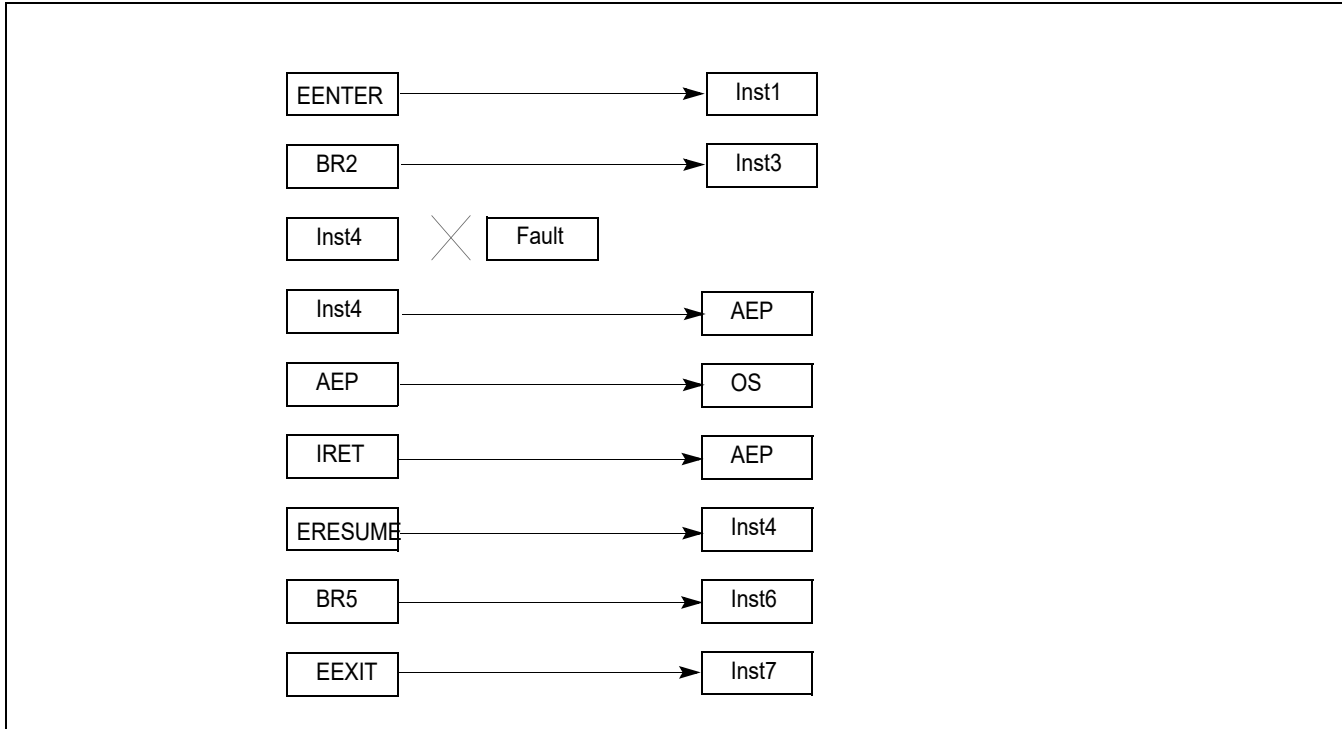


Figure 40-3. LBR Stack Interaction with Opt-in Entry

40.5.2.2 LBR Stack on Opt-out Entry

An opt-out entry into an enclave suppresses last branch recording facilities, and enclave exit after an opt-out entry un-suppresses last branch recording facilities.

Opt-out entry into an enclave does not push any record on LBR stack.

If last branch recording facilities were enabled at the time of enclave entry, then EEXIT following such an enclave entry pushes one record on LBR stack. The `MSR_LASTBRANCH_n_FROM_IP` of such record holds the linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while the `MSR_LASTBRANCH_n_TO_IP` of such record holds linear address of the destination of EEXIT.

Additionally, if last branch recording facilities were enabled at the time of enclave entry, then an AEX after such an entry pushes one record on LBR stack, before pushing record for the event causing the AEX if the event pushes a record on LBR stack. The `MSR_LASTBRANCH_n_FROM_IP` of the new record holds linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while `MSR_LASTBRANCH_n_TO_IP` of the new record holds linear address of the AEP. If the event causing AEX pushes a record on LBR stack, then the `MSR_LASTBRANCH_n_FROM_IP` for that record holds linear address of the AEP.

Figure 40-4 shows an example of LBR stack manipulation after an opt-out entry. Every arrow in this picture indicates a branch record pushed on the LBR stack. The "From IP" of the branch record contains the linear address of the instruction located at the start of the arrow, while the "To IP" of the branch record contains the linear address of the instruction at the end of the arrow.

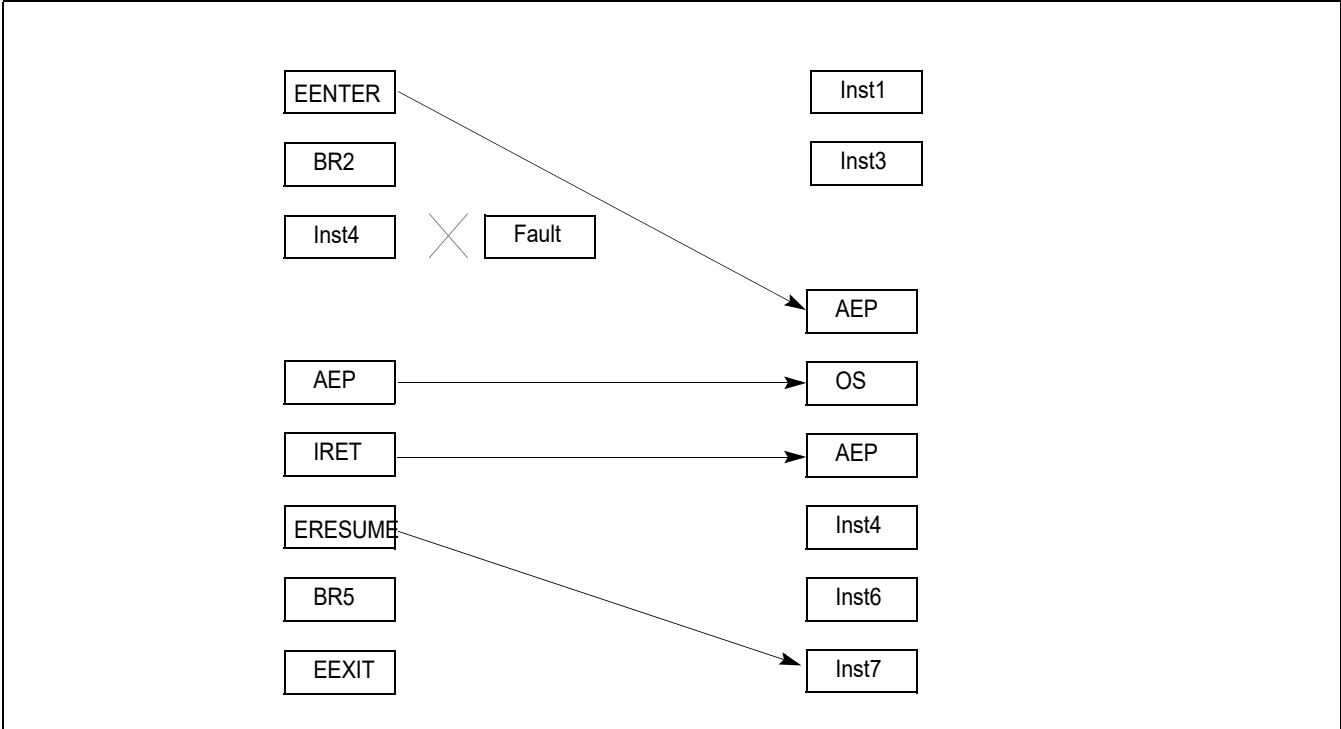


Figure 40-4. LBR Stack Interaction with Opt-out Entry

40.5.2.3 Mispredict Bit, Record Type, and Filtering

All branch records resulting from Intel SGX instructions/AEXs are reported as predicted branches, and consequently, bit 63 of MSR_LASTBRANCH_n_FROM_IP for such records is set. Branch records due to these Intel SGX operations are always non-HLE/non-RTM records.

EENTER, ERESUME, EEXIT, and AEX are considered to be far branches. Consequently, bit 8 in MSR_LBR_SELECT controls filtering of the new records introduced by Intel SGX.

40.6 INTERACTION WITH PERFORMANCE MONITORING

40.6.1 IA32_PERF_GLOBAL_STATUS Enhancement

On processors supporting Intel SGX, the IA32_PERF_GLOBAL_STATUS MSR provides a bit indicator, known as "Anti Side-channel Interference" (ASCI) at bit position 60. If this bit is 0, the performance monitoring data in various performance monitoring counters are accumulated normally as defined by relevant architectural/microarchitectural conditions. If the ASCI bit is set, the contents in various performance monitoring counters can be affected by the direct or indirect consequence of Intel SGX protection of enclave code executing in the processor.

40.6.2 Performance Monitoring with Opt-in Entry

An opt-in enclave entry allow performance monitoring logic to observe the contribution of enclave code executing in the processor. Thus the contents of performance monitoring counters does not distinguish between contribution originating from enclave code or otherwise. All counters, events, precise events, etc. continue to work as defined in the IA32/Intel 64 Software Developer Manual. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is not set.

40.6.3 Performance Monitoring with Opt-out Entry

In general, performance monitoring activities are suppressed when entering an opt-out enclave. This applies to all thread-specific, configured performance monitoring, except for the cycle-counting fixed counter, IA32_FIXED_CTR1 and IA32_FIXED_CTR2. Upon entering an opt-out enclave, IA32_FIXED_CTR0, IA32_PMCx will stop accumulating counts. Additionally, if PEBS is configured to capture PEBS record for this thread, PEBS record generation will also be suppressed. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is set.

Performance monitoring on the sibling thread may also be affected. Any one of IA32_FIXED_CTRx or IA32_PMCx on the sibling thread configured to monitor thread-specific eventing logic with AnyThread = 1 is demoted to count only MyThread while an opt-out enclave is executing on the other thread.

40.6.4 Enclave Exit and Performance Monitoring

When a logical processor exits an enclave, either via ENCLU[EEXIT] or via AEX, all performance monitoring activity (including PEBS) on that logical processor that was suppressed is unsuppressed.

Any counters that were demoted from AnyThread to MyThread on the sibling thread are promoted back to AnyThread.

40.6.5 PEBS Record Generation on Intel® SGX Instructions

All leaf functions of the ENCLS instruction report “Eventing RIP” of the ENCLS instruction if a PEBS record is generated at the end of the instruction execution. Additionally, the EGETKEY and EREPORT leaf functions of the ENCLU instruction report “Eventing RIP” of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution.

If the EENTER and ERESUME leaf functions are performing an opt-in entry report “Eventing RIP” of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution. On the other hand, if these leaf functions are performing an opt-out entry, then these leaf functions result in PEBS being suppressed, and no PEBS record is generated at the end of these instructions.

A PEBS record is generated if there is a PEBS event pending at the end of EEXIT (due to a counter overflowing during enclave execution or during EEXIT execution). This PEBS record contains the architectural state of the logical processor at the end of EEXIT. If the enclave was entered via an opt-in entry, then this record reports the “Eventing RIP” as the linear address of the ENCLU[EEXIT] instruction. If the enclave was entered via an opt-out entry, then the record reports the “Eventing RIP” as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A PEBS record is generated after the AEX if there is a PEBS event pending at the end of AEX (due to a counter overflowing during enclave execution or during AEX execution). This PEBS record contains the synthetic state of the logical processor that is established at the end of AEX. For opt-in entry, this record has the EVENTING_RIP set to the RIP saved in the SSA. For opt-out entry, the record has the EVENTING_RIP set to the linear address of EENTER/ERESUME used for the last enclave entry.

If the enclave was entered via an opt-in entry, then this record reports the “Eventing RIP” as the linear address in the SSA of the enclave (a.k.a., the “Eventing LIP” inside the enclave). If the enclave was entered via an opt-out entry, then the record reports the “Eventing RIP” as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A second PEBS event may be pended during the Enclave Exiting Event (EEE). If the PEBS event is taken at the end of delivery of the EEE then the “Eventing RIP” in this second PEBS record is the linear address of the AEP.

40.6.6 Exception-Handling on PEBS/BTS Loads/Stores after AEX

As noted in Section 18.4.9.2, recording in the BTS buffer or in the PEBS buffer may not operate properly if accesses to any of the DS save area sections cause page faults or VM exits. Such page faults or VM exits, if they occur, are delivered immediately to the OS or VMM, and generation of a BTS or PEBS record is skipped and may leave the buffers in a state where they have a partial BTS or PEBS records.

However, any events that are detected during PEBS/BTS record generation at the end of AEX and before delivering the Enclave Exiting Event (EEE) cannot be reported immediately to the OS/VMM, as an event window is not open at

the end of AEX. Consequently, fault-like events such as page faults, EPT faults, EPT mis-configuration, and accesses to APIC-access page detected on stores to the PEBS/BTS buffer are not reported, and generation of the PEBS and/or BTS record at the end of AEX is aborted (this may leave the buffers in a state where they have partial PEBS or BTS records). Trap-like events detected on stores to the PEBS/BTS buffer (such as debug traps) are pended until the next instruction boundary, where they are handled according to the architecturally defined priority. The processor continues the handling of the Enclave Exiting Event (SMI, NMI, interrupt, exception delivery, VM exit, etc.) after aborting the PEBS/BTS record generation.

40.6.6.1 Other Interactions with Performance Monitoring

For opt-in entry, EENTER, ERESUME, EEXIT, and AEX are all treated as predicted far branches, and any counters that are counting such branches are incremented by 1 as a part of retirement of these instructions. Retirement of these instructions is also counted in any counters configured to count instructions retired.

For opt-out entry, execution inside an enclave is treated as a single predicted branch, and all branch-counting performance monitoring counters are incremented accordingly. Additionally, such execution is also counted as a single instruction, and all performance monitoring counters counting instructions are incremented accordingly.

Enclave entry does not affect any performance monitoring counters shared between cores.

21. Updates to Appendix B, Volume 3D

Change bars and green text show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Added instruction-timeout control to Table B-8, "Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)."

APPENDIX B

FIELD ENCODING IN VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 25.11.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.).

B.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 25.11.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.1.1 16-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-1 enumerates the 16-bit control fields.

Table B-1. Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID) ¹	00000000B	00000000H
Posted-interrupt notification vector ²	00000001B	00000002H
EPTP index ³	00000010B	00000004H
HLAT prefix size ⁴	00000011B	00000006H
Last PID-pointer index ⁵	00000100B	00000008H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable VPID” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “IPI virtualization” VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES selector	00000000B	0000800H
Guest CS selector	00000001B	0000802H
Guest SS selector	00000010B	0000804H
Guest DS selector	00000011B	0000806H
Guest FS selector	00000100B	0000808H

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH
Guest TR selector	000000111B	0000080EH
Guest interrupt status ¹	000001000B	00000810H
PML index ²	000001001B	00000812H
Guest UINV ³	000001010B	00000814H

NOTES:

1. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of either the “clear UINV” VM-exit control or the “load UINV” VM-entry control.

B.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-3 enumerates the 16-bit host-state fields.

Table B-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host ES selector	000000000B	00000C00H
Host CS selector	000000001B	00000C02H
Host SS selector	000000010B	00000C04H
Host DS selector	000000011B	00000C06H
Host FS selector	000000100B	00000C08H
Host GS selector	000000101B	00000C0AH
Host TR selector	000000110B	00000C0CH

B.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 25.11.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)		00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)		00002003H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Address of MSR bitmaps (full) ¹	000000010B	00002004H
Address of MSR bitmaps (high) ¹		00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)		00002007H
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)		00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)		0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)		0000200DH
PML address (full) ²	000000111B	0000200EH
PML address (high) ²		0000200FH
TSC offset (full)	000001000B	00002010H
TSC offset (high)		00002011H
Virtual-APIC address (full) ³	000001001B	00002012H
Virtual-APIC address (high) ³		00002013H
APIC-access address (full) ⁴	000001010B	00002014H
APIC-access address (high) ⁴		00002015H
Posted-interrupt descriptor address (full) ⁵	000001011B	00002016H
Posted-interrupt descriptor address (high) ⁵		00002017H
VM-function controls (full) ⁶	000001100B	00002018H
VM-function controls (high) ⁶		00002019H
EPT pointer (EPTP; full) ⁷	000001101B	0000201AH
EPT pointer (EPTP; high) ⁷		0000201BH
EOI-exit bitmap 0 (EOI_EXIT0; full) ⁸	000001110B	0000201CH
EOI-exit bitmap 0 (EOI_EXIT0; high) ⁸		0000201DH
EOI-exit bitmap 1 (EOI_EXIT1; full) ⁸	000001111B	0000201EH
EOI-exit bitmap 1 (EOI_EXIT1; high) ⁸		0000201FH
EOI-exit bitmap 2 (EOI_EXIT2; full) ⁸	000010000B	00002020H
EOI-exit bitmap 2 (EOI_EXIT2; high) ⁸		00002021H
EOI-exit bitmap 3 (EOI_EXIT3; full) ⁸	000010001B	00002022H
EOI-exit bitmap 3 (EOI_EXIT3; high) ⁸		00002023H
EPTP-list address (full) ⁹	000010010B	00002024H
EPTP-list address (high) ⁹		00002025H
VMREAD-bitmap address (full) ¹⁰	000010011B	00002026H
VMREAD-bitmap address (high) ¹⁰		00002027H
VMWRITE-bitmap address (full) ¹⁰	000010100B	00002028H
VMWRITE-bitmap address (high) ¹⁰		00002029H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Virtualization-exception information address (full) ¹¹	000010101B	0000202AH
Virtualization-exception information address (high) ¹¹		0000202BH
XSS-exiting bitmap (full) ¹²	000010110B	0000202CH
XSS-exiting bitmap (high) ¹²		0000202DH
ENCLS-exiting bitmap (full) ¹³	000010111B	0000202EH
ENCLS-exiting bitmap (high) ¹³		0000202FH
Sub-page-permission-table pointer (full) ¹⁴	000011000B	00002030H
Sub-page-permission-table pointer (high) ¹⁴		00002031H
TSC multiplier (full) ¹⁵	000011001B	00002032H
TSC multiplier (high) ¹⁵		00002033H
Tertiary processor-based VM-execution controls (full) ¹⁶	000011010B	00002034H
Tertiary processor-based VM-execution controls (high) ¹⁶		00002035H
ENCLV-exiting bitmap (full) ¹⁷	000011011B	00002036H
ENCLV-exiting bitmap (high) ¹⁷		00002037H
Low PASID directory address (full) ¹⁸	000011100B	00002038H
Low PASID directory address (high) ¹⁸		00002039H
High PASID directory address (full) ¹⁸	000011101B	0000203AH
High PASID directory address (high) ¹⁸		0000203BH
Shared EPT pointer (full) ¹⁹	000011110B	0000203CH
Shared EPT pointer (high) ¹⁹		0000203DH
PCONFIG-exiting bitmap (full) ²⁰	000011111B	0000203EH
PCONFIG-exiting bitmap (high) ²⁰		0000203FH
Hypervisor-managed linear-address translation pointer (HLATP; full) ²¹	000100000B	00002040H
HLATP (high) ²¹		00002041H
PID-pointer table address (full) ²²	000100001B	00002042H
PID-pointer table address (high) ²²		00002043H
Secondary VM-exit controls (full) ²³	000100010B	00002044H
Secondary VM-exit controls (high) ²³		00002045H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use MSR bitmaps” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
6. This field exists only on processors that support the 1-setting of the “enable VM functions” VM-execution control.
7. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
8. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
9. This field exists only on processors that support the 1-setting of the “EPTP switching” VM-function control.
10. This field exists only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control.
11. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
12. This field exists only on processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control.

13. This field exists only on processors that support the 1-setting of the “enable ENCLS exiting” VM-execution control.
14. This field exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.
15. This field exists only on processors that support the 1-setting of the “use TSC scaling” VM-execution control.
16. This field exists only on processors that support the 1-setting of the “activate tertiary controls” VM-execution control.
17. This field exists only on processors that support the 1-setting of the “enable ENCLV exiting” VM-execution control.
18. This field exists only on processors that support the 1-setting of the “PASID translation” VM-execution control.
19. This field exists only on processors that support the 1-setting of the “shared-EPTP” VM-execution control.
20. This field exists only on processors that support the 1-setting of the “enable PCONFIG” VM-execution control.
21. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.
22. This field exists only on processors that support the 1-setting of the “IPI virtualization” VM-execution control.
23. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.

B.2.2 64-Bit Read-Only Data Field

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. There is only one such 64-bit field as given in Table B-5. (As with other 64-bit fields, this one has two encodings.)

Table B-5. Encodings for 64-Bit Read-Only Data Field (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full) ¹	000000000B	00002400H
Guest-physical address (high) ¹		00002401H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)		00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H
Guest IA32_DEBUGCTL (high)		00002803H
Guest IA32_PAT (full) ¹	000000010B	00002804H
Guest IA32_PAT (high) ¹		00002805H
Guest IA32_EFER (full) ²	000000011B	00002806H
Guest IA32_EFER (high) ²		00002807H
Guest IA32_PERF_GLOBAL_CTRL (full) ³	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high) ³		00002809H
Guest PDPTE0 (full) ⁴	000000101B	0000280AH
Guest PDPTE0 (high) ⁴		0000280BH

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Guest PDPTE1 (full) ⁴	000000110B	0000280CH
Guest PDPTE1 (high) ⁴		0000280DH
Guest PDPTE2 (full) ⁴	000000111B	0000280EH
Guest PDPTE2 (high) ⁴		0000280FH
Guest PDPTE3 (full) ⁴	000001000B	00002810H
Guest PDPTE3 (high) ⁴		00002811H
Guest IA32_BNDCFGS (full) ⁵	000001001B	00002812H
Guest IA32_BNDCFGS (high) ⁵		00002813H
Guest IA32_RTIT_CTL (full) ⁶	000001010B	00002814H
Guest IA32_RTIT_CTL (high) ⁶		00002815H
Guest IA32_LBR_CTL (full) ⁷	000001011B	00002816H
Guest IA32_LBR_CTL (high) ⁷		00002817H
Guest IA32_PKRS (full) ⁸	000001100B	00002818H
Guest IA32_PKRS (high) ⁸		00002819H

NOTES:

1. This field exists only on processors that support either the 1-setting of the “load IA32_PAT” VM-entry control or that of the “save IA32_PAT” VM-exit control.
2. This field exists only on processors that support either the 1-setting of the “load IA32_EFER” VM-entry control or that of the “save IA32_EFER” VM-exit control.
3. This field exists only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.
4. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
5. This field exists only on processors that support either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control.
6. This field exists only on processors that support either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control.
7. This field exists only on processors that support either the 1-setting of the “load IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control.
8. This field exists only on processors that support the 1-setting of the “load PKRS” VM-entry control.

B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full) ¹	000000000B	00002C00H
Host IA32_PAT (high) ¹		00002C01H
Host IA32_EFER (full) ²	000000001B	00002C02H
Host IA32_EFER (high) ²		00002C03H
Host IA32_PERF_GLOBAL_CTRL (full) ³	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high) ³		00002C05H

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Host IA32_PKRS (full) ⁴	000000011B	00002C06H
Host IA32_PKRS (high) ⁴		00002C07H

NOTES:

1. This field exists only on processors that support the 1-setting of the “load IA32_PAT” VM-exit control.
2. This field exists only on processors that support the 1-setting of the “load IA32_EFER” VM-exit control.
3. This field exists only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
4. This field exists only on processors that support the 1-setting of the “load PKRS” VM-exit control.

B.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 25.11.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-8 enumerates the 32-bit control fields.

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Pin-based VM-execution controls	000000000B	00004000H
Primary processor-based VM-execution controls	000000001B	00004002H
Exception bitmap	000000010B	00004004H
Page-fault error-code mask	000000011B	00004006H
Page-fault error-code match	000000100B	00004008H
CR3-target count	000000101B	0000400AH
Primary VM-exit controls	000000110B	0000400CH
VM-exit MSR-store count	000000111B	0000400EH
VM-exit MSR-load count	000001000B	00004010H
VM-entry controls	000001001B	00004012H
VM-entry MSR-load count	000001010B	00004014H
VM-entry interruption-information field	000001011B	00004016H
VM-entry exception error code	000001100B	00004018H
VM-entry instruction length	000001101B	0000401AH
TPR threshold ¹	000001110B	0000401CH
Secondary processor-based VM-execution controls ²	000001111B	0000401EH
PLE_Gap ³	000010000B	00004020H
PLE_Window ³	000010001B	00004022H
Instruction-timeout control ⁴	000010010B	00004024H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

2. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “instruction timeout” VM-execution control.

B.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-9 enumerates the 32-bit read-only data fields.

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
VM-instruction error	00000000B	00004400H
Exit reason	00000001B	00004402H
VM-exit interruption information	00000010B	00004404H
VM-exit interruption error code	00000011B	00004406H
IDT-vectoring information field	00000100B	00004408H
IDT-vectoring error code	00000101B	0000440AH
VM-exit instruction length	00000110B	0000440CH
VM-exit instruction information	00000111B	0000440EH

B.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-10 enumerates the 32-bit guest-state fields.

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES limit	00000000B	00004800H
Guest CS limit	00000001B	00004802H
Guest SS limit	00000010B	00004804H
Guest DS limit	00000011B	00004806H
Guest FS limit	00000100B	00004808H
Guest GS limit	00000101B	0000480AH
Guest LDTR limit	00000110B	0000480CH
Guest TR limit	00000111B	0000480EH
Guest GDTR limit	00001000B	00004810H
Guest IDTR limit	00001001B	00004812H
Guest ES access rights	00001010B	00004814H
Guest CS access rights	00001011B	00004816H
Guest SS access rights	00001100B	00004818H
Guest DS access rights	00001101B	0000481AH
Guest FS access rights	00001110B	0000481CH
Guest GS access rights	00001111B	0000481EH
Guest LDTR access rights	00010000B	00004820H

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest TR access rights	000010001B	00004822H
Guest interruptibility state	000010010B	00004824H
Guest activity state	000010011B	00004826H
Guest SMBASE	000010100B	00004828H
Guest IA32_SYSENTER_CS	000010101B	0000482AH
VMX-preemption timer value ¹	000010111B	0000482EH

NOTES:

1. This field exists only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control.

The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

B.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table B-11.

Table B-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host IA32_SYSENTER_CS	000000000B	00004C00H

B.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 25.11.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-12 enumerates the natural-width control fields.

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR0 guest/host mask	000000000B	00006000H
CR4 guest/host mask	000000001B	00006002H
CR0 read shadow	000000010B	00006004H
CR4 read shadow	000000011B	00006006H
CR3-target value 0	000000100B	00006008H
CR3-target value 1	000000101B	0000600AH
CR3-target value 2	000000110B	0000600CH
CR3-target value 3 ¹	000000111B	0000600EH

NOTES:

1. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

B.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-13 enumerates the natural-width read-only data fields.

Table B-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
Exit qualification	00000000B	00006400H
I/O RCX	00000001B	00006402H
I/O RSI	00000010B	00006404H
I/O RDI	00000011B	00006406H
I/O RIP	00000100B	00006408H
Guest-linear address	00000101B	0000640AH

B.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-14 enumerates the natural-width guest-state fields.

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CR0	00000000B	00006800H
Guest CR3	00000001B	00006802H
Guest CR4	00000010B	00006804H
Guest ES base	00000011B	00006806H
Guest CS base	00000100B	00006808H
Guest SS base	00000101B	0000680AH
Guest DS base	00000110B	0000680CH
Guest FS base	00000111B	0000680EH
Guest GS base	00001000B	00006810H
Guest LDTR base	00001001B	00006812H
Guest TR base	00001010B	00006814H
Guest GDTR base	00001011B	00006816H
Guest IDTR base	00001100B	00006818H
Guest DR7	00001101B	0000681AH
Guest RSP	00001110B	0000681CH
Guest RIP	00001111B	0000681EH
Guest RFLAGS	00010000B	00006820H
Guest pending debug exceptions	00010001B	00006822H

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest IA32_SYSENTER_ESP	000010010B	00006824H
Guest IA32_SYSENTER_EIP	000010011B	00006826H
Guest IA32_S_CET ¹	000010100B	00006828H
Guest SSP ¹	000010101B	0000682AH
Guest IA32_INTERRUPT_SSP_TABLE_ADDR ¹	000010110B	0000682CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

B.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-15 enumerates the natural-width host-state fields.

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host CR0	00000000B	00006C00H
Host CR3	00000001B	00006C02H
Host CR4	00000010B	00006C04H
Host FS base	00000011B	00006C06H
Host GS base	00000100B	00006C08H
Host TR base	00000101B	00006C0AH
Host GDTR base	00000110B	00006C0CH
Host IDTR base	00000111B	00006C0EH
Host IA32_SYSENTER_ESP	00001000B	00006C10H
Host IA32_SYSENTER_EIP	00001001B	00006C12H
Host RSP	00001010B	00006C14H
Host RIP	00001011B	00006C16H
Host IA32_S_CET ¹	00001100B	00006C18H
Host SSP ¹	00001101B	00006C1AH
Host IA32_INTERRUPT_SSP_TABLE_ADDR ¹	00001110B	00006C1CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

22. Updates to Appendix C, Volume 3D

Change bars and green text show changes to Appendix C of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Added basic exit reasons 74 and 75 to Table C-1, "Basic Exit Reasons."

APPENDIX C VMX BASIC EXIT REASONS

Every VM exit writes a 32-bit exit reason to the VMCS (see Section 25.9.1). Certain VM-entry failures also do this (see Section 27.8). The low 16 bits of the exit-reason field form the basic exit reason which provides basic information about the cause of the VM exit or VM-entry failure.

Table C-1 lists values for basic exit reasons and explains their meaning. Entries apply to VM exits, unless otherwise noted.

Table C-1. Basic Exit Reasons

Basic Exit Reason	Description
0	Exception or non-maskable interrupt (NMI). Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. This case includes executions of BOUND that cause #BR, executions of INT1 (they cause #DB), executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UDO, UD1, and UD2 (they cause #UD). 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1.
1	External interrupt. An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	Triple fault. The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	INIT signal. An INIT signal arrived
4	Start-up IPI (SIPI). A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	I/O system-management interrupt (SMI). An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 32.15.2).
6	Other SMI. An SMI arrived and caused an SMM VM exit (see Section 32.15.2) but not immediately after retirement of an I/O instruction.
7	Interrupt window. At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	NMI window. At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	Task switch. Guest software attempted a task switch.
10	CPUID. Guest software attempted to execute CPUID.
11	GETSEC. Guest software attempted to execute GETSEC.
12	HLT. Guest software attempted to execute HLT and the "HLT exiting" VM-execution control was 1.
13	INVD. Guest software attempted to execute INVD.
14	INVLPG. Guest software attempted to execute INVLPG and the "INVLPG exiting" VM-execution control was 1.
15	RDPMC. Guest software attempted to execute RDPMC and the "RDPMC exiting" VM-execution control was 1.
16	RDTS. Guest software attempted to execute RDTS and the "RDTS exiting" VM-execution control was 1.
17	RSM. Guest software attempted to execute RSM in SMM.
18	VMCALL. VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 32.15.2).
19	VMCLEAR. Guest software attempted to execute VMCLEAR.
20	VMLAUNCH. Guest software attempted to execute VMLAUNCH.
21	VMPTRLD. Guest software attempted to execute VMPTRLD.
22	VMPTRST. Guest software attempted to execute VMPTRST.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
23	VMREAD. Guest software attempted to execute VMREAD.
24	VMRESUME. Guest software attempted to execute VMRESUME.
25	VMWRITE. Guest software attempted to execute VMWRITE.
26	VMXOFF. Guest software attempted to execute VMXOFF.
27	VMXON. Guest software attempted to execute VMXON.
28	Control-register accesses. Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 26.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1. Such VM exits instead use basic exit reason 43.
29	MOV DR. Guest software attempted a MOV to or from a debug register and the “MOV-DR exiting” VM-execution control was 1.
30	I/O instruction. Guest software attempted to execute an I/O instruction and either: 1: The “use I/O bitmaps” VM-execution control was 0 and the “unconditional I/O exiting” VM-execution control was 1. 2: The “use I/O bitmaps” VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1.
31	RDMSR. Guest software attempted to execute RDMSR and either: 1: The “use MSR bitmaps” VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H - 00001FFFH nor in the range C0000000H - C0001FFFH. 3: The value of RCX was in the range 00000000H - 00001FFFH and the n^{th} bit in read bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H - C0001FFFH and the n^{th} bit in read bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
32	WRMSR. Guest software attempted to execute WRMSR and either: 1: The “use MSR bitmaps” VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H - 00001FFFH nor in the range C0000000H - C0001FFFH. 3: The value of RCX was in the range 00000000H - 00001FFFH and the n^{th} bit in write bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H - C0001FFFH and the n^{th} bit in write bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
33	VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 27.3.1.
34	VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs. See Section 27.4.
36	MWAIT. Guest software attempted to execute MWAIT and the “MWAIT exiting” VM-execution control was 1.
37	Monitor trap flag. A VM exit occurred due to the 1-setting of the “monitor trap flag” VM-execution control (see Section 26.5.2) or VM entry injected a pending MTF VM exit as part of VM entry (see Section 27.6.2).
39	MONITOR. Guest software attempted to execute MONITOR and the “MONITOR exiting” VM-execution control was 1.
40	PAUSE. Either guest software attempted to execute PAUSE and the “PAUSE exiting” VM-execution control was 1 or the “PAUSE-loop exiting” VM-execution control was 1 and guest software executed a PAUSE loop with execution time exceeding PLE_Window (see Section 26.1.3).
41	VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 27.9).
43	TPR below threshold. The logical processor determined that the value of bits 7:4 of the byte at offset 080H on the virtual-APIC page was below that of the TPR threshold VM-execution control field while the “use TPR shadow” VM-execution control was 1 either as part of TPR virtualization (Section 30.1.2) or VM entry (Section 27.7.7).
44	APIC access. Guest software attempted to access memory at a physical address on the APIC-access page and the “virtualize APIC accesses” VM-execution control was 1 (see Section 30.4).
45	Virtualized EOI. EOI virtualization was performed for a virtual interrupt whose vector indexed a bit set in the EOI-exit bitmap.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
46	Access to GDTR or IDTR. Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the “descriptor-table exiting” VM-execution control was 1.
47	Access to LDTR or TR. Guest software attempted to execute LLDT, LTR, SLDT, or STR and the “descriptor-table exiting” VM-execution control was 1.
48	EPT violation. An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.
49	EPT misconfiguration. An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.
50	INVEPT. Guest software attempted to execute INVEPT.
51	RDTSCP. Guest software attempted to execute RDTSCP and the “enable RDTSCP” and “RDTSC exiting” VM-execution controls were both 1.
52	VMX-preemption timer expired. The preemption timer counted down to zero.
53	INNVPID. Guest software attempted to execute INNVPID.
54	WBINVD or WBNOINVD. Guest software attempted to execute WBINVD or WBNOINVD and the “WBINVD exiting” VM-execution control was 1.
55	XSETBV. Guest software attempted to execute XSETBV.
56	APIC write. Guest software completed a write to the virtual-APIC page that must be virtualized by VMM software (see Section 30.4.3.3).
57	RDRAND. Guest software attempted to execute RDRAND and the “RDRAND exiting” VM-execution control was 1.
58	INVPCID. Guest software attempted to execute INVPCID and the “enable INVPCID” and “INVLPG exiting” VM-execution controls were both 1.
59	VMFUNC. Guest software invoked a VM function with the VMFUNC instruction and the VM function either was not enabled or generated a function-specific condition causing a VM exit.
60	ENCLS. Guest software attempted to execute ENCLS, “enable ENCLS exiting” VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the ENCLS-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the ENCLS-exiting bitmap is 1.
61	RDSEED. Guest software attempted to execute RDSEED and the “RDSEED exiting” VM-execution control was 1.
62	Page-modification log full. The processor attempted to create a page-modification log entry and the value of the PML index was not in the range 0–511.
63	XSAVES. Guest software attempted to execute XSAVES, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
64	XRSTORS. Guest software attempted to execute XRSTORS, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
65	PCONFIG. Guest software attempted to execute PCONFIG, “enable PCONFIG” VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the PCONFIG-exiting bitmap is 1.
66	SPP-related event. The processor attempted to determine an access’s sub-page write permission and encountered an SPP miss or an SPP misconfiguration. See Section 29.3.4.2.
67	UMWAIT. Guest software attempted to execute UMWAIT and the “enable user wait and pause” and “RDTSC exiting” VM-execution controls were both 1.
68	TPAUSE. Guest software attempted to execute TPAUSE and the “enable user wait and pause” and “RDTSC exiting” VM-execution controls were both 1.
69	LOADIWKEY. Guest software attempted to execute LOADIWKEY and the “LOADIWKEY exiting” VM-execution control was 1.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
70	ENCLV. Guest software attempted to execute ENCLV, the “enable ENCLV exiting” VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the ENCLV-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the ENCLV-exiting bitmap is 1.
72	ENQCMD PASID translation failure. A VM exit occurred during PASID translation because the present bit was clear in a PASID-directory entry, the valid bit was clear in a PASID-table entry, or one of the entries set a reserved bit.
73	ENQCMDS PASID translation failure. A VM exit occurred during PASID translation because the present bit was clear in a PASID-directory entry, the valid bit was clear in a PASID-table entry, or one of the entries set a reserved bit.
74	Bus lock. The processor asserted a bus lock while the “bus-lock detection” VM-execution control was 1. (Such VM exits will also set bit 26 of the exit-reason field.)
75	Instruction timeout. The “instruction timeout” VM-execution control was 1 and certain operations prevented the processor from reaching an instruction boundary within the amount of time specified by the instruction-timeout control.

23. Updates to Chapter 2, Volume 4

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers*.

Changes to this chapter:

- Update to the IA32_ARCH_CAPABILITIES MSR to add bit 9, MCU_CONTROL.
- Update to the IA32_DEBUGCTL MSR to add bit 2, bus-lock detection.
- Update to MSR_MEMORY_CTRL to add bit 29, SPLIT_LOCK_DISABLE, in the tables of products supporting this bit.
- Update to MSR_MEMORY_CTRL to add bit 28, UC_LOCK_DISABLE, in the tables of products supporting this bit.
- Updated the IA32_CORE_CAPABILITIES MSR to update definitions of bits 4 and 5 (UC_LOCK_DISABLE_SUPPORTED and SPLIT_LOCK_DISABLE_SUPPORTED) in the tables of products supporting these bits.
- Added MSR_PREFETCH_CONTROL to Table 2-47, "MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core."

CHAPTER 2

MODEL-SPECIFIC REGISTERS (MSRS)

This chapter lists MSRs across Intel processor families. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions. The scope of an MSR defines the set of processors that access the same MSR with RDMSR and WRMSR. Thread-scope MSRs are unique to every logical processor. Core-scope MSRs are shared by the threads in the same core; similarly for module-scope, die-scope, and package-scope.

When a processor package contains a single die, die-scope and package-scope are synonymous. When a package contains multiple die, they are distinct.

NOTE

For information on hierarchical level types supported, refer to the CPUID Leaf 1FH definition for the actual level type numbers: "V2 Extended Topology Enumeration Leaf" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. Also see Section 9.9.1, "Hierarchical Mapping of Shared Resources," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H leaf function to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). Table 2-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_85H	Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture
06_57H	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series based on Knights Landing microarchitecture
06_8FH	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture
06_BAH, 06_B7H, 06_BFH	13th generation Intel® Core™ processors supporting Raptor Lake performance hybrid architecture
06_97H, 06_9AH	12th generation Intel® Core™ processors supporting Alder Lake performance hybrid architecture
06_8CH, 06_8DH	11th generation Intel® Core™ processors based on Tiger Lake microarchitecture
06_A7H, 06_A8H	11th generation Intel® Core™ processors based on Rocket Lake microarchitecture
06_7DH, 06_7EH	10th generation Intel® Core™ processors based on Ice Lake microarchitecture
06_A5H, 06_A6H	10th generation Intel® Core™ processors based on Comet Lake microarchitecture
06_66H	Intel® Core™ processors based on Cannon Lake microarchitecture
06_8EH, 06_9EH	7th generation Intel® Core™ processors based on Kaby Lake microarchitecture, 8th and 9th generation Intel® Core™ processors based on Coffee Lake microarchitecture, Intel® Xeon® E processors based on Coffee Lake microarchitecture
06_6AH, 06_6CH	3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture
06_55H	Intel® Xeon® Scalable Processor Family based on Skylake microarchitecture, 2nd generation Intel® Xeon® Scalable Processor Family based on Cascade Lake product, and 3rd generation Intel® Xeon® Scalable Processor Family based on Cooper Lake product
06_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family and E3-1200 v5 product family based on Skylake microarchitecture

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture
06_4FH	Intel Xeon processor E5 v4 Family based on Broadwell microarchitecture, Intel Xeon processor E7 v4 Family, Intel Core i7-69xx Processor Extreme Edition
06_47H	5th generation Intel Core processors, Intel Xeon processor E3-1200 v4 product family based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-4600/2600/1600 v3 product families, Intel Xeon processor E7 v3 product families based on Haswell-E microarchitecture, Intel Core i7-59xx Processor Extreme Edition
06_3CH, 06_45H, 06_46H	4th Generation Intel Core processor and Intel Xeon processor E3-1200 v3 product family based on Haswell microarchitecture
06_3EH	Intel Xeon processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture
06_3EH	Intel Xeon processor E5-2600/1600 v2 product families and Intel Xeon processor E5-2400 v2 product family based on Ivy Bridge-E microarchitecture, Intel Core i7-49xx Processor Extreme Edition
06_3AH	3rd Generation Intel Core Processor and Intel Xeon processor E3-1200 v2 product family based on Ivy Bridge microarchitecture
06_2DH	Intel Xeon processor E5 Family based on Sandy Bridge microarchitecture, Intel Core i7-39xx Processor Extreme Edition
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon processor E3-1200 product family; 2nd Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5, and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon processor 3400, 3500, 5500 series
06_1DH	Intel Xeon processor MP 7400 series
06_17H	Intel Xeon processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_86H, 06_96H, 06_9CH	Intel Atom® processors, Intel® Celeron® processors, Intel® Pentium® processors, and Intel® Pentium® Silver processors based on Tremont Microarchitecture
06_7AH	Intel Atom processors based on Goldmont Plus microarchitecture
06_5FH	Intel Atom processors based on Goldmont microarchitecture (Denverton)
06_5CH	Intel Atom processors based on Goldmont microarchitecture
06_4CH	Intel Atom processor X7-Z8000 and X5-Z8000 series based on Airmont microarchitecture
06_5DH	Intel Atom processor X3-C3000 based on Silvermont microarchitecture
06_5AH	Intel Atom processor Z3500 series
06_4AH	Intel Atom processor Z3400 series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 series

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_36H	Intel Atom processor S1000 Series
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	Intel Atom processor family, Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon processor, Intel Xeon processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon processor, Intel Pentium III processor
06_03H, 06_05H	Intel Pentium II Xeon processor, Intel Pentium II processor
06_01H	Intel Pentium Pro processor
05_01H, 05_02H, 05_04H	Intel Pentium processor, Intel Pentium processor with MMX Technology

The Intel® Quark™ SoC X1000 processor can be identified by the signature of DisplayFamily_DisplayModel = 05_09H and SteppingID = 0

2.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table 2-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 2-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a model-specific MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 2-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of “DF_DM” (see Table 2-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-2. “MAXPHYADDR” is reported by CPUID.8000_0008H leaf.

MSR address range between 40000000H - 4000FFFFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

Table 2-2. IA-32 Architectural MSRs

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Section 2.23, “MSRs in Pentium Processors.”	Pentium Processor (05_01H)
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 2.23, “MSRs in Pentium Processors.”	DF_DM = 05_01H
6H	6	IA32_MONITOR_FILTER_SIZE	See Section 9.10.5, “Monitor/Mwait Address Range Determination.”	0F_03H

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment	
Hex	Decimal				
10H	16	IA32_TIME_STAMP_COUNTER (TSC)	See Section 18.17, "Time-Stamp Counter."	05_01H	
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	Platform ID (R/O) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	06_01H	
		49:0	Reserved		
		52:50	Platform Id (R/O) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7		
		63:53	Reserved		
1BH	27	IA32_APIC_BASE (APIC_BASE)	This register holds the APIC base address, permitting the relocation of the APIC memory map. See Section 11.4.4, "Local APIC Status and Location," and Section 11.4.5, "Relocating the Local APIC Registers."	06_01H	
		7:0	Reserved		
		8	BSP flag (R/W)		
		9	Reserved		
		10	Enable x2APIC mode.		06_1AH
		11	APIC Global Enable (R/W)		
		(MAXPHYADDR - 1):12	APIC Base (R/W)		
63: MAXPHYADDR	Reserved				
3AH	58	IA32_FEATURE_CONTROL	Control Features in Intel 64 Processor (R/W)	If any one enumeration condition for defined bit field holds.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written; writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL contents are preserved across RESET when PWRGOOD is not deasserted.	If any one enumeration condition for defined bit field position greater than bit 0 holds.
		1	Enable VMX inside SMX operation (R/WL) This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology. BIOS must set this bit only when the CPUID function 1 returns VMX feature flag and SMX feature flag set (ECX bits 5 and 6 respectively).	If CPUID.01H:ECX[5] = 1 && CPUID.01H:ECX[6] = 1
		2	Enable VMX outside SMX operation (R/WL) This bit enables VMX for a system executive that does not require SMX. BIOS must set this bit only when the CPUID function 1 returns the VMX feature flag set (ECX bit 5).	If CPUID.01H:ECX[5] = 1
		7:3	Reserved	
		14:8	SENTER Local Function Enables (R/WL) When set, each bit in the field represents an enable control for a corresponding SENTER function. This field is supported only if CPUID.1:ECX.[bit 6] is set.	If CPUID.01H:ECX[6] = 1
		15	SENTER Global Enable (R/WL) This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.1:ECX.[bit 6] is set.	If CPUID.01H:ECX[6] = 1
		16	Reserved	
		17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime re-configuration of SGX Launch Control via the IA32_SGXLEPUBKEYHASHn MSR.	If CPUID.(EAX=07H, ECX=0H): ECX[30] = 1
		18	SGX Global Enable (R/WL) This bit must be set to enable SGX leaf functions.	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		19	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		20	LMCE On (R/W) When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.	If IA32_MCG_CAP[27] = 1
		63:21	Reserved	
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H): EBX[1] = 1
		63:0	THREAD_ADJUST Local offset value of the IA32_TSC for a logical processor. Reset value is zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
48H	72	IA32_SPEC_CTRL	Speculation Control (R/W) The MSR bits are defined as logical processor scope. On some core implementations, the bits may impact sibling logical processors on the same core. This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.	If any one of the enumeration conditions for defined bit field positions holds.
		0	Indirect Branch Restricted Speculation (IBRS). Restricts speculation of indirect branch.	If CPUID.(EAX=07H, ECX=0):EDX[26]=1
		1	Single Thread Indirect Branch Predictors (STIBP). Prevents indirect branch predictions on all logical processors on the core from being controlled by any sibling logical processor in the same core.	If CPUID.(EAX=07H, ECX=0):EDX[27]=1
		2	Speculative Store Bypass Disable (SSBD) delays speculative execution of a load until the addresses for all older stores are known.	If CPUID.(EAX=07H, ECX=0):EDX[31]=1
		3	IPRED_DIS_U If 1, enables IPRED_DIS control for CPL3.	If CPUID.(EAX=07H, ECX=2):EDX[1]=1
		4	IPRED_DIS_S If 1, enables IPRED_DIS control for CPL0/1/2.	If CPUID.(EAX=07H, ECX=2):EDX[1]=1
		5	RRSBA_DIS_U If 1, disables RRSBA behavior for CPL3.	If CPUID.(EAX=07H, ECX=2):EDX[2]=1
		6	RRSBA_DIS_S If 1, disables RRSBA behavior for CPL0/1/2.	If CPUID.(EAX=07H, ECX=2):EDX[2]=1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		7	PSFD If 1, disables Fast Store Forwarding Predictor. Note that setting bit 2 (SSBD) also disables this.	If CPUID.(EAX=07H, ECX=2):EDX[0]=1
		8	DDPD_U If 1, disables the Data Dependent Prefetcher that examines data values in memory while CPL = 3. Note that setting bit 2 (SSBD) also disables this.	If CPUID.(EAX=07H, ECX=2):EDX[3]=1
		9	Reserved	
		10	BHI_DIS_S When '1, enables BHI_DIS_S behavior.	If CPUID.(EAX=07H, ECX=2):EDX[4]=1
		63:11	Reserved	
49H	73	IA32_PRED_CMD	Prediction Command (WO) Gives software a way to issue commands that affect the state of predictors.	If any one of the enumeration conditions for defined bit field positions holds.
		0	Indirect Branch Prediction Barrier (IBPB)	If CPUID.(EAX=07H, ECX=0):EDX[26]=1
		63:1	Reserved	
4EH	78	IA32_PPIN_CTL	Protected Processor Inventory Number Enable Control (R/W)	If CPUID.(EAX=07H, ECX=01H):EBX[0]=1 ¹
		0	LockOut (R/W) If 0, indicates that further writes to IA32_PPIN_CTL is allowed. If 1, indicates that further writes to IA32_PPIN_CTL is disallowed. Writing 1 to this bit is only permitted if the Enable_PPIN bit is clear. The Privileged System Software Inventory Agent should read IA32_PPIN_CTL[bit 1] to determine if IA32_PPIN is accessible. The Privileged System Software Inventory Agent is not expected to write to this MSR.	
		1	Enable_PPIN (R/W) If 1, indicates that IA32_PPIN is accessible using RDMSR. If 0, indicates that IA32_PPIN is inaccessible using RDMSR. Any attempt to read IA32_PPIN will cause #GP.	
		63:2	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
4FH	79	IA32_PPIN	Protected Processor Inventory Number (R/O)	If CPUID.(EAX=07H, ECX=01H):EBX[0]=1 ¹
		63:0	Protected Processor Inventory Number (R/O) A unique value within a given CPUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to IA32_PPIN is enabled. Access to IA32_PPIN is permitted only if IA32_PPIN_CTL[bits 1:0] = '10b'.	
79H	121	IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)	BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 10.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
8BH	139	IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)	BIOS Update Signature (R/W) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
		31:0	Reserved	
		63:32	It is recommended that this field be pre-loaded with zero prior to executing CPUID. If the field remains zero following the execution of CPUID, this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
8CH	140	IA32_SGXLEPUBKEYHASH0	IA32_SGXLEPUBKEYHASH[63:0] (R/W) Bits 63:0 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && CPUID.(EAX=07H, ECX=0H):ECX[30]=1. Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1.
8DH	141	IA32_SGXLEPUBKEYHASH1	IA32_SGXLEPUBKEYHASH[127:64] (R/W) Bits 127:64 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	
8EH	142	IA32_SGXLEPUBKEYHASH2	IA32_SGXLEPUBKEYHASH[191:128] (R/W) Bits 191:128 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	
8FH	143	IA32_SGXLEPUBKEYHASH3	IA32_SGXLEPUBKEYHASH[255:192] (R/W) Bits 255:192 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	
9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	
		0	Valid (R/W)	If CPUID.01H: ECX[5]=1 CPUID.01H: ECX[6] = 1
		1	Reserved	
		2	Controls SMI unblocking by VMXOFF (see Section 32.14.4).	If IA32_VMX_MISC[28]
		11:3	Reserved	
		31:12	MSEG Base (R/W)	
		63:32	Reserved	
9EH	158	IA32_SMBASE	Base address of the logical processor's SMRAM image (R/O, SMM only).	If IA32_VMX_MISC[15]
BCH	188	IA32_MISC_PACKAGE_CTL	Power Filtering Control (R/W) This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.	If IA32_ARCH_CAPABILITIES [10] = 1
		0	ENERGY_FILTERING_ENABLE (R/W) If set, RAPL MSRs report filtered processor power consumption data. This bit can be changed from 0 to 1, but cannot be changed from 1 to 0. After setting, all attempts to clear it are ignored until the next processor reset.	If IA32_ARCH_CAPABILITIES [11] = 1
		63:1	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
BDH	189	IA32_XAPIC_DISABLE_STATUS	xAPIC Disable Status (R/O)	If CPUID.(EAX=07H, ECX=0);EDX[29]=1 and IA32_ARCH_CAPABILITIES [21] = 1
		0	LEGACY_XAPIC_DISABLED When set, indicates that the local APIC is in x2APIC mode (IA32_APIC_BASE.EXTD = 1) and that attempts to clear IA32_APIC_BASE.EXTD will fail (e.g., WRMSR will #GP).	
		63:1	Reserved	
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/w)	If CPUID.0AH: EAX[15:8] > 0
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/w)	If CPUID.0AH: EAX[15:8] > 1
C3H	195	IA32_PMC2	General Performance Counter 2 (R/w)	If CPUID.0AH: EAX[15:8] > 2
C4H	196	IA32_PMC3	General Performance Counter 3 (R/w)	If CPUID.0AH: EAX[15:8] > 3
C5H	197	IA32_PMC4	General Performance Counter 4 (R/w)	If CPUID.0AH: EAX[15:8] > 4
C6H	198	IA32_PMC5	General Performance Counter 5 (R/w)	If CPUID.0AH: EAX[15:8] > 5
C7H	199	IA32_PMC6	General Performance Counter 6 (R/w)	If CPUID.0AH: EAX[15:8] > 6
C8H	200	IA32_PMC7	General Performance Counter 7 (R/w)	If CPUID.0AH: EAX[15:8] > 7
		IA32_CORE_CAPABILITIES	IA32 Core Capabilities Register	If CPUID.(EAX=07H, ECX=0);EDX[30] = 1
		63:0	Reserved.	No architecturally defined bits.
E1H	225	IA32_UMWAIT_CONTROL	UMWAIT Control (R/w)	
		0	C0.2 is not allowed by the OS. Value of "1" means all C0.2 requests revert to C0.1.	
		1	Reserved	
		31:2	Determines the maximum time in TSC-quanta that the processor can reside in either C0.1 or C0.2. A zero value indicates no maximum time. The maximum time value is a 32-bit value where the upper 30 bits come from this field and the lower two bits are zero.	
E7H	231	IA32_MPERF	TSC Frequency Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:0	CO_MCNT: CO TSC Frequency Clock Count Increments at fixed interval (relative to TSC freq.) when the logical processor is in CO. Cleared upon overflow / wrap-around of IA32_APERF.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	CO_ACNT: CO Actual Frequency Clock Count Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in CO. Cleared upon overflow / wrap-around of IA32_MPERF.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (R/O) See Section 12.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	06_01H
		7:0	VCNT: The number of variable memory type ranges in the processor.	
		8	Fixed range MTRRs are supported when set.	
		9	Reserved	
		10	WC Supported when set.	
		11	SMRR Supported when set.	
		12	PRMRR supported when set.	
		63:13	Reserved	
10AH	266	IA32_ARCH_CAPABILITIES	Enumeration of Architectural Features (R/O)	If CPUID.(EAX=07H, ECX=0):EDX[29]=1
		0	RDCL_NO: The processor is not susceptible to Rogue Data Cache Load (RDCL).	
		1	IBRS_ALL: The processor supports enhanced IBRS.	
		2	RSBA: The processor supports RSB Alternate. Alternative branch predictors may be used by RET instructions when the RSB is empty. SW using retpoline may be affected by this behavior.	
		3	SKIP_L1DFL_VMENTRY: A value of 1 indicates the hypervisor need not flush the L1D on VM entry.	
		4	SSB_NO: Processor is not susceptible to Speculative Store Bypass.	
		5	MDS_NO: Processor is not susceptible to Microarchitectural Data Sampling (MDS).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6	IF_PSCCHANGE_MC_NO: The processor is not susceptible to a machine check error due to modifying the size of a code page without TLB invalidation.	
		7	TSX_CTRL: If 1, indicates presence of IA32_TSX_CTRL MSR.	
		8	TAA_NO: If 1, processor is not affected by TAA.	
		9	MCU_CONTROL: If 1, the processor supports the IA32_MCU_CONTROL MSR.	
		10	MISC_PACKAGE_CTLs: The processor supports IA32_MISC_PACKAGE_CTLs MSR.	
		11	ENERGY_FILTERING_CTL: The processor supports setting and reading the IA32_MISC_PACKAGE_CTLs[0] (ENERGY_FILTERING_ENABLE) bit.	
		12	DOITM: If 1, the processor supports Data Operand Independent Timing Mode.	
		13	SBDR_SSDP_NO: The processor is not affected by either the Shared Buffers Data Read (SBDR) vulnerability or the Sideband Stale Data Propagator (SSDP).	
		14	FBSDP_NO: The processor is not affected by the Fill Buffer Stale Data Propagator (FBSDP).	
		15	PSDP_NO: The processor is not affected by vulnerabilities involving the Primary Stale Data Propagator (PSDP).	
		16	Reserved	
		17	FB_CLEAR: If 1, the processor supports overwrite of fill buffer values as part of MD_CLEAR operations with the VERW instruction.	
		18	FB_CLEAR_CTRL: If 1, the processor supports the IA32_MCU_OPT_CTRL MSR and allows software to set bit 3 of that MSR (FB_CLEAR_DIS).	
		19	RRSBA: A value of 1 indicates the processor may have the RRSBA alternate prediction behavior, if not disabled by RRSBA_DIS_U or RRSBA_DIS_S.	
		20	BHI_NO: A value of 1 indicates BHI_NO branch prediction behavior, regardless of the value of IA32_SPEC_CTRL[BHI_DIS_S] MSR bit.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		21	XAPIC_DISABLE_STATUS: Enumerates that the IA32_XAPIC_DISABLE_STATUS MSR exists, and that bit 0 specifies whether the legacy xAPIC is disabled and APIC state is locked to x2APIC.	
		22	Reserved	
		23	OVERCLOCKING_STATUS: If set, the IA32_OVERCLOCKING_STATUS MSR exists.	
		24	PBRBSB_NO: If 1, the processor is not affected by issues related to Post-Barrier Return Stack Buffer Predictions.	
		63:25	Reserved	
10BH	267	IA32_FLUSH_CMD	Flush Command (wO) Gives software a way to invalidate structures with finer granularity than other architectural methods.	If any one of the enumeration conditions for defined bit field positions holds.
		0	L1D_FLUSH: Writeback and invalidate the L1 data cache.	If CPUID.(EAX=07H, ECX=0):EDX[28]=1
		63:1	Reserved	
10FH	271	IA32_TSX_FORCE_ABORT	TSX Force Abort	If CPUID.(EAX=07H, ECX=0):EDX[13]=1
		0	RTM_FORCE_ABORT If 1, all RTM transactions abort with EAX code 0.	R/W, Default: 0 If CPUID.(EAX=07H, ECX=0):EDX[11]=1, bit 0 is always 1 and writes to change it are ignored. If SDV_ENABLE_RTM is 1, bit 0 is always 0 and writes to change it are ignored.
		1	TSX_CPUID_CLEAR When set, CPUID.(EAX=07H, ECX=0):EBX[11]=0 and CPUID.(EAX=07H, ECX=0):EBX[4]=0.	R/W, Default: 0 Can be set only if CPUID.(EAX=07H, ECX=0): EDX[11]=1 or if SDV_ENABLE_RTM is 1.
		2	SDV_ENABLE_RTM When set, CPUID.(EAX=07H, ECX=0):EDX[11]=0 and the processor may not force abort RTM. This unsupported mode should only be used for software development and not for production usage.	R/W, Default: 0 If 0, can be set only if CPUID.(EAX=07H, ECX=0): EDX[11]=1.
		63:3	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
122H	290	IA32_TSX_CTRL	IA32_TSX_CTRL	Thread scope. Not architecturally serializing. Available when CPUID.ARCH_CAP(EAX=7H, ECX = 0):EDX[29] = 1 and IA32_ARCH_CAPABILITIES. bit 7 = 1.
		0	RTM_DISABLE When set to 1, XBEGIN will always abort with EAX code 0.	
		1	TSX_CPUID_CLEAR When set to 1, CPUID.07H.EBX.RTM [bit 11] and CPUID.07H.EBX.HLE [bit 4] report 0. When set to 0 and the SKU supports TSX, these bits will return 1.	
		63:2	Reserved	
123H	291	IA32_MCU_OPT_CTRL	Microcode Update Option Control (R/W)	If CPUID.(EAX=07H, ECX=0):EDX[9]=1 or IA32_ARCH_CAPABILITIES [18] = 1 or IA32_ARCH_CAPABILITIES. FB_CLEAR_CTRL=1
		0	RNGDS_MITG_DIS (R/W) If 0 (default), SRBDS mitigation is enabled for RDRAND and RDSEED. If 1, SRBDS mitigation is disabled for RDRAND and RDSEED executed outside of Intel SGX enclaves.	If CPUID.(EAX=07H, ECX=0):EDX[9]=1
		1	RTM_ALLOW If 0, XBEGIN will always abort with EAX code 0. If 1, XBEGIN behavior depends on the value of IA32_TSX_CTRL[RTM_DISABLE].	Read/Write Setting RTM_LOCKED prevents writes to this bit.
		2	RTM_LOCKED When 1, RTM_ALLOW is locked at zero, writes to RTM_ALLOW will be ignored.	Read-Only status bit
		3	FB_CLEAR_DIS If 1, prevents the VERW instruction from performing an FB_CLEAR action.	If IA32_ARCH_CAPABILITIES. FB_CLEAR_CTRL=1
		63:4	Reserved	
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR (R/W)	06_01H
		15:0	CS Selector.	
		31:16	Not used.	Can be read and written.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:32	Not used.	Writes ignored; reads return zero.
175H	373	IA32_SYSENTER_ESP	SYSENTER_ESP_MSR (R/W)	06_01H
176H	374	IA32_SYSENTER_EIP	SYSENTER_EIP_MSR (R/W)	06_01H
179H	377	IA32_MCG_CAP (MCG_CAP)	Global Machine Check Capability (R/O)	06_01H
		7:0	Count: Number of reporting banks.	
		8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set.	
		9	MCG_EXT_P: Extended machine check state registers are present if this bit is set.	
		10	MCP_CMCI_P: Support for corrected MC error event is present.	06_01H
		11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
		15:12	Reserved	
		23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	
		24	MCG_SER_P: The processor supports software error recovery if this bit is set.	
		25	Reserved	
		26	MCG_ELOG_P: Indicates that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format "Generic Error Data Entry" that augments the data included in machine check bank registers.	06_3EH
		27	MCG_LMCE_P: Indicates that the processor supports extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (R/W0)	06_01H
		0	RIPV. Restart IP valid.	06_01H
		1	EIPV. Error IP valid.	06_01H
		2	MCIP. Machine check in progress.	06_01H
		3	LMCE_S	If IA32_MCG_CAP.LMCE_P[27] = 1
		63:4	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	If IA32_MCG_CAP.CTL_P[8] = 1
180H-185H	384-389	Reserved		06_0EH ²
186H	390	IA32_PERFEVTSELO (PERFEVTSELO)	Performance Event Select Register 0 (R/W)	If CPUID.OAH: EAX[15:8] > 0
		7:0	Event Select: Selects a performance event logic unit.	
		15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	
		16	USR: Counts while in privilege level is not ring 0.	
		17	OS: Counts while in privilege level is ring 0.	
		18	Edge: Enables edge detection if set.	
		19	PC: Enables pin control.	
		20	INT: Enables interrupt on counter overflow.	
		21	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
		22	EN: Enables the corresponding performance counter to commence counting when this bit is set.	
		23	INV: Invert the CMASK.	
	31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.		
	63:32	Reserved		
187H	391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	Performance Event Select Register 1 (R/W)	If CPUID.OAH: EAX[15:8] > 1
188H	392	IA32_PERFEVTSEL2	Performance Event Select Register 2 (R/W)	If CPUID.OAH: EAX[15:8] > 2
189H	393	IA32_PERFEVTSEL3	Performance Event Select Register 3 (R/W)	If CPUID.OAH: EAX[15:8] > 3
18AH	394	IA32_PERFEVTSEL4	Performance Event Select Register 4 (R/W)	If CPUID.OAH: EAX[15:8] > 4
18BH	395	IA32_PERFEVTSEL5	Performance Event Select Register 5 (R/W)	If CPUID.OAH: EAX[15:8] > 5

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
18CH	396	IA32_PERFEVTSEL6	Performance Event Select Register 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
18DH	397	IA32_PERFEVTSEL7	Performance Event Select Register 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
18AH-194H	394-404	Reserved		06_0EH ³
195H	405	IA32_OVERCLOCKING_STATUS	Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR.	
		0	Overclocking Utilized Indicates if specific forms of overclocking have been enabled on this boot or reset cycle: 0 indicates no, 1 indicates yes.	
		1	Undervolt Protection Indicates if the "Dynamic OC Undervolt Protection" security feature is active: 0 indicates disabled, 1 indicates enabled.	
		2	Overclocking Secure Status Indicates that overclocking capabilities have been unlocked by BIOS, with or without overclocking: 0 indicates Not Secured, 1 indicates Secure.	
		63:4	Reserved	
196H-197H	406-407	Reserved		06_0EH ³
198H	408	IA32_PERF_STATUS	Current Performance Status (R/O) See Section 15.1.1, "Software Interface For Initiating Performance State Transitions."	0F_03H
		15:0	Current Performance State Value.	
		63:16	Reserved	
199H	409	IA32_PERF_CTL	Performance Control MSR (R/W) Software makes a request for a new Performance state (P-State) by writing this MSR. See Section 15.1.1, "Software Interface For Initiating Performance State Transitions."	0F_03H
		15:0	Target performance State Value.	
		31:16	Reserved	
		32	Intel® Dynamic Acceleration Technology Engage (R/W) When set to 1: Disengages Intel Dynamic Acceleration Technology.	06_0FH (Mobile only)
		63:33	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 15.8.3, "Software Controlled Clock Modulation."	If CPUID.01H:EDX[22] = 1
		0	Extended On-Demand Clock Modulation Duty Cycle.	If CPUID.06H:EAX[5] = 1
		3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation.	If CPUID.01H:EDX[22] = 1
		4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	If CPUID.01H:EDX[22] = 1
		63:5	Reserved	
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 15.8.2, "Thermal Monitor."	If CPUID.01H:EDX[22] = 1
		0	High-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
		1	Low-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
		2	PROCHOT# Interrupt Enable	If CPUID.01H:EDX[22] = 1
		3	FORCEPR# Interrupt Enable	If CPUID.01H:EDX[22] = 1
		4	Critical Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
		7:5	Reserved	
		14:8	Threshold #1 Value	If CPUID.01H:EDX[22] = 1
		15	Threshold #1 Interrupt Enable	If CPUID.01H:EDX[22] = 1
		22:16	Threshold #2 Value	If CPUID.01H:EDX[22] = 1
		23	Threshold #2 Interrupt Enable	If CPUID.01H:EDX[22] = 1
		24	Power Limit Notification Enable	If CPUID.06H:EAX[4] = 1
		25	Hardware Feedback Notification Enable	If CPUID.06H:EAX[24] = 1
63:26	Reserved			
19CH	412	IA32_THERM_STATUS	Thermal Status Information (R/O) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 15.8.2, "Thermal Monitor."	If CPUID.01H:EDX[22] = 1
		0	Thermal Status (R/O)	If CPUID.01H:EDX[22] = 1
		1	Thermal Status Log (R/W)	If CPUID.01H:EDX[22] = 1
		2	PROCHOT # or FORCEPR# event (R/O)	If CPUID.01H:EDX[22] = 1
		3	PROCHOT # or FORCEPR# log (R/WCO)	If CPUID.01H:EDX[22] = 1
		4	Critical Temperature Status (R/O)	If CPUID.01H:EDX[22] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		5	Critical Temperature Status log (R/WCO)	If CPUID.01H:EDX[22] = 1
		6	Thermal Threshold #1 Status (R/O)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WCO)	If CPUID.01H:ECX[8] = 1
		8	Thermal Threshold #2 Status (R/O)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #2 log (R/WCO)	If CPUID.01H:ECX[8] = 1
		10	Power Limitation Status (R/O)	If CPUID.06H:EAX[4] = 1
		11	Power Limitation log (R/WCO)	If CPUID.06H:EAX[4] = 1
		12	Current Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
		13	Current Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
		14	Cross Domain Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
		15	Cross Domain Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
		22:16	Digital Readout (R/O)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved	
		30:27	Resolution in Degrees Celsius (R/O)	If CPUID.06H:EAX[0] = 1
		31	Reading Valid (R/O)	If CPUID.06H:EAX[0] = 1
63:32	Reserved			
1A0H	416	IA32_MISC_ENABLE	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.	
		0	Fast-Strings Enable When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default). When clear, fast-strings are disabled.	OF_OH
		2:1	Reserved	
		3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled. Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2, and adaptive thermal throttling will still be activated. The default value of this field varies with product. See respective tables where default value is listed.	OF_OH
		6:4	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.	0F_0H
		10:8	Reserved	
		11	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS). 0 = BTS is supported.	0F_0H
		12	Processor Event Based Sampling (PEBS) Unavailable (R/O) 1 = PEBS is not supported. 0 = PEBS is supported.	06_0FH
		15:13	Reserved	
		16	Enhanced Intel SpeedStep Technology Enable (R/W) 0= Enhanced Intel SpeedStep Technology disabled. 1 = Enhanced Intel SpeedStep Technology enabled.	If CPUID.01H: ECX[7] =1
		17	Reserved	
		18	ENABLE MONITOR FSM (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
		21:19	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		22	<p>Limit CPUID Maxval (R/W)</p> <p>When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2.</p> <p>BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 2.</p> <p>Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 2, this bit is supported.</p> <p>Otherwise, this bit is not supported. Setting this bit when the maximum value is not greater than 2 may generate a #GP exception.</p> <p>Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 2.</p>	0F_03H
		23	<p>xTPR Message Disable (R/W)</p> <p>When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.</p>	If CPUID.01H:ECX[14] = 1
		33:24	Reserved	
		34	<p>XD Bit Disable (R/W)</p> <p>When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0).</p> <p>When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages.</p> <p>BIOS must not alter the contents of this bit location, if XD bit is not supported. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.</p>	If CPUID.80000001H:EDX[20] = 1
		63:35	Reserved	
1B0H	432	IA32_ENERGY_PERF_BIAS	Performance Energy Bias Hint (R/W)	If CPUID.6H:ECX[3] = 1
		3:0	<p>Power Policy Preference:</p> <p>0 indicates preference to highest performance.</p> <p>15 indicates preference to maximize energy saving.</p>	
		63:4	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package Thermal Status Information (R/O) Contains status information about the package's thermal sensor. See Section 15.9, "Package Level Thermal Management."	If CPUID.06H: EAX[6] = 1
		0	Pkg Thermal Status (R/O)	
		1	Pkg Thermal Status Log (R/W)	
		2	Pkg PROCHOT # event (R/O)	
		3	Pkg PROCHOT # log (R/WCO)	
		4	Pkg Critical Temperature Status (R/O)	
		5	Pkg Critical Temperature Status Log (R/WCO)	
		6	Pkg Thermal Threshold #1 Status (R/O)	
		7	Pkg Thermal Threshold #1 Log (R/WCO)	
		8	Pkg Thermal Threshold #2 Status (R/O)	
		9	Pkg Thermal Threshold #1 Log (R/WCO)	
		10	Pkg Power Limitation Status (R/O)	
		11	Pkg Power Limitation Log (R/WCO)	
		15:12	Reserved	
		22:16	Pkg Digital Readout (R/O)	
		25:23	Reserved	
26	Hardware Feedback Interface Structure Change Status	If CPUID.06H:EAX.[19] = 1		
63:27	Reserved			
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Pkg Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the package's thermal sensor. See Section 15.9, "Package Level Thermal Management."	If CPUID.06H: EAX[6] = 1
		0	Pkg High-Temperature Interrupt Enable	
		1	Pkg Low-Temperature Interrupt Enable	
		2	Pkg PROCHOT# Interrupt Enable	
		3	Reserved	
		4	Pkg Overheat Interrupt Enable	
		7:5	Reserved	
		14:8	Pkg Threshold #1 Value	
		15	Pkg Threshold #1 Interrupt Enable	
22:16	Pkg Threshold #2 Value			

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		23	Pkg Threshold #2 Interrupt Enable	
		24	Pkg Power Limit Notification Enable	
		25	Hardware Feedback Interrupt Enable	If CPUID.06H:EAX.[19] = 1
		63:26	Reserved	
1C4H	452	IA32_XFD	Extended Feature Disable Control (R/W) Controls which XSAVE-enabled features are temporarily disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.	If CPUID.(EAX=0DH,ECX=1): EAX[4] = 1
1C5H	453	IA32_XFD_ERR	Extended Feature Disable Error Code (R/W) Reports which XSAVE-enabled features caused a fault due to being disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.	If CPUID.(EAX=0DH,ECX=1): EAX[4] = 1
1D9H	473	IA32_DEBUGCTL (MSR_DEBUGCTLA, MSR_DEBUGCTLB)	Trace/Profile Resource Control (R/W)	06_0EH
		0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	06_01H
		1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	06_01H
		2	BLD: Enable OS bus-lock detection. See Section 18.3.1.6 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.	If (CPUID.(EAX=07H, ECX=0):ECX[24] = 1)
		5:3	Reserved	
		6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH
		7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH
		8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
		9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
		10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H: ECX[15] = 1 && CPUID.0AH: EAX[7:0] > 1
		12	FREEZE_PERFMON_ON_PMI: When set, each ENABLE bit of the global counter control MSR are frozen (address 38FH) on a PMI request.	If CPUID.01H: ECX[15] = 1 && CPUID.0AH: EAX[7:0] > 1
		13	ENABLE_UNCORE_PMI: When set, enables the logical processor to receive and generate PMI on behalf of the uncore.	06_1AH
		14	FREEZE_WHILE_SMM: When set, freezes perfmon and trace messages while in SMM.	If IA32_PERF_CAPABILITIES[12] = 1
		15	RTM_DEBUG: When set, enables DR7 debug bit on XBEGIN.	If (CPUID.(EAX=07H, ECX=0);EBX[11] = 1)
		63:16	Reserved	
1DDH	477	IA32_LER_FROM_IP	Last Event Record Source IP Register (R/W)	
		63:0	FROM_IP The source IP of the recorded branch or event, in canonical form.	Reset Value: 0
1DEH	478	IA32_LER_TO_IP	Last Event Record Destination IP Register (R/W)	
		63:0	TO_IP The destination IP of the recorded branch or event, in canonical form.	Reset Value: 0
1E0H	480	IA32_LER_INFO	Last Event Record Info Register (R/W)	
		55:0	Undefined, may be zero or non-zero. Writes of non- zero values do not fault, but reads may return a different value.	Reset Value: 0
		59:56	BR_TYPE The branch type recorded by this LBR. Encodings match those of IA32_LBR_x_INFO.	Reset Value: 0
		60	Undefined, may be zero or non-zero. Writes of non- zero values do not fault, but reads may return a different value.	Reset Value: 0
		61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel® TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	Reset Value: 0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel® TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	Reset Value: 0
		63	MISPRED The recorded branch taken/not-taken resolution (for conditional branches) or target (for any indirect branch, including RETs) was mispredicted.	Reset Value: 0
1F2H	498	IA32_SMRR_PHYSBASE	SMRR Base Address (Writeable only in SMM) Base address of SMM memory range.	If IA32_MTRRCAP.SMRR[11] = 1
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved	
		31:12	PhysBase SMRR physical Base Address.	
		63:32	Reserved	
1F3H	499	IA32_SMRR_PHYSMASK	SMRR Range Mask (Writeable only in SMM) Range Mask of SMM memory range.	If IA32_MTRRCAP[SMRR] = 1
		10:0	Reserved	
		11	Valid Enable range mask.	
		31:12	PhysMask SMRR address range mask.	
		63:32	Reserved	
1F8H	504	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	If CPUID.01H: ECX[18] = 1
1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	If CPUID.01H: ECX[18] = 1
1FAH	506	IA32_DCA_0_CAP	DCA type 0 Status and Control register.	If CPUID.01H: ECX[18] = 1
		0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	
		2:1	TRANSACTION	
		6:3	DCA_TYPE	
		10:7	DCA_QUEUE_SIZE	
		12:11	Reserved	
		16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	
		23:17	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		24	SW_BLOCK: SW can request DCA block by setting this bit.	
		25	Reserved	
		26	HW_BLOCK: Set when DCA is blocked by HW (e.g., CR0.CD = 1).	
		31:27	Reserved	
200H	512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	See Section 12.11.2.3, "Variable Range MTRRs."	If IA32_MTRRCAP[7:0] > 0
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	If IA32_MTRRCAP[7:0] > 0
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	If IA32_MTRRCAP[7:0] > 1
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	If IA32_MTRRCAP[7:0] > 1
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	If IA32_MTRRCAP[7:0] > 2
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	If IA32_MTRRCAP[7:0] > 2
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	If IA32_MTRRCAP[7:0] > 3
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	If IA32_MTRRCAP[7:0] > 3
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	If IA32_MTRRCAP[7:0] > 4
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	If IA32_MTRRCAP[7:0] > 4
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	If IA32_MTRRCAP[7:0] > 5
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	If IA32_MTRRCAP[7:0] > 5
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	If IA32_MTRRCAP[7:0] > 6
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	If IA32_MTRRCAP[7:0] > 6
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	If IA32_MTRRCAP[7:0] > 7
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	If IA32_MTRRCAP[7:0] > 7
210H	528	IA32_MTRR_PHYSBASE8	MTRRphysBase8	If IA32_MTRRCAP[7:0] > 8
211H	529	IA32_MTRR_PHYSMASK8	MTRRphysMask8	If IA32_MTRRCAP[7:0] > 8
212H	530	IA32_MTRR_PHYSBASE9	MTRRphysBase9	If IA32_MTRRCAP[7:0] > 9
213H	531	IA32_MTRR_PHYSMASK9	MTRRphysMask9	If IA32_MTRRCAP[7:0] > 9
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	If CPUID.01H: EDX.MTRR[12] = 1
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	If CPUID.01H: EDX.MTRR[12] = 1
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	If CPUID.01H: EDX.MTRR[12] = 1
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	See Section 12.11.2.2, "Fixed Range MTRRs."	If CPUID.01H: EDX.MTRR[12] = 1
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	If CPUID.01H: EDX.MTRR[12] = 1
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	If CPUID.01H: EDX.MTRR[12] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	If CPUID.01H: EDX.MTRR[12] = 1
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	If CPUID.01H: EDX.MTRR[12] = 1
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	If CPUID.01H: EDX.MTRR[12] = 1
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	If CPUID.01H: EDX.MTRR[12] = 1
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	If CPUID.01H: EDX.MTRR[12] = 1
277H	631	IA32_PAT	IA32_PAT (R/W)	If CPUID.01H: EDX.MTRR[16] = 1
		2:0	PA0	
		7:3	Reserved	
		10:8	PA1	
		15:11	Reserved	
		18:16	PA2	
		23:19	Reserved	
		26:24	PA3	
		31:27	Reserved	
		34:32	PA4	
		39:35	Reserved	
		42:40	PA5	
		47:43	Reserved	
		50:48	PA6	
		55:51	Reserved	
58:56	PA7			
63:59	Reserved			
280H	640	IA32_MCO_CTL2	MSR to enable/disable CMCI capability for bank 0. (R/W) See Section 16.3.2.5, "IA32_MCi_CTL2 MSRs."	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 0
		14:0	Corrected error count threshold.	
		29:15	Reserved	
		30	CMCI_EN	
		63:31	Reserved	
281H	641	IA32_MC1_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
282H	642	IA32_MC2_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
283H	643	IA32_MC3_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
284H	644	IA32_MC4_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
285H	645	IA32_MC5_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
286H	646	IA32_MC6_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
287H	647	IA32_MC7_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
288H	648	IA32_MC8_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
289H	649	IA32_MC9_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
28AH	650	IA32_MC10_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
28BH	651	IA32_MC11_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
28CH	652	IA32_MC12_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
28DH	653	IA32_MC13_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
28EH	654	IA32_MC14_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
28FH	655	IA32_MC15_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
290H	656	IA32_MC16_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
291H	657	IA32_MC17_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
292H	658	IA32_MC18_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
293H	659	IA32_MC19_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
294H	660	IA32_MC20_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
295H	661	IA32_MC21_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
296H	662	IA32_MC22_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
297H	663	IA32_MC23_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
298H	664	IA32_MC24_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
299H	665	IA32_MC25_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
29AH	666	IA32_MC26_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
29BH	667	IA32_MC27_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
29CH	668	IA32_MC28_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
29DH	669	IA32_MC29_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
29EH	670	IA32_MC30_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
29FH	671	IA32_MC31_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType (R/W)	If CPUID.01H: EDX.MTRR[12] = 1
		2:0	Default Memory Type	
		9:3	Reserved	
		10	Fixed Range MTRR Enable	
		11	MTRR Enable	
		63:12	Reserved	
309H	777	IA32_FIXED_CTR0	Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any.	If CPUID.0AH: EDX[4:0] > 0
30AH	778	IA32_FIXED_CTR1	Fixed-Function Performance Counter 1 (R/W): Counts CPU_CLK_Unhalted.Core.	If CPUID.0AH: EDX[4:0] > 1
30BH	779	IA32_FIXED_CTR2	Fixed-Function Performance Counter 2 (R/W): Counts CPU_CLK_Unhalted.Ref.	If CPUID.0AH: EDX[4:0] > 2
345H	837	IA32_PERF_CAPABILITIES	Read Only MSR that enumerates the existence of performance monitoring features. (R/O)	If CPUID.01H: ECX[15] = 1
		5:0	LBR format	
		6	PEBS Trap	
		7	PEBSSaveArchRegs	
		11:8	PEBS Record Format	
		12	1: Freeze while SMM is supported.	
		13	1: Full width of counter writable via IA32_A_PMCx.	
		14	PEBS_BASELINE	
		15	1: Performance metrics available.	
		16	1: PEBS output will be written into the Intel PT trace stream.	If CPUID.0x7.0.EBX[25]=1
		63:17	Reserved	
38DH	909	IA32_FIXED_CTR_CTRL	Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.	If CPUID.0AH: EAX[7:0] > 1
		0	ENO_OS: Enable Fixed Counter 0 to count while CPL = 0.	
		1	ENO_Usr: Enable Fixed Counter 0 to count while CPL > 0.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		2	AnyThr0: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15]=0
		3	EN0_PMI: Enable PMI when fixed counter 0 overflows.	
		4	EN1_OS: Enable Fixed Counter 1 to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1 to count while CPL > 0.	
		6	AnyThr1: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15]=0
		7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
		8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	
		9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0.	
		10	AnyThr2: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15]=0
		11	EN2_PMI: Enable PMI when fixed counter 2 overflows.	
		12	EN3_OS: Enable Fixed Counter 3 to count while CPL = 0.	
		13	EN3_Usr: Enable Fixed Counter 3 to count while CPL > 0.	
		14	Reserved	
		15	EN3_PMI: Enable PMI when fixed counter 3 overflows.	
		63:16	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
38EH	910	IA32_PERF_GLOBAL_STATUS	Global Performance Counter Status (R/O)	If CPUID.0AH: EAX[7:0] > 0 (CPUID.(EAX=07H, ECX=0);EBX[25] = 1 && CPUID.(EAX=014H, ECX=0);ECX[0] = 1)
		0	Ovf_PMC0: Overflow status of IA32_PMC0.	If CPUID.0AH: EAX[15:8] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH: EAX[15:8] > 1
		2	Ovf_PMC2: Overflow status of IA32_PMC2.	If CPUID.0AH: EAX[15:8] > 2
		3	Ovf_PMC3: Overflow status of IA32_PMC3.	If CPUID.0AH: EAX[15:8] > 3
		n	Ovf_PMCn: Overflow status of IA32_PMCn.	If CPUID.0AH: EAX[15:8] > n
		31:n+1	Reserved	
		32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0.	If CPUID.0AH: EAX[7:0] > 1
		33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1.	If CPUID.0AH: EAX[7:0] > 1
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH: EAX[7:0] > 1
		47:35	Reserved	
		48	Ovf_PERF_METRICS: If this bit is set, it indicates that PERF_METRIC counter has overflowed and a PMI is triggered; however, an overflow of fixed counter 3 should normally happen first. If this bit is clear no overflow occurred.	
		54:49	Reserved	
		55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer that was completely filled.	If CPUID.(EAX=07H, ECX=0);EBX[25] = 1 && CPUID.(EAX=014H, ECX=0);ECX[0] = 1
		57:56	Reserved	
		58	LBR_Frz. LBRs are frozen due to: <ul style="list-style-type: none"> ▪ IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1. ▪ The LBR stack overflowed. 	If CPUID.0AH: EAX[7:0] > 3
		59	CTR_Frz. Performance counters in the core PMU are frozen due to: <ul style="list-style-type: none"> ▪ IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1. ▪ One or more core PMU counters overflowed. 	If CPUID.0AH: EAX[7:0] > 3

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		60	ASCI: Data in the performance counters in the core PMU may include contributions from the direct or indirect operation Intel SGX to protect an enclave.	If CPUID.(EAX=07H, ECX=0):EBX[2] = 1
		61	Ovf_Uncore: Uncore counter overflow status.	If CPUID.0AH: EAX[7:0] > 2
		62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.0AH: EAX[7:0] > 0
		63	CondChgd: Status bits of this register have changed.	If CPUID.0AH: EAX[7:0] > 0
38FH	911	IA32_PERF_GLOBAL_CTRL	Global Performance Counter Control (R/W) Counter increments while the result of ANDing the respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.	If CPUID.0AH: EAX[7:0] > 0
		0	EN_PMC0	If CPUID.0AH: EAX[15:8] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[15:8] > 1
		2	EN_PMC2	If CPUID.0AH: EAX[15:8] > 2
		n	EN_PMCn	If CPUID.0AH: EAX[15:8] > n
		31:n+1	Reserved	
		32	EN_FIXED_CTR0	If CPUID.0AH: EDX[4:0] > 0
		33	EN_FIXED_CTR1	If CPUID.0AH: EDX[4:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EDX[4:0] > 2
		47:35	Reserved	
		48	EN_PERF_METRICS: If this bit is set and fixed counter 3 is effectively enabled, built-in performance metrics are enabled.	
		63:49	Reserved	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0 && CPUID.0AH: EAX[7:0] <= 3
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH: EAX[15:8] > n

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		31:n	Reserved	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EDX[4:0] > 2
		54:35	Reserved	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If (CPUID.(EAX=07H, ECX=0);EBX[25] = 1) && IA32_RTIT_CTL.ToPA = 1
		60:56	Reserved	
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf bit.	If CPUID.0AH: EAX[7:0] > 0
		63	Set 1 to clear CondChgd bit.	If CPUID.0AH: EAX[7:0] > 0
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Global Performance Counter Overflow Reset Control (R/W)	If CPUID.0AH: EAX[7:0] > 3 (CPUID.(EAX=07H, ECX=0);EBX[25] = 1 && CPUID.(EAX=014H, ECX=0);ECX[0] = 1)
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EDX[4:0] > 2
		47:35	Reserved	
		48	RESET_OVF_PERF_METRICS: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
		54:49	Reserved	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If CPUID.(EAX=07H, ECX=0);EBX[25] = 1 && CPUID.(EAX=014H, ECX=0);ECX[0] = 1
		57:56	Reserved	
		58	Set 1 to Clear LBR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3
59	Set 1 to Clear CTR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		58	Set 1 to Clear ASCII bit.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf bit.	If CPUID.0AH: EAX[7:0] > 0
		63	Set 1 to clear CondChgd bit.	If CPUID.0AH: EAX[7:0] > 0
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Global Performance Counter Overflow Set Control (R/W)	If CPUID.0AH: EAX[7:0] > 3 (CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):ECX[0] = 1)
		0	Set 1 to cause Ovf_PMC0 = 1.	If CPUID.0AH: EAX[7:0] > 3
		1	Set 1 to cause Ovf_PMC1 = 1.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to cause Ovf_PMC2 = 1.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to cause Ovf_PMCn = 1.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved	
		32	Set 1 to cause Ovf_FIXED_CTR0 = 1.	If CPUID.0AH: EAX[7:0] > 3
		33	Set 1 to cause Ovf_FIXED_CTR1 = 1.	If CPUID.0AH: EAX[7:0] > 3
		34	Set 1 to cause Ovf_FIXED_CTR2 = 1.	If CPUID.0AH: EAX[7:0] > 3
		47:35	Reserved	
		48	SET_OVF_PERF_METRICS: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
		54:49	Reserved	
		55	Set 1 to cause Trace_ToPA_PMI = 1.	If CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):ECX[0] = 1
		57:56	Reserved	
		58	Set 1 to cause LBR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to cause CTR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to cause ASCII = 1.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.0AH: EAX[7:0] > 3
		62	Set 1 to cause OvfBuf = 1.	If CPUID.0AH: EAX[7:0] > 3
		63	Reserved	
392H	914	IA32_PERF_GLOBAL_INUSE	Indicator that core perfmon interface is in use. (R/O)	If CPUID.0AH: EAX[7:0] > 3
		0	IA32_PERFEVTSEL0 in use.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		1	IA32_PERFEVTSEL1 in use.	If CPUID.OAH: EAX[15:8] > 1
		2	IA32_PERFEVTSEL2 in use.	If CPUID.OAH: EAX[15:8] > 2
		n	IA32_PERFEVTSELn in use.	If CPUID.OAH: EAX[15:8] > n
		31:n+1	Reserved	
		32	IA32_FIXED_CTR0 in use.	
		33	IA32_FIXED_CTR1 in use.	
		34	IA32_FIXED_CTR2 in use.	
		62:35	Reserved or model specific.	
		63	PMI in use.	
3F1H	1009	IA32_PEBB_ENABLE	PEBS Control (R/W)	
		0	Enable PEBS on IA32_PMC0.	06_0FH
		3:1	Reserved or model specific.	
		31:4	Reserved	
		35:32	Reserved or model specific.	
		63:36	Reserved	
400H	1024	IA32_MCO_CTL	MCO_CTL	If IA32_MCG_CAP.CNT > 0
401H	1025	IA32_MCO_STATUS	MCO_STATUS	If IA32_MCG_CAP.CNT > 0
402H	1026	IA32_MCO_ADDR ¹	MCO_ADDR	If IA32_MCG_CAP.CNT > 0
403H	1027	IA32_MCO_MISC	MCO_MISC	If IA32_MCG_CAP.CNT > 0
404H	1028	IA32_MC1_CTL	MC1_CTL	If IA32_MCG_CAP.CNT > 1
405H	1029	IA32_MC1_STATUS	MC1_STATUS	If IA32_MCG_CAP.CNT > 1
406H	1030	IA32_MC1_ADDR ²	MC1_ADDR	If IA32_MCG_CAP.CNT > 1
407H	1031	IA32_MC1_MISC	MC1_MISC	If IA32_MCG_CAP.CNT > 1
408H	1032	IA32_MC2_CTL	MC2_CTL	If IA32_MCG_CAP.CNT > 2
409H	1033	IA32_MC2_STATUS	MC2_STATUS	If IA32_MCG_CAP.CNT > 2
40AH	1034	IA32_MC2_ADDR ¹	MC2_ADDR	If IA32_MCG_CAP.CNT > 2
40BH	1035	IA32_MC2_MISC	MC2_MISC	If IA32_MCG_CAP.CNT > 2
40CH	1036	IA32_MC3_CTL	MC3_CTL	If IA32_MCG_CAP.CNT > 3
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	If IA32_MCG_CAP.CNT > 3
40EH	1038	IA32_MC3_ADDR ¹	MC3_ADDR	If IA32_MCG_CAP.CNT > 3
40FH	1039	IA32_MC3_MISC	MC3_MISC	If IA32_MCG_CAP.CNT > 3
410H	1040	IA32_MC4_CTL	MC4_CTL	If IA32_MCG_CAP.CNT > 4
411H	1041	IA32_MC4_STATUS	MC4_STATUS	If IA32_MCG_CAP.CNT > 4
412H	1042	IA32_MC4_ADDR ¹	MC4_ADDR	If IA32_MCG_CAP.CNT > 4
413H	1043	IA32_MC4_MISC	MC4_MISC	If IA32_MCG_CAP.CNT > 4

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
414H	1044	IA32_MC5_CTL	MC5_CTL	If IA32_MCG_CAP.CNT >5
415H	1045	IA32_MC5_STATUS	MC5_STATUS	If IA32_MCG_CAP.CNT >5
416H	1046	IA32_MC5_ADDR ¹	MC5_ADDR	If IA32_MCG_CAP.CNT >5
417H	1047	IA32_MC5_MISC	MC5_MISC	If IA32_MCG_CAP.CNT >5
418H	1048	IA32_MC6_CTL	MC6_CTL	If IA32_MCG_CAP.CNT >6
419H	1049	IA32_MC6_STATUS	MC6_STATUS	If IA32_MCG_CAP.CNT >6
41AH	1050	IA32_MC6_ADDR ¹	MC6_ADDR	If IA32_MCG_CAP.CNT >6
41BH	1051	IA32_MC6_MISC	MC6_MISC	If IA32_MCG_CAP.CNT >6
41CH	1052	IA32_MC7_CTL	MC7_CTL	If IA32_MCG_CAP.CNT >7
41DH	1053	IA32_MC7_STATUS	MC7_STATUS	If IA32_MCG_CAP.CNT >7
41EH	1054	IA32_MC7_ADDR ¹	MC7_ADDR	If IA32_MCG_CAP.CNT >7
41FH	1055	IA32_MC7_MISC	MC7_MISC	If IA32_MCG_CAP.CNT >7
420H	1056	IA32_MC8_CTL	MC8_CTL	If IA32_MCG_CAP.CNT >8
421H	1057	IA32_MC8_STATUS	MC8_STATUS	If IA32_MCG_CAP.CNT >8
422H	1058	IA32_MC8_ADDR ¹	MC8_ADDR	If IA32_MCG_CAP.CNT >8
423H	1059	IA32_MC8_MISC	MC8_MISC	If IA32_MCG_CAP.CNT >8
424H	1060	IA32_MC9_CTL	MC9_CTL	If IA32_MCG_CAP.CNT >9
425H	1061	IA32_MC9_STATUS	MC9_STATUS	If IA32_MCG_CAP.CNT >9
426H	1062	IA32_MC9_ADDR ¹	MC9_ADDR	If IA32_MCG_CAP.CNT >9
427H	1063	IA32_MC9_MISC	MC9_MISC	If IA32_MCG_CAP.CNT >9
428H	1064	IA32_MC10_CTL	MC10_CTL	If IA32_MCG_CAP.CNT >10
429H	1065	IA32_MC10_STATUS	MC10_STATUS	If IA32_MCG_CAP.CNT >10
42AH	1066	IA32_MC10_ADDR ¹	MC10_ADDR	If IA32_MCG_CAP.CNT >10
42BH	1067	IA32_MC10_MISC	MC10_MISC	If IA32_MCG_CAP.CNT >10
42CH	1068	IA32_MC11_CTL	MC11_CTL	If IA32_MCG_CAP.CNT >11
42DH	1069	IA32_MC11_STATUS	MC11_STATUS	If IA32_MCG_CAP.CNT >11
42EH	1070	IA32_MC11_ADDR ¹	MC11_ADDR	If IA32_MCG_CAP.CNT >11
42FH	1071	IA32_MC11_MISC	MC11_MISC	If IA32_MCG_CAP.CNT >11
430H	1072	IA32_MC12_CTL	MC12_CTL	If IA32_MCG_CAP.CNT >12
431H	1073	IA32_MC12_STATUS	MC12_STATUS	If IA32_MCG_CAP.CNT >12
432H	1074	IA32_MC12_ADDR ¹	MC12_ADDR	If IA32_MCG_CAP.CNT >12
433H	1075	IA32_MC12_MISC	MC12_MISC	If IA32_MCG_CAP.CNT >12
434H	1076	IA32_MC13_CTL	MC13_CTL	If IA32_MCG_CAP.CNT >13
435H	1077	IA32_MC13_STATUS	MC13_STATUS	If IA32_MCG_CAP.CNT >13
436H	1078	IA32_MC13_ADDR ¹	MC13_ADDR	If IA32_MCG_CAP.CNT >13
437H	1079	IA32_MC13_MISC	MC13_MISC	If IA32_MCG_CAP.CNT >13
438H	1080	IA32_MC14_CTL	MC14_CTL	If IA32_MCG_CAP.CNT >14

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
439H	1081	IA32_MC14_STATUS	MC14_STATUS	If IA32_MCG_CAP.CNT >14
43AH	1082	IA32_MC14_ADDR ¹	MC14_ADDR	If IA32_MCG_CAP.CNT >14
43BH	1083	IA32_MC14_MISC	MC14_MISC	If IA32_MCG_CAP.CNT >14
43CH	1084	IA32_MC15_CTL	MC15_CTL	If IA32_MCG_CAP.CNT >15
43DH	1085	IA32_MC15_STATUS	MC15_STATUS	If IA32_MCG_CAP.CNT >15
43EH	1086	IA32_MC15_ADDR ¹	MC15_ADDR	If IA32_MCG_CAP.CNT >15
43FH	1087	IA32_MC15_MISC	MC15_MISC	If IA32_MCG_CAP.CNT >15
440H	1088	IA32_MC16_CTL	MC16_CTL	If IA32_MCG_CAP.CNT >16
441H	1089	IA32_MC16_STATUS	MC16_STATUS	If IA32_MCG_CAP.CNT >16
442H	1090	IA32_MC16_ADDR ¹	MC16_ADDR	If IA32_MCG_CAP.CNT >16
443H	1091	IA32_MC16_MISC	MC16_MISC	If IA32_MCG_CAP.CNT >16
444H	1092	IA32_MC17_CTL	MC17_CTL	If IA32_MCG_CAP.CNT >17
445H	1093	IA32_MC17_STATUS	MC17_STATUS	If IA32_MCG_CAP.CNT >17
446H	1094	IA32_MC17_ADDR ¹	MC17_ADDR	If IA32_MCG_CAP.CNT >17
447H	1095	IA32_MC17_MISC	MC17_MISC	If IA32_MCG_CAP.CNT >17
448H	1096	IA32_MC18_CTL	MC18_CTL	If IA32_MCG_CAP.CNT >18
449H	1097	IA32_MC18_STATUS	MC18_STATUS	If IA32_MCG_CAP.CNT >18
44AH	1098	IA32_MC18_ADDR ¹	MC18_ADDR	If IA32_MCG_CAP.CNT >18
44BH	1099	IA32_MC18_MISC	MC18_MISC	If IA32_MCG_CAP.CNT >18
44CH	1100	IA32_MC19_CTL	MC19_CTL	If IA32_MCG_CAP.CNT >19
44DH	1101	IA32_MC19_STATUS	MC19_STATUS	If IA32_MCG_CAP.CNT >19
44EH	1102	IA32_MC19_ADDR ¹	MC19_ADDR	If IA32_MCG_CAP.CNT >19
44FH	1103	IA32_MC19_MISC	MC19_MISC	If IA32_MCG_CAP.CNT >19
450H	1104	IA32_MC20_CTL	MC20_CTL	If IA32_MCG_CAP.CNT >20
451H	1105	IA32_MC20_STATUS	MC20_STATUS	If IA32_MCG_CAP.CNT >20
452H	1106	IA32_MC20_ADDR ¹	MC20_ADDR	If IA32_MCG_CAP.CNT >20
453H	1107	IA32_MC20_MISC	MC20_MISC	If IA32_MCG_CAP.CNT >20
454H	1108	IA32_MC21_CTL	MC21_CTL	If IA32_MCG_CAP.CNT >21
455H	1109	IA32_MC21_STATUS	MC21_STATUS	If IA32_MCG_CAP.CNT >21
456H	1110	IA32_MC21_ADDR ¹	MC21_ADDR	If IA32_MCG_CAP.CNT >21
457H	1111	IA32_MC21_MISC	MC21_MISC	If IA32_MCG_CAP.CNT >21
458H	1112	IA32_MC22_CTL	MC22_CTL	If IA32_MCG_CAP.CNT >22
459H	1113	IA32_MC22_STATUS	MC22_STATUS	If IA32_MCG_CAP.CNT >22
45AH	1114	IA32_MC22_ADDR ¹	MC22_ADDR	If IA32_MCG_CAP.CNT >22
45BH	1115	IA32_MC22_MISC	MC22_MISC	If IA32_MCG_CAP.CNT >22
45CH	1116	IA32_MC23_CTL	MC23_CTL	If IA32_MCG_CAP.CNT >23
45DH	1117	IA32_MC23_STATUS	MC23_STATUS	If IA32_MCG_CAP.CNT >23

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
45EH	1118	IA32_MC23_ADDR ¹	MC23_ADDR	If IA32_MCG_CAP.CNT >23
45FH	1119	IA32_MC23_MISC	MC23_MISC	If IA32_MCG_CAP.CNT >23
460H	1120	IA32_MC24_CTL	MC24_CTL	If IA32_MCG_CAP.CNT >24
461H	1121	IA32_MC24_STATUS	MC24_STATUS	If IA32_MCG_CAP.CNT >24
462H	1122	IA32_MC24_ADDR ¹	MC24_ADDR	If IA32_MCG_CAP.CNT >24
463H	1123	IA32_MC24_MISC	MC24_MISC	If IA32_MCG_CAP.CNT >24
464H	1124	IA32_MC25_CTL	MC25_CTL	If IA32_MCG_CAP.CNT >25
465H	1125	IA32_MC25_STATUS	MC25_STATUS	If IA32_MCG_CAP.CNT >25
466H	1126	IA32_MC25_ADDR ¹	MC25_ADDR	If IA32_MCG_CAP.CNT >25
467H	1127	IA32_MC25_MISC	MC25_MISC	If IA32_MCG_CAP.CNT >25
468H	1128	IA32_MC26_CTL	MC26_CTL	If IA32_MCG_CAP.CNT >26
469H	1129	IA32_MC26_STATUS	MC26_STATUS	If IA32_MCG_CAP.CNT >26
46AH	1130	IA32_MC26_ADDR ¹	MC26_ADDR	If IA32_MCG_CAP.CNT >26
46BH	1131	IA32_MC26_MISC	MC26_MISC	If IA32_MCG_CAP.CNT >26
46CH	1132	IA32_MC27_CTL	MC27_CTL	If IA32_MCG_CAP.CNT >27
46DH	1133	IA32_MC27_STATUS	MC27_STATUS	If IA32_MCG_CAP.CNT >27
46EH	1134	IA32_MC27_ADDR ¹	MC27_ADDR	If IA32_MCG_CAP.CNT >27
46FH	1135	IA32_MC27_MISC	MC27_MISC	If IA32_MCG_CAP.CNT >27
470H	1136	IA32_MC28_CTL	MC28_CTL	If IA32_MCG_CAP.CNT >28
471H	1137	IA32_MC28_STATUS	MC28_STATUS	If IA32_MCG_CAP.CNT >28
472H	1138	IA32_MC28_ADDR ¹	MC28_ADDR	If IA32_MCG_CAP.CNT >28
473H	1139	IA32_MC28_MISC	MC28_MISC	If IA32_MCG_CAP.CNT >28
480H	1152	IA32_VMX_BASIC	Reporting Register of Basic VMX Capabilities (R/O) See Appendix A.1, "Basic VMX Information."	If CPUID.01H:ECX.[5] = 1
481H	1153	IA32_VMX_PINBASED_CTL	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
482H	1154	IA32_VMX_PROCBASED_CTL	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
483H	1155	IA32_VMX_EXIT_CTL	Capability Reporting Register of Primary VM-Exit Controls (R/O) See Appendix A.4.1, "Primary VM-Exit Controls."	If CPUID.01H:ECX.[5] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
484H	1156	IA32_VMX_ENTRY_CTL5	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If CPUID.01H:ECX.[5] = 1
485H	1157	IA32_VMX_MISC	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data."	If CPUID.01H:ECX.[5] = 1
486H	1158	IA32_VMX_CRO_FIXED0	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CRO."	If CPUID.01H:ECX.[5] = 1
487H	1159	IA32_VMX_CRO_FIXED1	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CRO."	If CPUID.01H:ECX.[5] = 1
488H	1160	IA32_VMX_CR4_FIXED0	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[5] = 1
489H	1161	IA32_VMX_CR4_FIXED1	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[5] = 1
48AH	1162	IA32_VMX_VMCS_ENUM	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration."	If CPUID.01H:ECX.[5] = 1
48BH	1163	IA32_VMX_PROCBASED_CTL52	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."	If (CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTL5[63])
48CH	1164	IA32_VMX_EPT_VPID_CAP	Capability Reporting Register of EPT and VPID (R/O) See Appendix A.10, "VPID and EPT Capabilities."	If (CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTL5[63] && (IA32_VMX_PROCBASED_CTL52[33] IA32_VMX_PROCBASED_CTL52[37]))
48DH	1165	IA32_VMX_TRUE_PINBASED_CTL5	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If (CPUID.01H:ECX.[5] && IA32_VMX_BASIC[55])
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTL5	Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If(CPUID.01H:ECX.[5] && IA32_VMX_BASIC[55])
48FH	1167	IA32_VMX_TRUE_EXIT_CTL5	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Appendix A.4, "VM-Exit Controls."	If(CPUID.01H:ECX.[5] && IA32_VMX_BASIC[55])

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
490H	1168	IA32_VMX_TRUE_ENTRY_CTL5	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If (CPUID.01H:ECX.[5] && IA32_VMX_BASIC[55])
491H	1169	IA32_VMX_VMFUNC	Capability Reporting Register of VM-Function Controls (R/O)	If (CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTL5[63] && IA32_VMX_PROCBASED_CTL52[45])
492H	1170	IA32_VMX_PROCBASED_CTL53	Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.4, "Tertiary Processor-Based VM-Execution Controls."	If (CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTL5[49])
493H	1171	IA32_VMX_EXIT_CTL52	Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Appendix A.4.2, "Secondary VM-Exit Controls."	If (CPUID.01H:ECX.[5] && IA32_VMX_EXIT_CTL5[63])
4C1H	1217	IA32_A_PMC0	Full Width Writable IA32_PMC0 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 0) && IA32_PERF_CAPABILITIES[13] = 1
4C2H	1218	IA32_A_PMC1	Full Width Writable IA32_PMC1 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 1) && IA32_PERF_CAPABILITIES[13] = 1
4C3H	1219	IA32_A_PMC2	Full Width Writable IA32_PMC2 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 2) && IA32_PERF_CAPABILITIES[13] = 1
4C4H	1220	IA32_A_PMC3	Full Width Writable IA32_PMC3 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 3) && IA32_PERF_CAPABILITIES[13] = 1
4C5H	1221	IA32_A_PMC4	Full Width Writable IA32_PMC4 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 4) && IA32_PERF_CAPABILITIES[13] = 1
4C6H	1222	IA32_A_PMC5	Full Width Writable IA32_PMC5 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 5) && IA32_PERF_CAPABILITIES[13] = 1
4C7H	1223	IA32_A_PMC6	Full Width Writable IA32_PMC6 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 6) && IA32_PERF_CAPABILITIES[13] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
4C8H	1224	IA32_A_PMC7	Full Width Writable IA32_PMC7 Alias (R/W)	If CPUID.0AH: EAX[15:8] > 7) && IA32_PERF_CAPABILITIES[13] = 1
4D0H	1232	IA32_MCG_EXT_CTL	Allows software to signal some MCEs to only a single logical processor in the system. (R/W) See Section 16.3.1.4, "IA32_MCG_EXT_CTL MSR."	If IA32_MCG_CAP.LMCE_P = 1
		0	LMCE_EN	
		63:1	Reserved	
500H	1280	IA32_SGX_SVN_STATUS	Status and SVN Threshold of SGX Support for ACM (R/O).	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		0	Lock	See Section 39.11.3, "Interactions with Authenticated Code Modules (ACMs)."
		15:1	Reserved	
		23:16	SGX_SVN_SINIT	See Section 39.11.3, "Interactions with Authenticated Code Modules (ACMs)."
		63:24	Reserved	
560H	1376	IA32_RTIT_OUTPUT_BASE	Trace Output Base Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && (CPUID.(EAX=14H,ECX=0):ECX[0] = 1) (CPUID.(EAX=14H,ECX=0):ECX[2] = 1))
		6:0	Reserved	
		MAXPHYADDR ⁴ -1:7	Base physical address.	
		63:MAXPHYADDR	Reserved	
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Trace Output Mask Pointers Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && (CPUID.(EAX=14H,ECX=0):ECX[0] = 1) (CPUID.(EAX=14H,ECX=0):ECX[2] = 1))
		6:0	Reserved	
		31:7	MaskOrTableOffset	
		63:32	Output Offset	
570H	1392	IA32_RTIT_CTL	Trace Control Register (R/W)	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		0	TraceEn	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		1	CYCEn	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		2	OS	
		3	User	
		4	PwrEvtEn	If (CPUID.(EAX=07H, ECX=1):EBX[5] = 1)
		5	FUPonPTW	If (CPUID.(EAX=07H, ECX=1):EBX[4] = 1)
		6	FabricEn	If (CPUID.(EAX=07H, ECX=0):ECX[3] = 1)
		7	CR3Filter	If (CPUID.(EAX=14H, ECX=0):EBX[0] = 1)
		8	ToPA	
		9	MTCEn	If (CPUID.(EAX=07H, ECX=0):EBX[3] = 1)
		10	TSCEn	
		11	DisRETC	
		12	PTWEn	If (CPUID.(EAX=07H, ECX=1):EBX[4] = 1)
		13	BranchEn	
		17:14	MTCFreq	If (CPUID.(EAX=07H, ECX=0):EBX[3] = 1)
		18	Reserved, must be zero.	
		22:19	CycThresh	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		23	Reserved, must be zero.	
		27:24	PSBFreq	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		30:28	Reserved, must be zero.	
		31	EventEn	If (CPUID.(EAX=14H, ECX=0):EBX[7] = 1)
		35:32	ADDR0_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		39:36	ADDR1_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		43:40	ADDR2_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:44	ADDR3_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		54:48	Reserved, must be zero.	
		55	DisTNT	If (CPUID.(EAX=14H, ECX=0):EBX[8] = 1)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		56	InjectPsbPmiOnEnable	If (CPUID.(EAX=07H, ECX=1);EBX[6] = 1)
		63:57	Reserved, must be zero.	
571H	1393	IA32_RTIT_STATUS	Tracing Status Register (R/W)	If (CPUID.(EAX=07H, ECX=0);EBX[25] = 1)
		0	FilterEn (writes ignored)	If (CPUID.(EAX=07H, ECX=0);EBX[2] = 1)
		1	ContexEn (writes ignored)	
		2	TriggerEn (writes ignored)	
		3	Reserved	
		4	Error	
		5	Stopped	
		6	PendPSB	If (CPUID.(EAX=07H, ECX=0);EBX[6] = 1)
		7	PendToPAPMI	If (CPUID.(EAX=07H, ECX=0);EBX[6] = 1)
		31:8	Reserved, must be zero.	
		48:32	PacketByteCnt	If (CPUID.(EAX=07H, ECX=0);EBX[1] > 3)
		63:49	Reserved	
572H	1394	IA32_RTIT_CR3_MATCH	Trace Filter CR3 Match Register (R/W)	If (CPUID.(EAX=07H, ECX=0);EBX[25] = 1)
		4:0	Reserved	
		63:5	CR3[63:5] value to match.	
580H	1408	IA32_RTIT_ADDR0_A	Region 0 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1);EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
581H	1409	IA32_RTIT_ADDR0_B	Region 0 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1);EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
582H	1410	IA32_RTIT_ADDR1_A	Region 1 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1);EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	
583H	1411	IA32_RTIT_ADDR1_B	Region 1 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1);EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
584H	1412	IA32_RTIT_ADDR2_A	Region 2 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	
		63:48	SignExt_VA	
585H	1413	IA32_RTIT_ADDR2_B	Region 2 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	
		63:48	SignExt_VA	
586H	1414	IA32_RTIT_ADDR3_A	Region 3 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		47:0	Virtual Address	
		63:48	SignExt_VA	
587H	1415	IA32_RTIT_ADDR3_B	Region 3 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		47:0	Virtual Address	
		63:48	SignExt_VA	
600H	1536	IA32_DS_AREA	DS Save Area (R/W) Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 20.6.3.4, "Debug Store (DS) Mechanism."	If (CPUID.01H:EDX.DS[21] = 1)
		63:0	The linear address of the first byte of the DS buffer management area, if IA-32e mode is active.	
		31:0	The linear address of the first byte of the DS buffer management area, if not in IA-32e mode.	
		63:32	Reserved if not in IA-32e mode.	
6A0H	1696	IA32_U_CET	Configure User Mode CET (R/W)	Bits 1:0 are defined if CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1. Bits 5:2 and bits 63:10 are defined if CPUID.(EAX=07H, ECX=0H):EDX.CET_IBT[20] = 1.
		0	SH_STK_EN: When set to 1, enable shadow stacks at CPL3.	
		1	WR_SHSTK_EN: When set to 1, enables the WRSSD/WRSSQ instructions.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		2	ENDBR_EN: When set to 1, enables indirect branch tracking.	
		3	LEG_IW_EN: Enable legacy compatibility treatment for indirect branch tracking.	
		4	NO_TRACK_EN: When set to 1, enables use of no-track prefix for indirect branch tracking.	
		5	SUPPRESS_DIS: When set to 1, disables suppression of CET indirect branch tracking on legacy compatibility.	
		9:6	Reserved; must be zero.	
		10	SUPPRESS: When set to 1, indirect branch tracking is suppressed. This bit can be written to 1 only if TRACKER is written as IDLE.	
		11	TRACKER: Value of the indirect branch tracking state machine. Values: IDLE (0), WAIT_FOR_ENDBRANCH(1).	
		63:12	EB_LEG_BITMAP_BASE: Linear address bits 63:12 of a legacy code page bitmap used for legacy compatibility when indirect branch tracking is enabled. If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	
6A2H	1698	IA32_S_CET	Configure Supervisor Mode CET (R/W)	See IA32_U_CET (6A0H) for reference; similar format.
6A4H	1700	IA32_PLO_SSP	Linear address to be loaded into SSP on transition to privilege level 0. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
6A5H	1701	IA32_PL1_SSP	Linear address to be loaded into SSP on transition to privilege level 1. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6A6H	1702	IA32_PL2_SSP	Linear address to be loaded into SSP on transition to privilege level 2. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6A7H	1703	IA32_PL3_SSP	Linear address to be loaded into SSP on transition to privilege level 3. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6A8H	1704	IA32_INTERRUPT_SSP_TABLE_ADDR	Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) This MSR is not present on processors that do not support Intel 64 architecture. This field cannot represent a non-canonical address.	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6E0H	1760	IA32_TSC_DEADLINE	TSC Target of Local APIC's TSC Deadline Mode (R/W)	If CPUID.01H:ECX.[24] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
6E1H	1761	IA32_PKRS	Specifies the PK permissions associated with each protection domain for supervisor pages (R/W)	If CPUID.(EAX=07H, ECX=0H):ECX.PKS [31] = 1
		31:0	For domain i (i between 0 and 15), bits 2i and 2i+1 contain the AD and WD permissions, respectively.	
		63:32	Reserved.	
770H	1904	IA32_PM_ENABLE	Enable/disable HWP (R/W)	If CPUID.06H:EAX.[7] = 1
		0	HWP_ENABLE (R/W1-Once) See Section 15.4.2, "Enabling HWP."	If CPUID.06H:EAX.[7] = 1
		63:1	Reserved	
771H	1905	IA32_HWP_CAPABILITIES	HWP Performance Range Enumeration (R/O)	If CPUID.06H:EAX.[7] = 1
		7:0	Highest_Performance See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX.[7] = 1
		15:8	Guaranteed_Performance See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX.[7] = 1
		23:16	Most_Efficient_Performance See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities".	If CPUID.06H:EAX.[7] = 1
		31:24	Lowest_Performance See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX.[7] = 1
		63:32	Reserved	
772H	1906	IA32_HWP_REQUEST_PKG	Power Management Control Hints for All Logical Processors in a Package (R/W)	If CPUID.06H:EAX.[11] = 1
		7:0	Minimum_Performance See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[11] = 1
		15:8	Maximum_Performance See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[11] = 1
		23:16	Desired_Performance See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[11] = 1
		31:24	Energy_Performance_Preference See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	Activity_Window See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[9] = 1
		63:42	Reserved	
773H	1907	IA32_HWP_INTERRUPT	Control HWP Native Interrupts (R/W)	If CPUID.06H:EAX.[8] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	EN_Guaranteed_Performance_Change See Section 15.4.6, "HWP Notifications."	If CPUID.06H:EAX.[8] = 1
		1	EN_Excursion_Minimum See Section 15.4.6, "HWP Notifications."	If CPUID.06H:EAX.[8] = 1
		63:2	Reserved	
774H	1908	IA32_HWP_REQUEST	Power Management Control Hints to a Logical Processor (R/W)	If CPUID.06H:EAX.[7] = 1
		7:0	Minimum_Performance See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[7] = 1
		15:8	Maximum_Performance See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[7] = 1
		23:16	Desired_Performance See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[7] = 1
		31:24	Energy_Performance_Preference See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	Activity_Window See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[9] = 1
		42	Package_Control See Section 15.4.4, "Managing HWP."	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[11] = 1
		63:43	Reserved	
775H	1909	IA32_PECI_HWP_REQUEST_INFO	IA32_PECI_HWP_REQUEST_INFO	
		7:0	Minimum Performance (MINIMUM_PERFORMANCE): Used by OS to read the latest value of Peci minimum performance input. Default value is 0.	
		15:8	Maximum Performance (MAXIMUM_PERFORMANCE): Used by OS to read the latest value of Peci maximum performance input. Default value is 0.	
		23:16	Reserved.	
		31:24	Energy Performance Preference (ENERGY_PERFORMANCE_PREFERENCE): Used by OS to read the latest value of Peci Energy Performance Preference input. Default value is 0.	
		59:32	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		60	EPP Peci Override (EPP_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Energy Performance Preference input. If set to '1', Peci is overriding the Energy Performance Preference input. If clear (0), OS has control over Energy Performance Preference input. Default value is 0.	
		61	Reserved.	
		62	Max Peci Override (MAX_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Maximum Performance input. If set to '1', Peci is overriding the Maximum Performance input. If clear (0), OS has control over Maximum Performance input. Default value is 0.	
		63	Min Peci Override (MIN_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Minimum Performance input. If set to '1', Peci is overriding the Minimum Performance input. If clear (0), OS has control over Minimum Performance input. Default value is 0.	
776H	1910	IA32_HWP_CTL	IA32_HWP_CTL	If CPUID.06H:EAX.[22] = 1
		0	PKG_CTL_POLARITY Defines which HWP Request MSR is used whether Thread level or package level. When package MSR is used, the thread MSR valid bits define which thread MSR fields override the package. Default value is 0.	If CPUID.06H:EAX.[22] = 1
		63:1	Reserved	
777H	1911	IA32_HWP_STATUS	Log bits indicating changes to Guaranteed & excursions to Minimum (R/W)	If CPUID.06H:EAX.[7] = 1
		0	Guaranteed_Performance_Change (R/WCO) See Section 15.4.5, "HWP Feedback."	If CPUID.06H:EAX.[7] = 1
		1	Reserved	
		2	Excursion_To_Minimum (R/WCO) See Section 15.4.5, "HWP Feedback."	If CPUID.06H:EAX.[7] = 1
		63:3	Reserved	
802H	2050	IA32_X2APIC_APICID	x2APIC ID Register (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
803H	2051	IA32_X2APIC_VERSION	x2APIC Version Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
808H	2056	IA32_X2APIC_TPR	x2APIC Task Priority Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80AH	2058	IA32_X2APIC_PPR	x2APIC Processor Priority Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80BH	2059	IA32_X2APIC_EOI	x2APIC EOI Register (w/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80DH	2061	IA32_X2APIC_LDR	x2APIC Logical Destination Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80FH	2063	IA32_X2APIC_SIVR	x2APIC Spurious Interrupt Vector Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
810H	2064	IA32_X2APIC_ISR0	x2APIC In-Service Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
811H	2065	IA32_X2APIC_ISR1	x2APIC In-Service Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
812H	2066	IA32_X2APIC_ISR2	x2APIC In-Service Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
813H	2067	IA32_X2APIC_ISR3	x2APIC In-Service Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
814H	2068	IA32_X2APIC_ISR4	x2APIC In-Service Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
815H	2069	IA32_X2APIC_ISR5	x2APIC In-Service Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
816H	2070	IA32_X2APIC_ISR6	x2APIC In-Service Register Bits 223:192 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
817H	2071	IA32_X2APIC_ISR7	x2APIC In-Service Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
818H	2072	IA32_X2APIC_TMRO	x2APIC Trigger Mode Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
819H	2073	IA32_X2APIC_TMR1	x2APIC Trigger Mode Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81AH	2074	IA32_X2APIC_TMR2	x2APIC Trigger Mode Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81BH	2075	IA32_X2APIC_TMR3	x2APIC Trigger Mode Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81CH	2076	IA32_X2APIC_TMR4	x2APIC Trigger Mode Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81DH	2077	IA32_X2APIC_TMR5	x2APIC Trigger Mode Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81EH	2078	IA32_X2APIC_TMR6	x2APIC Trigger Mode Register Bits 223:192 (R/O)	If (CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1)
81FH	2079	IA32_X2APIC_TMR7	x2APIC Trigger Mode Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
820H	2080	IA32_X2APIC_IRR0	x2APIC Interrupt Request Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
821H	2081	IA32_X2APIC_IRR1	x2APIC Interrupt Request Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
822H	2082	IA32_X2APIC_IRR2	x2APIC Interrupt Request Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
823H	2083	IA32_X2APIC_IRR3	x2APIC Interrupt Request Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
824H	2084	IA32_X2APIC_IRR4	x2APIC Interrupt Request Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
825H	2085	IA32_X2APIC_IRR5	x2APIC Interrupt Request Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
826H	2086	IA32_X2APIC_IRR6	x2APIC Interrupt Request Register Bits 223:192 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
827H	2087	IA32_X2APIC_IRR7	x2APIC Interrupt Request Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
828H	2088	IA32_X2APIC_ESR	x2APIC Error Status Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
82FH	2095	IA32_X2APIC_LVT_CMCI	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
830H	2096	IA32_X2APIC_ICR	x2APIC Interrupt Command Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
832H	2098	IA32_X2APIC_LVT_TIMER	x2APIC LVT Timer Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
833H	2099	IA32_X2APIC_LVT_THERMAL	x2APIC LVT Thermal Sensor Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
834H	2100	IA32_X2APIC_LVT_PMI	x2APIC LVT Performance Monitor Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
835H	2101	IA32_X2APIC_LVT_LINT0	x2APIC LVT LINT0 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
836H	2102	IA32_X2APIC_LVT_LINT1	x2APIC LVT LINT1 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
837H	2103	IA32_X2APIC_LVT_ERROR	x2APIC LVT Error Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
838H	2104	IA32_X2APIC_INIT_COUNT	x2APIC Initial Count Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
839H	2105	IA32_X2APIC_CUR_COUNT	x2APIC Current Count Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
83EH	2110	IA32_X2APIC_DIV_CONF	x2APIC Divide Configuration Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
83FH	2111	IA32_X2APIC_SELF_IPI	x2APIC Self IPI Register (w/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
981H	2433	IA32_TME_CAPABILITY	Memory Encryption Capability MSR	If CPUID.07H:ECX.[13] = 1
		0	Support for AES-XTS 128-bit encryption algorithm. (NIST standard)	
		1	Support for AES-XTS 128-bit encryption with integrity algorithm.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		2	Support for AES-XTS 256-bit encryption algorithm.	
		30:3	Reserved.	
		31	TME encryption bypass supported.	
		35:32	MK_TME_MAX_KEYID_BITS Number of bits which can be allocated for usage as key identifiers for multi-key memory encryption. 4 bits allow for a maximum value of 15, which could address 32K keys. Zero if MKTME is not supported.	
		50:36	MK_TME_MAX_KEYS Indicates the maximum number of keys which are available for usage. This value may not be a power of 2. KeyID 0 is specially reserved and is not accounted for in this field.	
		63:51	Reserved.	
982H	2434	IA32_TME_ACTIVATE	Memory Encryption Activation MSR This MSR is used to lock the MSRs listed below. Any write to the following MSRs will be ignored after they are locked. The lock is reset when CPU is reset. <ul style="list-style-type: none"> ▪ IA32_TME_ACTIVATE ▪ IA32_TME_EXCLUDE_MASK ▪ IA32_TME_EXCLUDE_BASE Note that IA32_TME_EXCLUDE_MASK and IA32_TME_EXCLUDE_BASE must be configured before IA32_TME_ACTIVATE.	If CPUID.07H:ECX.[13] = 1
		0	Lock R/O - Will be set upon successful WRMSR (or first SMI); written value ignored.	
		1	Hardware Encryption Enable This bit also enables MKTME; MKTME cannot be enabled without enabling encryption hardware.	
		2	Key Select 0: Create a new TME key (expected cold/warm boot). 1: Restore the TME key from storage (Expected when resume from standby).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		3	Save TME Key for Standby Save key into storage to be used when resume from standby. Note: This may not be supported in all processors.	
		7:4	TME Policy/Encryption Algorithm Only algorithms enumerated in IA32_TME_CAPABILITY are allowed. For example: 0000 - AES-XTS-128. 0001 - AES-XTS-128 with integrity. 0010 - AES-XTS-256. Other values are invalid.	
		30:8	Reserved.	
		31	TME Encryption Bypass Enable When encryption hardware is enabled: <ul style="list-style-type: none"> ▪ Total Memory Encryption is enabled using a CPU generated ephemeral key based on a hardware random number generator when this bit is set to 0. ▪ Total Memory Encryption is bypassed (no encryption/decryption for KeyID0) when this bit is set to 1. Software must inspect Hardware Encryption Enable (bit 1) and TME encryption bypass Enable (bit 31) to determine if TME encryption is enabled.	
		35:32	MK_TME_KEYID_BITS Reserved if MKTME is not enumerated, otherwise: The number of key identifier bits to allocate to MKTME usage. Similar to enumeration, this is an encoded value. Writing a value greater than MK_TME_MAX_KEYID_BITS will result in #GP. Writing a non-zero value to this field will #GP if bit 1 of EAX (Hardware Encryption Enable) is not also set to '1, as encryption hardware must be enabled to use MKTME. Example: To support 255 keys, this field would be set to a value of 8.	
		47:36	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:48	MK_TME_CRYPTO_ALGS Reserved if MKTME is not enumerated, otherwise: Bit 48: AES-XTS 128. Bit 49: AES-XTS 128 with integrity. Bit 50: AES-XTS 256. Bit 63:51: Reserved (#GP) Bitmask for BIOS to set which encryption algorithms are allowed for MKTME, would be later enforced by the key loading ISA ('1 = allowed).	
983H	2435	IA32_TME_EXCLUDE_MASK	Memory Encryption Exclude Mask	If CPUID.07H:ECX.[13] = 1
		10:0	Reserved.	
		11	Enable: When set to '1', then TME_EXCLUDE_BASE and TME_EXCLUDE_MASK are used to define an exclusion region for TME/MKTME (for KeyID=0).	
		MAXPHYSADDR-1:12	TMEEMASK: This field indicates the bits that must match TMEEBASE in order to qualify as a TME/MKTME (for KeyID=0) exclusion memory range access.	
		63:MAXPHYSADDR	Reserved; must be zero.	
984H	2436	IA32_TME_EXCLUDE_BASE	Memory Encryption Exclude Base	IF CPUID.07H:ECX.[13] = 1
		11:0	Reserved.	
		MAXPHYSADDR-1:12	TMEEBASE: Base physical address to be excluded for TME/MKTME (for KeyID=0) encryption.	
		63:MAXPHYSADDR	Reserved; must be zero.	
985H	2437	IA32_UINTR_RR	User Interrupt Request Register (R/W)	IF CPUID.07H.01H:EDX[13]=1
		63:0	UIRR Bitmap of requested user interrupt vectors.	
986H	2438	IA32_UINTR_HANDLER	User Interrupt Handler Address (R/W)	IF CPUID.07H.01H:EDX[13]=1
		63:0	UIHANDLER User interrupt handler linear address.	
987H	2439	IA32_UINTR_STACKADJUST	User Interrupt Stack Adjustment (R/W)	IF CPUID.07H.01H:EDX[13]=1
		0	LOAD_RSP User interrupt stack mode.	
		2:1	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:3	STACK_ADJUST Stack adjust value.	
988H	2440	IA32_UINTR_MISC	User-Interrupt Target-Table Size and Notification Vector (R/W)	If CPUID.07H.01H:EDX[13]=1
		31:0	UITTSZ The highest index of a valid entry in the user-interrupt target table. Valid entries are indices 0..UITTSZ (inclusive).	
		39:32	UINV User-interrupt notification vector.	
		63:40	Reserved.	
989H	2441	IA32_UINTR_PD	User Interrupt PID Address (R/W)	If CPUID.07H.01H:EDX[13]=1
		5:0	Reserved.	
		63:6	UPIDADDR User-interrupt notification processing accesses a UPID at this linear address.	
98AH	2442	IA32_UINTR_TT	User-Interrupt Target Table (R/W)	If CPUID.07H.01H:EDX[13]=1
		0	SENDUIPI_ENABLE User-interrupt target table is valid.	
		3:1	Reserved.	
		63:4	UITTADDR User-interrupt target table base linear address.	
990H	2448	IA32_COPY_STATUS ⁵	Status of Most Recent Platform to Local or Local to Platform Copies (R/O)	If ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		0	IWKEY_COPY_SUCCESSFUL Status of most recent copy to or from IwKeyBackup	If ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		63:1	Reserved	
991H	2449	IA32_IWKEYBACKUP_STATUS ⁵	Information about IwKeyBackup Register (R/O)	If ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Backup/Restore Valid Cleared when a write to <code>lWKeyBackup</code> is initiated, and then set when the latest write of <code>lWKeyBackup</code> has been written to storage that persists across S3/S4 sleep state. If S3/S4 is entered between when an <code>lWKeyBackup</code> write occurs and when this bit is set, then <code>lWKeyBackup</code> may not be recovered after S3/S4 exit. During S3/S4 sleep state exit (system wake up), this bit is cleared. It is set again when <code>lWKeyBackup</code> is restored from persistent storage and thus available to be copied to <code>lWKey</code> using <code>IA32_COPY_PLATFORM_TO_LOCAL</code> MSR. Another write to <code>lWKeyBackup</code> (via <code>IA32_COPY_LOCAL_TO_PLATFORM</code> MSR) may fail if a previous write has not yet set this bit.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		1	Reserved	
		2	Backup Key Storage Read/Write Error Updated prior to backup/restore valid being set. Set when an error is encountered while backing up or restoring a key to persistent storage.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		3	<code>lWKeyBackup</code> Consumed Set after the previous backup operation has been consumed by the platform. This does not indicate that the system is ready for a second <code>lWKeyBackup</code> write as the previous <code>lWKeyBackup</code> write may still need to set Backup/restore valid.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		63:4	Reserved	
C80H	3200	IA32_DEBUG_INTERFACE	Silicon Debug Feature Control (R/W)	If CPUID.01H:ECX.[11] = 1
		0	Enable (R/W) BIOS set 1 to enable Silicon debug features. Default is 0.	If CPUID.01H:ECX.[11] = 1
		29:1	Reserved	
		30	Lock (R/W): If 1, locks any further change to the MSR. The lock bit is set automatically on the first SMI assertion even if not explicitly set by BIOS. Default is 0.	If CPUID.01H:ECX.[11] = 1
		31	Debug Occurred (R/O): This “sticky bit” is set by hardware to indicate the status of bit 0. Default is 0.	If CPUID.01H:ECX.[11] = 1
		63:32	Reserved	
C81H	3201	IA32_L3_QOS_CFG	L3 QOS Configuration (R/W)	If (CPUID.(EAX=10H, ECX=1);ECX.[2] = 1)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Enable (R/W) Set 1 to enable L3 CAT masks and COS to operate in Code and Data Prioritization (CDP) mode.	
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
C82H	3202	IA32_L2_QOS_CFG	L2 QOS Configuration (R/W)	If (CPUID.(EAX=10H, ECX=2):ECX.[2] = 1)
		0	Enable (R/W) Set 1 to enable L2 CAT masks and COS to operate in Code and Data Prioritization (CDP) mode.	
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
C8DH	3213	IA32_QM_EVTSEL	Monitoring Event Select Register (R/W)	If (CPUID.(EAX=07H, ECX=0):EBX.[12] = 1)
		7:0	Event ID: ID of a supported monitoring event to report via IA32_QM_CTR.	
		31: 8	Reserved	
		N+31:32	Resource Monitoring ID: ID for monitoring hardware to report monitored data via IA32_QM_CTR.	N = Ceil (Log ₂ (CPUID.(EAX= 0FH, ECX=0H):EBX[31:0] + 1))
		63:N+32	Reserved	
C8EH	3214	IA32_QM_CTR	Monitoring Counter Register (R/O)	If (CPUID.(EAX=07H, ECX=0):EBX.[12] = 1)
		61:0	Resource Monitored Data	
		62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
		63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
C8FH	3215	IA32_PQR_ASSOC	Resource Association Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[12] = 1) or (CPUID.(EAX=07H, ECX=0):EBX[15] = 1))
		N-1:0	Resource Monitoring ID (R/W): ID for monitoring hardware to track internal operation, e.g., memory access.	N = Ceil (Log ₂ (CPUID.(EAX= 0FH, ECX=0H):EBX[31:0] + 1))
		31:N	Reserved	
		63:32	COS (R/W): The class of service (COS) to enforce (on writes); returns the current COS when read.	If (CPUID.(EAX=07H, ECX=0):EBX.[15] = 1)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C90H - D8FH	3216 - 3471	Reserved MSR Address Space for CAT Mask Registers	See Section 18.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology."	
C90H	3216	IA32_L3_MASK_0	L3 CAT Mask for COS0 (R/W)	If (CPUID.(EAX=10H, ECX=0H):EBX[1] != 0)
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
C90H+n	3216+n	IA32_L3_MASK_n	L3 CAT Mask for COSn (R/W)	n = CPUID.(EAX=10H, ECX=1H):EDX[15:0]
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D10H - D4FH	3344 - 3407	Reserved MSR Address Space for L2 CAT Mask Registers	See Section 18.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology."	
D10H	3344	IA32_L2_MASK_0	L2 CAT Mask for COS0 (R/W)	If (CPUID.(EAX=10H, ECX=0H):EBX[2] != 0)
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D10H+n	3344+n	IA32_L2_MASK_n	L2 CAT Mask for COSn (R/W)	n = CPUID.(EAX=10H, ECX=2H):EDX[15:0]
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D90H	3472	IA32_BNDCFGS	Supervisor State of MPX Configuration (R/W)	If (CPUID.(EAX=07H, ECX=0H):EBX[14] = 1)
		0	EN: Enable Intel MPX in supervisor mode.	
		1	BNDPRESERVE: Preserve the bounds registers for near branch instructions in the absence of the BND prefix.	
		11:2	Reserved, must be zero.	
		63:12	Base Address of Bound Directory.	
D91H	3473	IA32_COPY_LOCAL_TO_PLATFORM ⁵	Copy Local State to Platform State (w)	If ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H):ECX[23] = 1))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	IAKeyBackup Copy IAKey to IAKeyBackup	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		63:1	Reserved	
D92H	3474	IA32_COPY_PLATFORM_TO_LOCAL ⁵	Copy Platform State to Local State (W)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		0	IAKeyBackup Copy IAKeyBackup to IAKey	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		63:1	Reserved	
D93H	3475	IA32_PASID	Process Address Space Identifier (R/W)	
		19:0	Process address space identifier (PASID). Specifies the PASID of the currently running software thread.	
		30:20	Reserved	
		31	Valid. Execution of ENQCMD causes a #GP if this bit is clear.	
		63:32	Reserved	
DA0H	3488	IA32_XSS	Extended Supervisor State Mask (R/W)	If (CPUID.(0DH, 1):EAX.[3] = 1)
		7:0	Reserved.	
		8	PT State (R/W)	
		9	Reserved.	
		10	PASID State (R/W)	
		11	CET_U State (R/W)	
		12	CET_S State (R/W)	
		13	HDC State (R/W)	
		14	UINTR State (R/W)	
		15	LBR State (R/W)	
		16	HWP State (R/W)	
		63:17	Reserved.	
DB0H	3504	IA32_PKG_HDC_CTL	Package Level Enable/disable HDC (R/W)	If CPUID.06H:EAX.[13] = 1
		0	HDC_Pkg_Enable (R/W) Force HDC idling or wake up HDC-idled logical processors in the package. See Section 15.5.2, "Package level Enabling HDC."	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved	
DB1H	3505	IA32_PM_CTL1	Enable/disable HWP (R/W)	If CPUID.06H:EAX.[13] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	HDC_Allow_Block (R/W) Allow/Block this logical processor for package level HDC control. See Section 15.5.3.	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved	
DB2H	3506	IA32_THREAD_STALL	Per-Logical_Processor_ID HDC Idle Residency (R/O)	If CPUID.06H:EAX.[13] = 1
		63:0	Stall_Cycle_Cnt (R/W) Stalled cycles due to HDC forced idle on this logical processor. See Section 15.5.4.1.	If CPUID.06H:EAX.[13] = 1
1200H - 121FH	4608 - 4639	IA32_LBR_x_INFO	Last Branch Record Entry X Info Register (R/W) An attempt to read or write IA32_LBR_x_INFO such that x ≥ IA32_LBR_DEPTH.DEPTH will #GP.	
		15:0	CYC_CNT The elapsed CPU cycles (saturating) since the last LBR was recorded. See Section 18.1.3.3.	Reset Value: 0
		55:16	Undefined, may be zero or non-zero. Writes of non- zero values do not fault, but reads may return a different value.	Reset Value: 0
		59:56	BR_TYPE The branch type recorded by this LBR. Encodings: 0000B: COND 0001B: JMP Indirect 0010B: JMP Direct 0011B: CALL Indirect 0100B: CALL Direct 0101B: RET 011xB: Reserved 1xxxB: Other Branch	Reset Value: 0
		60	CYC_CNT_VALID CYC_CNT value is valid. See Section 19.1.3.3.	Reset Value: 0
		61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel TSX (CPUID.07H:EBX.HLE[bit 4]=0 and CPUID.07H:EBX.RTM[bit 11]=0), this bit is undefined.	Reset Value: 0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	Reset Value: 0
		63	MISPRED The recorded branch direction (conditional branch) or target (indirect branch) was mispredicted.	Reset Value: 0
1406H	5126	IA32_MCU_CONTROL	MCU Control (R/W) Controls the behavior of the Microcode Update Trigger MSR, IA32_BIOS_UPDT_TRIG.	If CPUID.07H.0H:EDX[29]=1 && MSR.IA32_ARCH_CAPABILITIES.MCU_CONTROL=1
		0	LOCK Once set, further writes to this MSR will cause a #GP(0) fault. Bypassed during SMM if EN_SMM_BYPASS (bit 2) is set.	
		1	DIS_MCU_LOAD If this bit is set on a given logical processor, then any subsequent attempts to load a microcode update by that logical processor will be silently dropped (WRMSR 0x79 has no effect).	
		2	EN_SMM_BYPASS If set, then writes to IA32_MCU_CONTROL are allowed during SMM regardless of the LOCK bit. This enables BIOS to Opt-In to the SMM Bypass functionality.	
		63:3	Reserved.	
14CEH	5326	IA32_LBR_CTL	Last Branch Record Enabling and Configuration Register (R/W)	
		0	LBREn When set, enables LBR recording.	Reset Value: 0
		1	OS When set, allows LBR recording when CPL == 0.	Reset Value: 0
		2	USR When set, allows LBR recording when CPL != 0.	Reset Value: 0
		3	CALL_STACK When set, records branches in call-stack mode. See Section 19.1.2.4.	Reset Value: 0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		15:4	Reserved	Reset Value: 0
		16	COND When set, records taken conditional branches. See Section 19.1.2.3.	
		17	NEAR_REL_JMP When set, records near relative JMPs. See Section 19.1.2.3.	
		18	NEAR_IND_JMP When set, records near indirect JMPs. See Section 19.1.2.3.	
		19	NEAR_REL_CALL When set, records near relative CALLs. See Section 19.1.2.3.	
		20	NEAR_IND_CALL When set, records near indirect CALLs. See Section 19.1.2.3.	
		21	NEAR_RET When set, records near RETs. See Section 19.1.2.3.	
		22	OTHER_BRANCH When set, records other branches. See Section 19.1.2.3.	
		63:23	Reserved	
14CFH	5327	IA32_LBR_DEPTH	Last Branch Record Maximum Stack Depth Register (R/W)	
		N:0	DEPTH The number of LBRs to be used for recording. Supported values are indicated by the bitmap in CPUID.(EAX=01CH,ECX=0):EAX[7:0]. The reset value will match the maximum supported by the CPU. Writes of unsupported values will #GP fault.	Reset Value: Varies
		63:N+1	Reserved	Reset Value: 0
1500H - 151FH	5376 - 5407	IA32_LBR_x_FROM_IP	Last Branch Record entry X source IP register (R/W). An attempt to read or write IA32_LBR_x_FROM_IP such that $x \geq$ IA32_LBR_DEPTH.DEPTH will #GP.	
		63:0	FROM_IP The source IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
1600H - 161FH	5632 - 5663	IA32_LBR_x_TO_IP	Last Branch Record Entry X Destination IP Register (R/W) An attempt to read or write IA32_LBR_x_TO_IP such that $x \geq$ IA32_LBR_DEPTH.DEPTH will #GP.	
		63:0	TO_IP The destination IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
17D0H	6096	IA32_HW_FEEDBACK_PTR	Hardware Feedback Interface Pointer	If CPUID.06H:EAX.[19] = 1
		0	Valid (R/W) When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.	
		11:1	Reserved	
		(MAXPHYADDR-1):12	ADDR (R/W) Physical address of the page frame of the first page of the hardware feedback interface structure.	
		63:MAXPHYADDR	Reserved	
17D1H	6097	IA32_HW_FEEDBACK_CONFIG	Hardware Feedback Interface Configuration	If CPUID.06H:EAX.[19] = 1
		0	Enable (R/W) When set to 1, enables the hardware feedback interface.	
		63:1	Reserved	
17D2H	6098	IA32_THREAD_FEEDBACK_CHAR	Thread Feedback Characteristics (R/O)	If CPUID.06H:EAX.[23] = 1
		7:0	Application Class ID, pointing into the Intel Thread Director structure.	
		62:8	Reserved	
		63	Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.	
17D4H	6100	IA32_HW_FEEDBACK_THREAD_CONFIG	Hardware Feedback Thread Configuration (R/W)	
		0	Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).	
		63:1	Reserved	
17DAH	6106	IA32_HRESET_ENABLE	History Reset Enable (R/W)	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Enable reset of the Intel Thread Director history.	
		31:1	Reserved for other capabilities that can be reset by the HRESET instruction.	
		63:32	Reserved	
1B01H	6913	IA32_UARCH_MISC_CTL	IA32_UARCH_MISC_CTL	If IA32_ARCH_CAPABILITIES[12]=1
		0	Data Operand Independent Timing Mode (DOITM)	If IA32_ARCH_CAPABILITIES[12]=1
		63:1	Reserved	
4000_0000H - 4000_00FFH		Reserved MSR Address Space	All existing and future processors will not implement MSRs in this range.	
C000_0080H		IA32_EFER	Extended Feature Enables	If (CPUID.80000001H:EDX.[20] CPUID.80000001H:EDX.[29])
		0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.	
		7:1	Reserved	
		8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.	
		9	Reserved	
		10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.	
		11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W)	
	63:12	Reserved		
C000_0081H		IA32_STAR	System Call Target Address (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0082H		IA32_LSTAR	IA-32e Mode System Call Target Address (R/W) Target RIP for the called procedure when SYSCALL is executed in 64-bit mode.	If CPUID.80000001:EDX.[29] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C000_0083H		IA32_CSTAR	IA-32e Mode System Call Target Address (R/W) Not used, as the SYSCALL instruction is not recognized in compatibility mode.	If CPUID.80000001:EDX.[29] = 1
C000_0084H		IA32_FMASK	System Call Flag Mask (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0100H		IA32_FS_BASE	Map of BASE Address of FS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0101H		IA32_GS_BASE	Map of BASE Address of GS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0102H		IA32_KERNEL_GS_BASE	Swap Target of BASE Address of GS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0103H		IA32_TSC_AUX	Auxiliary TSC (R/W)	If CPUID.80000001H: EDX[27] = 1 or CPUID.(EAX=7,ECX=0):ECX[bit 22] = 1
		31:0	AUX: Auxiliary signature of TSC.	
		63:32	Reserved	

NOTES:

1. Some older processors may have supported this MSR as model-specific and do not enumerate it with CPUID.
2. In processors based on Intel NetBurst® microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.
3. The *_ADDR MSRs may or may not be present; this depends on flag settings in IA32_MCI_STATUS. See Section 16.3.2.3 and Section 16.3.2.4 for more information.
4. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].
5. Further details on Key Locker and usage of this MSR can be found here:

<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 2-3 lists model-specific registers (MSRs) for the Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table 2-3. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_0FH, see Table 2-1.

MSRs listed in Table 2-2 and Table 2-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. Processors based on the Enhanced Intel Core microarchitecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_17H.

The column “Shared/Unique” applies to multi-core processors based on Intel Core microarchitecture. “Unique” means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. “Shared” means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Unique	See Section 2.23, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Unique	See Section 2.23, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 9.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 18.17, “Time-Stamp Counter,” and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		7:0		Reserved
		12:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		49:13		Reserved
		52:50		See Table 2-2.
		63:53		Reserved
1BH	27	IA32_APIC_BASE	Unique	See Section 11.4.4, “Local APIC Status and Location,” and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processors implement R/W.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processors implement R/W.
		4		Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processors implement R/W.
		5		Reserved
		6		Reserved

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		7		BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processors implement R/W.
		8		Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled
		9		Execute BIST (R/O) 1 = Enabled; 0 = Disabled
		10		MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled
		11		Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present
		12		BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled
		13		Reserved
		14		1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		APIC Cluster ID (R/O)
		18		N/2 Non-Integer Bus Ratio (R/O) 0 = Integer ratio; 1 = Non-integer ratio
		19		Reserved
		21:20		Symmetric Arbitration ID (R/O)
		26:22		Integer Bus Frequency Ratio (R/O)
3AH	58	MSR_FEATURE_CONTROL	Unique	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		3	Unique	SMRR Enable (R/WL) When this bit is set and the lock bit is set, this makes the SMRR_PHYS_BASE and SMRR_PHYS_MASK registers read visible and writeable while in SMM.
40H	64	MSR_LASTBRANCH_0_FROM_IP	Unique	Last Branch Record 0 From IP (R/W) One of four pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.5.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	Last Branch Record 0 To IP (R/W) One of four pairs of last branch record registers on the last branch record stack. This To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
A0H	160	MSR_SMRR_PHYSBASE	Unique	System Management Mode Base Address register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		11:0		Reserved
		31:12		PhysBase: SMRR physical Base Address.
		63:32		Reserved
A1H	161	MSR_SMRR_PHYSMASK	Unique	System Management Mode Physical Address Mask register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		10:0		Reserved
		11		Valid: Physical address base and range mask are valid.
		31:12		PhysMask: SMRR physical address range mask.
		63:32		Reserved
C1H	193	IA32_PMC0	Unique	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (R/O) This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		2:0		<ul style="list-style-type: none"> ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333)
				<p>133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.</p> <p>166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.</p> <p>266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B.</p> <p>333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.</p>
		63:3		Reserved
CDH	205	MSR_FSB_FREQ	Shared	<p>Scaleable Bus Speed (R/O)</p> <p>This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture.</p>
		2:0		<ul style="list-style-type: none"> ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600)
				<p>133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.</p> <p>166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.</p> <p>266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B.</p> <p>333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.</p>
		63:3		Reserved
E7H	231	IA32_MPERF	Unique	<p>Maximum Performance Frequency Clock Count (R/W)</p> <p>See Table 2-2.</p>
E8H	232	IA32_APERF	Unique	<p>Actual Performance Frequency Clock Count (R/W)</p> <p>See Table 2-2.</p>
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
		11	Unique	SMRR Capability Using MSR OAOH and OA1H (R)
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Current performance status. See Section 15.1.1, "Software Interface For Initiating Performance State Transitions."
		15:0		Current Performance State Value
		30:16		Reserved
		31		XE Operation (R/O). If set, XE operation is enabled. Default is cleared.
		39:32		Reserved
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		45		Reserved
		46		Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.
		63:47		Reserved
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2.
19DH	413	MSR_THERM2_CTL	Unique	Thermal Monitor 2 Control
		15:0		Reserved
		16		TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:16		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0		Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.
		8		Reserved
		9		Hardware Prefetcher Disable (R/W) When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Shared	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		13	Shared	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19	Shared	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	Enhanced Intel SpeedStep Technology Select Lock (R/W/O) When set, this bit causes the following bits to become read-only: <ul style="list-style-type: none"> Enhanced Intel SpeedStep Technology Select Lock (this bit). Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved
		22	Shared	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Shared	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Unique	XD Bit Disable (R/W) See Table 2-2.
		36:35		Reserved
		37	Unique	DCU Prefetcher Disable (R/W) When set to 1, the DCU L1 data cache prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The DCU prefetcher is an L1 data cache prefetcher. When the DCU prefetcher detects multiple loads from the same line done within a time limit, the DCU prefetcher assumes the next line will be required. The next line is prefetched in to the L1 data cache from memory or L2.
		38	Shared	IDA Disable (R/W) When set to 1 on processors that support IDA, the Intel Dynamic Acceleration feature (IDA) is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of IDA is enabled. Note: The power-on default value is used by BIOS to detect hardware support of IDA. If the power-on default value is 1, IDA is available in the processor. If the power-on default value is 0, IDA is not available.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		39	Unique	IP Prefetcher Disable (R/W) When set to 1, the IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.
		63:40		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Unique	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Unique	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Unique	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Unique	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Unique	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Unique	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Unique	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Unique	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Unique	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Unique	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Unique	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Unique	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Unique	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Unique	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Unique	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Unique	See Table 2-2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
250H	592	IA32_MTRR_FIX64K_00000	Unique	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Unique	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Unique	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Unique	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Unique	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Unique	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Unique	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Unique	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Unique	See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."
345H	837	MSR_PERF_CAPABILITIES	Unique	R/O. This applies to processors that do not support architectural perfmon version 2.
		5:0		LBR Format. See Table 2-2.
		6		PEBS Record Format
		7		PEBSSaveArchRegs. See Table 2-2.
		63:8		Reserved
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Unique	See Section 20.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
38FH	911	MSR_PERF_GLOBAL_CTRL	Unique	See Section 20.6.2.2, "Global Counter Control Facilities."

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Unique	See Section 20.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Unique	See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MCO_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
402H	1026	IA32_MCO_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
406H	1030	IA32_MC1_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC4_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC4_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	IA32_MC4_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
410H	1040	IA32_MC3_CTL		See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC3_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC3_MISC	Unique	Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.
414H	1044	IA32_MC5_CTL	Unique	Machine Check Error Reporting Register: Controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
415H	1045	IA32_MC5_STATUS	Unique	Machine Check Error Reporting Register: Contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
416H	1046	IA32_MC5_ADDR	Unique	Machine Check Error Reporting Register: Contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCi_STATUS register is set.
417H	1047	IA32_MC5_MISC	Unique	Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.
419H	1045	IA32_MC6_STATUS	Unique	Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 24.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
484H	1156	IA32_VMX_ENTRY_CTL5	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Unique	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Unique	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."
107CH	67532	MSR_EMON_L3_CTR_CTL0	Unique	GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
107CDH	67533	MSR_EMON_L3_CTR_CTL1	Unique	GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
107CEH	67534	MSR_EMON_L3_CTR_CTL2	Unique	GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
107CF H	67535	MSR_EMON_L3_CTR_CTL3	Unique	GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
107D0 H	67536	MSR_EMON_L3_CTR_CTL4	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
107D1 H	67537	MSR_EMON_L3_CTR_CTL5	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
107D2 H	67538	MSR_EMON_L3_CTR_CTL6	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
107D3 H	67539	MSR_EMON_L3_CTR_CTL7	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
107D8 H	67544	MSR_EMON_L3_GL_CTL	Unique	L3/FSB Common Control Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.
C000_0080H		IA32_EFER	Unique	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

2.3 MSRS IN THE 45 NM AND 32 NM INTEL ATOM® PROCESSOR FAMILY

Table 2-4 lists model-specific registers (MSRs) for 45 nm and 32 nm Intel Atom processors, architectural MSR addresses are also included in Table 2-4. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1CH, 06_26H, 06_27H, 06_35H, or 06_36H; see Table 2-1.

The column “Shared/Unique” applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. “Unique” means each logical processor has a separate MSR, or a bit field in an MSR

governs only a logical processor. “Shared” means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 2.23, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Shared	See Section 2.23, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 9.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 18.17, “Time-Stamp Counter,” and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		7:0		Reserved
		12:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		63:13		Reserved
1BH	27	IA32_APIC_BASE	Unique	See Section 11.4.4, “Local APIC Status and Location,” and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		3		AERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		4		BERR# Enable for initiator bus requests (R/W) 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved
		6		Reserved
		7		BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		9		Execute BIST (R/O) 1 = Enabled; 0 = Disabled
		10		AERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved
		12		BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0.
		13		Reserved
		14		1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		APIC Cluster ID (R/O) Always 00B.
		19: 18		Reserved
		21: 20		Symmetric Arbitration ID (R/O) Always 00B.
		26:22		Integer Bus Frequency Ratio (R/O)
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in Intel 64Processor (R/W) See Table 2-2.
40H	64	MSR_LASTBRANCH_0_FROM_IP	Unique	Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.5.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Unique	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Unique	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Unique	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
47H	71	MSR_LASTBRANCH_7_FROM_IP	Unique	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Unique	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Unique	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Unique	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_LASTBRANCH_7_TO_IP	Unique	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Shared	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Unique	Performance counter register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Atom microarchitecture.
		2:0		<ul style="list-style-type: none"> ▪ 111B: 083 MHz (FSB 333) ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667)
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
		63:3		Reserved

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Shared	Memory Type Range Register (R) See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (R/O) 1 = Indicates the L2 is hardware-enabled. 0 = Indicates the L2 is hardware-disabled.
		7:1		Reserved
		8		L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set, the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved
		23		L2 Not Present (R/O) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Performance Status
		15:0		Current Performance State Value
		39:16		Reserved
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		63:45		Reserved
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2.
19DH	413	MSR_THERM2_CTL	Shared	Thermal Monitor 2 Control
		15:0		Reserved
		16		TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:17		Reserved
1A0H	416	IA32_MISC_ENABLE	Unique	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0		Fast-Strings Enable See Table 2-2.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		2:1		Reserved
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.
		8		Reserved
		9		Reserved
		10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Shared	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		13	Shared	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is cleared (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19		Reserved

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: <ul style="list-style-type: none"> ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit). ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved
		22	Unique	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Shared	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Unique	XD Bit Disable (R/W) See Table 2-2.
		63:35		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Shared	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Shared	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Shared	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Shared	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Shared	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Shared	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Shared	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Shared	See Table 2-2.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
208H	520	IA32_MTRR_PHYSBASE4	Shared	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Shared	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Shared	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Shared	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Shared	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Shared	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Shared	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Shared	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Shared	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Shared	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Shared	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Shared	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Shared	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Shared	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Shared	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Unique	See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0 (R/W)

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
400H	1024	IA32_MCO_CTL	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Shared	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDRVL flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRVL flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Shared	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRVL flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR	Shared	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRVL flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
481H	1153	IA32_VMX_PINBASED_CTL5	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL5	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL5	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL5	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Unique	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Unique	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."
C000_0080H		IA32_EFER	Unique	Extended Feature Enables See Table 2-2.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
C000_0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

Table 2-5 lists model-specific registers (MSRs) that are specific to Intel Atom® processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_27H.

Table 2-5. MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_27H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3F8H	1016	MSR_PKG_C2_RESIDENCY	Package	Package C2 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0	Package	Package C2 Residency Counter (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.
3F9H	1017	MSR_PKG_C4_RESIDENCY	Package	Package C4 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0	Package	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	Package C6 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0	Package	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.

2.4 MSRS IN INTEL PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 2-6 lists model-specific registers (MSRs) common to Intel processors based on the Silvermont microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_37H, 06_4AH, 06_4DH, 06_5AH, or 06_5DH; see Table 2-1. The MSRs listed in Table 2-6 are also common to processors based on the Airmont microarchitecture and newer microarchitectures for next generation Intel Atom processors.

Table 2-7 lists MSRs common to processors based on the Silvermont and Airmont microarchitectures, but not newer microarchitectures.

Table 2-8, Table 2-9, and Table 2-10 lists MSRs that are model-specific across processors based on the Silvermont microarchitecture.

In the Silvermont microarchitecture, the scope column indicates the following: "Core" means each processor core has a separate MSR, or a bit field not shared with another processor core. "Module" means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Silvermont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID leaf 04H enumeration can be used to figure out which processors are in the same module. "Package" means all processor cores in the physical package share the same MSR or bit interface.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Core	See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Core	See Section 18.17, "Time-Stamp Counter," and Table 2-2.
1BH	27	IA32_APIC_BASE	Core	See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Module	Processor Hard Power-On Configuration (R/W) Writes ignored.
		63:0		Reserved
34H	52	MSR_SMI_COUNT	Core	SMI Counter (R/O)
		31:0		SMI Count (R/O) Running count of SMI events since last RESET.
		63:32		Reserved
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Core	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Core	Performance counter register See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C2H	194	IA32_PMC1	Core	Performance Counter Register See Table 2-2.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 100b - C4 is the max C-State to include 110b - C6 is the max C-State to include 111b - C7 is the max C-State to include
		63:19		Reserved
E7H	231	IA32_MPERF	Core	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Core	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Core	Memory Type Range Register (R) See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction sets availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note: AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
174H	372	IA32_SYSENTER_CS	Core	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Core	See Table 2-2.
179H	377	IA32_MCG_CAP	Core	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Core	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Core	See Table 2-2.
		7:0		Event Select
		15:8		UMask
		16		USR
		17		OS
		18		Edge
		19		PC
		20		INT
		21		Reserved
		22		EN
		23		INV
		31:24		CMASK
		63:32		Reserved
		187H	391	IA32_PERFEVTSEL1
198H	408	IA32_PERF_STATUS	Module	See Table 2-2.
199H	409	IA32_PERF_CTL	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_MODULATION	Core	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R) The default thermal throttling or PROCHOT# activation temperature in degrees C. The effective temperature for thermal throttling or PROCHOT# activation is "Temperature Target" + "Target Offset".
		29:24		Target Offset (R/W) Specifies an offset in degrees C to adjust the throttling and PROCHOT# activation temperature from the default target specified in TEMPERATURE_TARGET (bits 23:16).
		63:30		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Module	Offcore Response Event Select Register (R/W)
1A7H	423	MSR_OFFCORE_RSP_1	Module	Offcore Response Event Select Register (R/W)
1B0H	432	IA32_ENERGY_PERF_BIAS	Core	See Table 2-2.
1D9H	473	IA32_DEBUGCTL	Core	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Core	Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Core	Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Core	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Core	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Core	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Core	See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Core	Fixed-Function-Counter Control Register (R/W) See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
38FH	911	IA32_PERF_GLOBAL_CTRL	Core	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Counts at the TSC Frequency.
400H	1024	IA32_MCO_CTL	Module	See Section 16.3.2.1, "IA32_MCI_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Module	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Module	See Section 16.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Module	See Section 16.3.2.1, "IA32_MCI_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Module	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Module	See Section 16.3.2.1, "IA32_MCI_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Module	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Module	See Section 16.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	Core	See Section 16.3.2.1, "IA32_MCI_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Core	See Section 16.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	Core	See Section 16.3.2.1, "IA32_MCI_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
412H	1042	IA32_MC4_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
414H	1044	IA32_MC5_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	IA32_MC5_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Core	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Core	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Core	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Core	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Core	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Core	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Core	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_FIXED1	Core	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Core	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Core	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Core	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLSS2	Core	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLSS	Core	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLSS	Core	Capability Reporting Register of Primary Processor-based VM-Execution Flex Controls (R/O) See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLSS	Core	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLSS	Core	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2
491H	1169	IA32_VMX_FMFUNC	Core	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2
4C1H	1217	IA32_A_PMC0	Core	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Core	DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C1 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C1 states. Counts at the TSC frequency.
6E0H	1760	IA32_TSC_DEADLINE	Core	TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.
C000_0080H		IA32_EFER	Core	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Core	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Core	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Core	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Core	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Core	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Core	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Core	AUXILIARY TSC Signature (R/W) See Table 2-2.

Table 2-7 lists model-specific registers (MSRs) that are common to Intel Atom® processors based on the Silvermont and Airmont microarchitectures but not newer microarchitectures.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	Model Specific Platform ID (R)
		7:0		Reserved
		13:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		49:13		Reserved
		52:50		See Table 2-2.
		63:33		Reserved

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Reserved
		2		Enable VMX outside SMX operation (R/WL)
40H	64	MSR_LASTBRANCH_0_FROM_IP	Core	Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.5 and record format in Section 18.4.8.1.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Core	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Core	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Core	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Core	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Core	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Core	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_LASTBRANCH_7_FROM_IP	Core	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Core	Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Core	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Core	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Core	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Core	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Core	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Core	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
67H	103	MSR_LASTBRANCH_7_TO_IP	Core	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information: Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.
		63:16		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 100b: C4 110b: C6 111b: C7 (Silvermont only).
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		63:16		Reserved
11EH	281	MSR_BBL_CR_CTL3	Module	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
		7:1		Reserved

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		8		L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved
		23		L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.
		63:24		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Core	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Module	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.
		6:4		Reserved
		7	Core	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Core	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Core	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Module	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Core	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
		22	Core	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Module	xTPR Message Disable (R/W) See Table 2-2.
33:24		Reserved		
34	Core	XD Bit Disable (R/W) See Table 2-2.		

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		37:35		Reserved
		38	Module	<p>Turbo Mode Disable (R/W)</p> <p>When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H: EAX[1]=0).</p> <p>When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled.</p> <p>Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.</p>
		63:39		Reserved
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
		63:9		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Core	<p>Last Branch Record Stack TOS (R/W)</p> <p>Contains an index (bits 0-2) that points to the MSR containing the most recent branch record.</p> <p>See MSR_LASTBRANCH_0_FROM_IP.</p>
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Core	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS for precise event on IA32_PMC0 (R/W)
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Counts at the TSC Frequency.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.

2.4.1 MSRs with Model-Specific Behavior in the Silvermont Microarchitecture

Table 2-8 lists MSRs that are specific to the Intel Atom[®] processor E3000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_37H) and Intel Atom processors (CPUID Signature DisplayFamily_DisplayModel value of 06_4AH, 06_5AH, or 06_5DH).

Table 2-8. Specific MSRs Supported by Intel Atom[®] Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Silvermont microarchitecture.
		2:0		<ul style="list-style-type: none"> ▪ 100B: 080.0 MHz ▪ 000B: 083.3 MHz ▪ 001B: 100.0 MHz ▪ 010B: 133.3 MHz ▪ 011B: 116.7 MHz
		63:3		Reserved
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."
		3:0		Power Units Power related information (in milliWatts) is based on the multiplier, 2 ^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units Energy related information (in microJoules) is based on the multiplier, 2 ^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.
		15:13		Reserved

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		19:16		Time Unit The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W)
		14:0		Package Power Limit #1 (R/W) See Section 15.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1 (R/W) See Section 15.10.3, "Package RAPL Domain."
		16		Package Clamping Limitation #1 (R/W) See Section 15.10.3, "Package RAPL Domain."
		23:17		Time Window for Power Limit #1 (R/W) In unit of second. If 0 is specified in bits [23:17], defaults to 1 second window.
		63:24		Reserved
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.

Table 2-9 lists model-specific registers (MSRs) that are specific to the Intel Atom® processor E3000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_37H).

Table 2-9. Specific MSRs Supported by the Intel Atom® Processor E3000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
668H	1640	MSR_CC6_DEMOTION_POLICY_CONFIG	Package	Core C6 Demotion Policy Config MSR
		63:0		Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.
669H	1641	MSR_MC6_DEMOTION_POLICY_CONFIG	Package	Module C6 Demotion Policy Config MSR
		63:0		Controls module (i.e., two cores sharing the second-level cache) C6 demotion policy. Writing a value of 0 disables module level HW demotion policy.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-9. Specific MSRs Supported by the Intel Atom® Processor E3000 Series (Contd.)with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.

Table 2-10 lists model-specific registers (MSRs) that are specific to Intel Atom® processor C2000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_4DH).

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		Reserved
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		63:3		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode (R/W)
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series (Contd.)with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		3:0		Power Units Power related information (in milliwatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliwatts increment.
		7:4		Reserved
		12:8		Energy Status Units. Energy related information (in microJoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.
		15:13		Reserved
		19:16		Time Unit The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."
66EH	1646	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameter (R/O)
		14:0		Thermal Spec Power (R/O) The unsigned integer value is the equivalent of the thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
		63:15		Reserved

2.4.2 MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Intel Atom processor X7-Z8000 and X5-Z8000 series are based on the Airmont microarchitecture. These processors support MSRs listed in Table 2-6, Table 2-7, Table 2-8, and Table 2-11. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_4CH; see Table 2-1.

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Airmont microarchitecture.

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		3:0		<ul style="list-style-type: none"> ▪ 0000B: 083.3 MHz ▪ 0001B: 100.0 MHz ▪ 0010B: 133.3 MHz ▪ 0011B: 116.7 MHz ▪ 0100B: 080.0 MHz ▪ 0101B: 093.3 MHz ▪ 0110B: 090.0 MHz ▪ 0111B: 088.9 MHz ▪ 1000B: 087.5 MHz
		63:5		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: No limit 001b: C1 010b: C2 110b: C6 111b: C7
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		63:16		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		18:16		C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include. 001b - Deep Power Down Technology is the max C-State. 010b - C7 is the max C-State to include.
		63:19		Reserved
638H	1592	MSR_PPO_POWER_LIMIT	Package	PPO RAPL Power Limit Control (R/W)
		14:0		PPO Power Limit #1 (R/W) See Section 15.10.4, "PPO/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1 (R/W) See Section 15.10.4, "PPO/PP1 RAPL Domains."
		16		Reserved
		23:17		Time Window for Power Limit #1 (R/W) Specifies the time duration over which the average power must remain below PPO_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.
		63:24		Reserved

2.5 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Intel Atom processors based on the Goldmont microarchitecture support MSRs listed in Table 2-6 and Table 2-12. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH; see Table 2-1.

In the Goldmont microarchitecture, the scope column indicates the following: "Core" means each processor core has a separate MSR, or a bit field not shared with another processor core. "Module" means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset

is model specific and may differ between different processors. For all processors based on Goldmont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID leaf 04H enumeration can be used to figure out which processors are in the same module. "Package" means all processor cores in the physical package share the same MSR or bit interface.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	Model Specific Platform ID (R)
		49:0		Reserved
		52:50		See Table 2-2.
		63:33		Reserved
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX inside SMX operation (R/WL)
		2		Enable VMX outside SMX operation (R/WL)
		14:8		SENTER local functions enables (R/WL)
		15		SENTER global functions enable (R/WL)
		18		SGX global functions enable (R/WL)
		63:19		Reserved
3BH	59	IA32_TSC_ADJUST	Core	Per-Core TSC ADJUST (R/W) See Table 2-2.
C3H	195	IA32_PMC2	Core	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Core	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		30	Package	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		3:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: No limit 0001b: C1 0010b: C3 0011b: C6 0100b: C7 0101b: C7S 0110b: C8 0111b: C9 1000b: C10
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:16		Reserved
17DH	381	MSR_SMM_MCA_CAP	Core	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved
188H	392	IA32_PERFEVTSEL2	Core	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Core	See Table 2-2.
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Core	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Package	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.
		6:4		Reserved
		7	Core	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Core	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Core	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Core	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
22	Core	Limit CPUID Maxval (R/W) See Table 2-2.		

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		23	Package	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Core	XD Bit Disable (R/W) See Table 2-2.
		37:35		Reserved
		38	Package	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.
		63:39		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		Reserved
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		63:3		Reserved
1AAH	426	MSR_MISC_PWR_MGMT	Package	Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .
		0		EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.
		21:1		Reserved
		22		Thermal Interrupt Coordination Enable (R/W) If set, then thermal interrupt on one core is routed to all cores.
		63:23		Reserved

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode by Core Groups (R/W) Specifies Maximum Ratio Limit for each Core Group. Max ratio for groups with more cores must decrease monotonically. For groups with less than 4 cores, the max ratio must be 32 or less. For groups with 4-5 cores, the max ratio must be 22 or less. For groups with more than 5 cores, the max ratio must be 16 or less.
		7:0	Package	Maximum Ratio Limit for Active Cores in Group 0 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 0 threshold.
		15:8	Package	Maximum Ratio Limit for Active Cores in Group 1 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 1 threshold, and greater than the Group 0 threshold.
		23:16	Package	Maximum Ratio Limit for Active Cores in Group 2 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 2 threshold, and greater than the Group 1 threshold.
		31:24	Package	Maximum Ratio Limit for Active Cores in Group 3 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 3 threshold, and greater than the Group 2 threshold.
		39:32	Package	Maximum Ratio Limit for Active Cores in Group 4 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 4 threshold, and greater than the Group 3 threshold.
		47:40	Package	Maximum Ratio Limit for Active Cores in Group 5 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 5 threshold, and greater than the Group 4 threshold.
		55:48	Package	Maximum Ratio Limit for Active Cores in Group 6 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 6 threshold, and greater than the Group 5 threshold.
		63:56	Package	Maximum Ratio Limit for Active Cores in Group 7 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 7 threshold, and greater than the Group 6 threshold.
1AEH	430	MSR_TURBO_GROUP_CORECNT	Package	Group Size of Active Cores for Turbo Mode Operation (R/W) Writes of 0 threshold is ignored.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7:0	Package	Group 0 Core Count Threshold Maximum number of active cores to operate under the Group 0 Max Turbo Ratio limit.
		15:8	Package	Group 1 Core Count Threshold Maximum number of active cores to operate under the Group 1 Max Turbo Ratio limit. Must be greater than the Group 0 Core Count.
		23:16	Package	Group 2 Core Count Threshold Maximum number of active cores to operate under the Group 2 Max Turbo Ratio limit. Must be greater than the Group 1 Core Count.
		31:24	Package	Group 3 Core Count Threshold Maximum number of active cores to operate under the Group 3 Max Turbo Ratio limit. Must be greater than the Group 2 Core Count.
		39:32	Package	Group 4 Core Count Threshold Maximum number of active cores to operate under the Group 4 Max Turbo Ratio limit. Must be greater than the Group 3 Core Count.
		47:40	Package	Group 5 Core Count Threshold Maximum number of active cores to operate under the Group 5 Max Turbo Ratio limit. Must be greater than the Group 4 Core Count.
		55:48	Package	Group 6 Core Count Threshold Maximum number of active cores to operate under the Group 6 Max Turbo Ratio limit. Must be greater than the Group 5 Core Count.
		63:56	Package	Group 7 Core Count Threshold Maximum number of active cores to operate under the Group 7 Max Turbo Ratio limit. Must be greater than the Group 6 Core Count, and not less than the total number of processor cores in the package. E.g., specify 255.
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		8		FAR_BRANCH
		9		EN_CALL_STACK
		63:10		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Core	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See http://biosbits.org .
		0		Reserved
		1	Package	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved
210H	528	IA32_MTRR_PHYSBASE8	Core	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Core	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Core	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Core	See Table 2-2.
280H	640	IA32_MC0_CTL2	Module	See Table 2-2.
281H	641	IA32_MC1_CTL2	Module	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Module	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
300H	768	MSR_SGXOWNEREPOCH0	Package	Lower 64 Bit CR_SGXOWNEREPOCH (w) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	769	MSR_SGXOWNEREPOCH1	Package	Upper 64 Bit CR_SGXOWNEREPOCH (w) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 20.2.4, "Architectural Performance Monitoring Version 4."
		0		Ovf_PMCO

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1		Ovf_PMC1
		2		Ovf_PMC2
		3		Ovf_PMC3
		31:4		Reserved
		32		Ovf_FixedCtr0
		33		Ovf_FixedCtr1
		34		Ovf_FixedCtr2
		54:35		Reserved
		55		Trace_ToPA_PMI
		57:56		Reserved
		58		LBR_Frz.
		59		CTR_Frz.
		60		ASCI
		61		Ovf_Uncore
		62		Ovf_BufDSSAVE
63		CondChgd		
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Core	See Table 2-2. See Section 20.2.4, "Architectural Performance Monitoring Version 4."
		0		Set 1 to clear Ovf_PMC0.
		1		Set 1 to clear Ovf_PMC1.
		2		Set 1 to clear Ovf_PMC2.
		3		Set 1 to clear Ovf_PMC3.
		31:4		Reserved
		32		Set 1 to clear Ovf_FixedCtr0.
		33		Set 1 to clear Ovf_FixedCtr1.
		34		Set 1 to clear Ovf_FixedCtr2.
		54:35		Reserved
		55		Set 1 to clear Trace_ToPA_PMI.
		57:56		Reserved
		58		Set 1 to clear LBR_Frz.
		59		Set 1 to clear CTR_Frz.
		60		Set 1 to clear ASCI.
61		Set 1 to clear Ovf_Uncore.		
62		Set 1 to clear Ovf_BufDSSAVE.		
63		Set 1 to clear CondChgd.		
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Core	See Table 2-2. See Section 20.2.4, "Architectural Performance Monitoring Version 4."

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		Set 1 to cause Ovf_PMC0 = 1.
		1		Set 1 to cause Ovf_PMC1 = 1.
		2		Set 1 to cause Ovf_PMC2 = 1.
		3		Set 1 to cause Ovf_PMC3 = 1.
		31:4		Reserved
		32		Set 1 to cause Ovf_FixedCtr0 = 1.
		33		Set 1 to cause Ovf_FixedCtr1 = 1.
		34		Set 1 to cause Ovf_FixedCtr2 = 1.
		54:35		Reserved
		55		Set 1 to cause Trace_ToPA_PMI = 1.
		57:56		Reserved
		58		Set 1 to cause LBR_Frz = 1.
		59		Set 1 to cause CTR_Frz = 1.
		60		Set 1 to cause ASCII = 1.
		61		Set 1 to cause Ovf_Uncore.
		62		Set 1 to cause Ovf_BufDSSAVE.
		63		Reserved
392H	914	IA32_PERF_GLOBAL_INUSE	Core	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
406H	1030	IA32_MC1_ADDR	Module	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
418H	1048	IA32_MC6_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
41AH	1050	IA32_MC6_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
4C3H	1219	IA32_A_PMC2	Core	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Core	See Table 2-2.
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.
		0		Lock (SMM-RW0) When set to '1' locks this register from further changes.
		1		Reserved
		2		SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is OFFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
500H	1280	IA32_SGX_SVN_STATUS	Core	Status and SVN Threshold of SGX Support for ACM (R/O)
		0		Lock See Section 39.1.1.3, "Interactions with Authenticated Code Modules (ACMs)."
		15:1		Reserved
		23:16		SGX_SVN_SINIT See Section 39.1.1.3, "Interactions with Authenticated Code Modules (ACMs)."
		63:24		Reserved
560H	1376	IA32_RTIT_OUTPUT_BASE	Core	Trace Output Base Register (R/W) See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Core	Trace Output Mask Pointers Register (R/W) See Table 2-2.
570H	1392	IA32_RTIT_CTL	Core	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		6:4		Reserved, must be zero.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7		CR3Filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.
		9		MTCEn
		10		TSCEn
		11		DisRETC
		12		Reserved, must be zero.
		13		BranchEn
		17:14		MTCFreq
		18		Reserved, must be zero.
		22:19		CycThresh
		23		Reserved, must be zero.
		27:24		PSBFreq
		31:28		Reserved, must be zero.
		35:32		ADDRO_CFG
		39:36		ADDR1_CFG
		63:40		Reserved, must be zero.
571H	1393	IA32_RTIT_STATUS	Core	Tracing Status Register (R/W)
		0		FilterEn Writes ignored.
		1		ContexEn Writes ignored.
		2		TriggerEn Writes ignored.
		3		Reserved
		4		Error (R/W)
		5		Stopped
		31:6		Reserved, must be zero.
		48:32		PacketByteCnt
		63:49		Reserved, must be zero.
572H	1394	IA32_RTIT_CR3_MATCH	Core	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match.
580H	1408	IA32_RTIT_ADDRO_A	Core	Region 0 Start Address (R/W)
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDRO_B	Core	Region 0 End Address (R/W)
		63:0		See Table 2-2.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
582H	1410	IA32_RTIT_ADDR1_A	Core	Region 1 Start Address (R/W)
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Core	Region 1 End Address (R/W)
		63:0		See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."
		3:0		Power Units Power related information (in Watts) is in unit of $1W/2^{PU}$; where PU is an unsigned integer represented by bits 3:0. Default value is 1000b, indicating power unit is in 3.9 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units Energy related information (in Joules) is in unit of $1Joule/2^{ESU}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 01110b, indicating energy unit is in 61 microJoules.
		15:13		Reserved
		19:16		Time Unit Time related information (in seconds) is in unit of $1S/2^{TU}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating power unit is in 0.977 millisecond.
		63:20		Reserved
60AH	1546	MSR_PKG_C3_IRT_L	Package	Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60BH	1547	MSR_PKG_C2_IRTL1	Package	Package C6/C7S Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7S state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7S state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
60CH	1548	MSR_PKG_C2_IRTL2	Package	Package C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 15.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W)
		14:0		Thermal Spec Power (R/W) See Section 15.10.3, "Package RAPL Domain."
		15		Reserved
		30:16		Minimum Power (R/W) See Section 15.10.3, "Package RAPL Domain."
		31		Reserved
		46:32		Maximum Power (R/W) See Section 15.10.3, "Package RAPL Domain."
		47		Reserved
		54:48		Maximum Time Window (R/W) Specified by $2^Y * (1.0 + Z/4.0) * \text{Time_Unit}$, where "Y" is the unsigned integer value represented by bits 52:48, "Z" is an unsigned integer represented by bits 54:53. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
63:55		Reserved		
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C10 Residency Counter (R/O) Value since last reset that the entire SOC is in an S0i3 state. Count at the same frequency as the TSC.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		7:0		MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.
		30:8		Reserved
		31		TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		2		Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		3		Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		8:4		Reserved
		9		Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
11		Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.		

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		12		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		13		Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		14		Maximum Efficiency Frequency Status (R0) When set, frequency is reduced below the maximum efficiency frequency.
		15		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24:20		Reserved
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		Maximum Efficiency Frequency Log When set, indicates that the Maximum Efficiency Frequency Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:31		Reserved
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Core	Last Branch Record 0 From IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction . See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.6 and record format in Section 18.4.8.1.
		0:47		From Linear Address (R/W)
		62:48		Signed extension of bits 47:0.
		63		Mispred
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Core	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Core	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Core	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Core	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Core	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Core	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Core	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Core	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Core	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Core	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Core	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Core	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Core	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Core	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Core	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Core	Last Branch Record 16 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Core	Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Core	Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Core	Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Core	Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Core	Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Core	Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Core	Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Core	Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Core	Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Core	Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Core	Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Core	Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Core	Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Core	Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Core	Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Core	Last Branch Record 0 To IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the Destination instruction and elapsed cycles from last LBR update. See Section 18.6.
		0:47		Target Linear Address (R/W)
		63:48		Elapsed cycles from last update to the LBR.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Core	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Core	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Core	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Core	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Core	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Core	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Core	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Core	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Core	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Core	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Core	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Core	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Core	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Core	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Core	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DOH	1744	MSR_LASTBRANCH_16_TO_IP	Core	Last Branch Record 16 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Core	Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Core	Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Core	Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Core	Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Core	Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Core	Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Core	Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Core	Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Core	Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Core	Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Core	Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Core	Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Core	Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Core	Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Core	Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Core	x2APIC ID register (R/O)
803H	2051	IA32_X2APIC_VERSION	Core	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Core	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Core	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Core	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Core	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Core	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Core	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Core	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Core	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Core	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Core	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Core	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Core	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Core	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Core	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Core	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Core	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Core	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Core	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Core	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Core	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Core	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Core	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Core	x2APIC Interrupt Request register bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Core	x2APIC Interrupt Request register bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Core	x2APIC Interrupt Request register bits [127:96] (R/O)

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
824H	2084	IA32_X2APIC_IRR4	Core	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Core	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Core	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Core	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Core	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Core	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Core	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Core	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Core	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Core	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Core	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Core	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Core	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Core	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Core	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Core	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Core	x2APIC Self IPI register (W/O)
C8FH	3215	IA32_PQR_ASSOC	Core	Resource Association Register (R/W)
		31:0		Reserved
		33:32		COS (R/W)
		63:34		Reserved
D10H	3344	IA32_L2_QOS_MASK_0	Module	L2 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.
		63:8		Reserved
D11H	3345	IA32_L2_QOS_MASK_1	Module	L2 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.
		63:8		Reserved
D12H	3346	IA32_L2_QOS_MASK_2	Module	L2 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:8		Reserved
D13H	3347	IA32_L2_QOS_MASK_3	Package	L2 Class Of Service Mask - COS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L2 ways for COS 3 enforcement.
		63:20		Reserved
D90H	3472	IA32_BNDCFGS	Core	See Table 2-2.
DA0H	3488	IA32_XSS	Core	See Table 2-2.

See Table 2-6, and Table 2-12 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH.

2.6 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Intel Atom processors based on the Goldmont Plus microarchitecture support MSRs listed in Table 2-6, Table 2-12, and Table 2-13. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH; see Table 2-1. For an MSR listed in Table 2-13 that also appears in the model-specific tables of prior generations, Table 2-13 supersedes prior generation tables.

In the Goldmont Plus microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont Plus microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID leaf 04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX inside SMX operation (R/WL)
		2		Enable VMX outside SMX operation (R/WL)
		14:8		SENTER local functions enables (R/WL)
		15		SENTER global functions enable (R/WL)
		17		SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Valid if CPUID.(EAX=07H, ECX=0H): ECX[30] = 1.

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		18		SGX global functions enable (R/WL)
		63:19		Reserved
8CH	140	IA32_SGXLEPUBKEYHASH0	Core	See Table 2-2.
8DH	141	IA32_SGXLEPUBKEYHASH1	Core	See Table 2-2.
8EH	142	IA32_SGXLEPUBKEYHASH2	Core	See Table 2-2.
8FH	143	IA32_SGXLEPUBKEYHASH3	Core	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	(R/W) See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0.
		1		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC1.
		2		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC2.
		3		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC3.
		31:4		Reserved
		32		Enable PEBS trigger and recording for IA32_FIXED_CTR0.
		33		Enable PEBS trigger and recording for IA32_FIXED_CTR1.
		34		Enable PEBS trigger and recording for IA32_FIXED_CTR2.
		63:35		Reserved
570H	1392	IA32_RTIT_CTL	Core	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		4		PwrEvtEn
		5		FUPonPTW
		6		FabricEn
		7		CR3Filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.
9		MTCEn		

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		10		TSCEn
		11		DisRETC
		12		PTWEn
		13		BranchEn
		17:14		MTCFreq
		18		Reserved, must be zero.
		22:19		CycThresh
		23		Reserved, must be zero.
		27:24		PSBFreq
		31:28		Reserved, must be zero.
		35:32		ADDRO_CFG
		39:36		ADDR1_CFG
		63:40		Reserved, must be zero.
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Core	Last Branch Record 0 From IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."
681H - 69FH	1665 - 1695	MSR_LASTBRANCH_1_FROM_IP	Core	Last Branch Record <i>i</i> From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP; <i>i</i> = 1-31.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Core	Last Branch Record 0 To IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The To_IP part of the stack contains pointers to the Destination instruction. See also: <ul style="list-style-type: none"> ▪ Section 18.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."
6C1H - 6DFH	1729 - 1759	MSR_LASTBRANCH_1_TO_IP	Core	Last Branch Record <i>i</i> To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP; <i>i</i> = 1-31.
DC0H	3520	MSR_LASTBRANCH_INFO_0	Core	Last Branch Record 0 Additional Information (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. This part of the stack contains flag and elapsed cycle information. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.9.1, "LBR Stack."
DC1H	3521	MSR_LASTBRANCH_INFO_1	Core	Last Branch Record 1 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DC2H	3522	MSR_LASTBRANCH_INFO_2	Core	Last Branch Record 2 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC3H	3523	MSR_LASTBRANCH_INFO_3	Core	Last Branch Record 3 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC4H	3524	MSR_LASTBRANCH_INFO_4	Core	Last Branch Record 4 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC5H	3525	MSR_LASTBRANCH_INFO_5	Core	Last Branch Record 5 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC6H	3526	MSR_LASTBRANCH_INFO_6	Core	Last Branch Record 6 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC7H	3527	MSR_LASTBRANCH_INFO_7	Core	Last Branch Record 7 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC8H	3528	MSR_LASTBRANCH_INFO_8	Core	Last Branch Record 8 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC9H	3529	MSR_LASTBRANCH_INFO_9	Core	Last Branch Record 9 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCAH	3530	MSR_LASTBRANCH_INFO_10	Core	Last Branch Record 10 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCBH	3531	MSR_LASTBRANCH_INFO_11	Core	Last Branch Record 11 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCCH	3532	MSR_LASTBRANCH_INFO_12	Core	Last Branch Record 12 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCDH	3533	MSR_LASTBRANCH_INFO_13	Core	Last Branch Record 13 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCEH	3534	MSR_LASTBRANCH_INFO_14	Core	Last Branch Record 14 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCFH	3535	MSR_LASTBRANCH_INFO_15	Core	Last Branch Record 15 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD0H	3536	MSR_LASTBRANCH_INFO_16	Core	Last Branch Record 16 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD1H	3537	MSR_LASTBRANCH_INFO_17	Core	Last Branch Record 17 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD2H	3538	MSR_LASTBRANCH_INFO_18	Core	Last Branch Record 18 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD3H	3539	MSR_LASTBRANCH_INFO_19	Core	Last Branch Record 19 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD4H	3520	MSR_LASTBRANCH_INFO_20	Core	Last Branch Record 20 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD5H	3521	MSR_LASTBRANCH_INFO_21	Core	Last Branch Record 21 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD6H	3522	MSR_LASTBRANCH_INFO_22	Core	Last Branch Record 22 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD7H	3523	MSR_LASTBRANCH_INFO_23	Core	Last Branch Record 23 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD8H	3524	MSR_LASTBRANCH_INFO_24	Core	Last Branch Record 24 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD9H	3525	MSR_LASTBRANCH_INFO_25	Core	Last Branch Record 25 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDAH	3526	MSR_LASTBRANCH_INFO_26	Core	Last Branch Record 26 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDBH	3527	MSR_LASTBRANCH_INFO_27	Core	Last Branch Record 27 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDCH	3528	MSR_LASTBRANCH_INFO_28	Core	Last Branch Record 28 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDDH	3529	MSR_LASTBRANCH_INFO_29	Core	Last Branch Record 29 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDEH	3530	MSR_LASTBRANCH_INFO_30	Core	Last Branch Record 30 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDFH	3531	MSR_LASTBRANCH_INFO_31	Core	Last Branch Record 31 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
See Table 2-6, Table 2-12, and Table 2-13 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH.				

2.7 MSRS IN INTEL ATOM® PROCESSORS BASED ON TREMONT MICROARCHITECTURE

Processors based on the Tremont microarchitecture support MSRs listed in Table 2-6, Table 2-12, Table 2-13, and Table 2-14. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_86H, 06_96H, or 06_9CH; see Table 2-1. For an MSR listed in Table 2-14 that also appears in the model-specific tables of prior generations, Table 2-14 supersedes prior generation tables.

In the Tremont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Tremont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID leaf 04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		28:0		Reserved.
		29		SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 9.1.2.3, "Features to Disable Bus Locks."
		30		Reserved.
		31		Reserved.
CFH	207	IA32_CORE_CAPABILITIES	Core	IA32 Core Capabilities Register If CPUID.(EAX=07H, ECX=0):EDX[30] = 1.
		4:0		Reserved.
		5		SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).
		63:6		Reserved.
2A0H	672	MSR_PRMRR_BASE_0	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MEMTYPE: PRMRR BASE Memory Type.
		3		CONFIGURED: PRMRR BASE Configured.
		11:4		Reserved.
		51:12		BASE: PRMRR Base Address.
		63:52		Reserved.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	(R/W) See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."
		<i>n</i> :0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMCx. The maximum value <i>n</i> can be determined from CPUID.0AH:EAX[15:8].
		31: <i>n</i> +1		Reserved.
		32+ <i>m</i> :32		Enable PEBS trigger and recording for IA32_FIXED_CTRx. The maximum value <i>m</i> can be determined from CPUID.0AH:EDX[4:0].
		59:33+ <i>m</i>		Reserved.
		60		Pend a PerfMon Interrupt (PMI) after each PEBS event.
		62:61		Specifies PEBS output destination. Encodings: 00B: DS Save Area 01B: Intel PT trace output. Supported if IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] and CPUID.07H.0.EBX[25] are set. 10B: Reserved 11B: Reserved

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63		Reserved.
1309H -	4873 -	MSR_RELOAD_FIXED_CTRx		Reload value for IA32_FIXED_CTRx (R/W)
130BH	4875	47:0		Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.
		63:48		Reserved.
14C1H -	5313 -	MSR_RELOAD_PMCx	Core	Reload value for IA32_PMCx (R/W)
14C4H	5316	47:0		Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.
		63:48		Reserved.

See Table 2-6, Table 2-12, Table 2-13, and Table 2-14 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_86H.

2.8 MSRS IN PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

Table 2-15 lists model-specific registers (MSRs) that are common for Nehalem microarchitecture. These include the Intel Core i7 and i5 processor family. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, 06_1FH, or 06_2EH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH are listed in Table 2-16. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at <http://biosbits.org>.

The column "Scope" represents the package/core/thread scope of individual bit field of an MSR. "Thread" means this bit field must be programmed on each logical processor independently. "Core" means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. "Package" means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 18.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Package	Model Specific Platform ID (R)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		49:0		Reserved
		52:50		See Table 2-2.
		63:53		Reserved
1BH	27	IA32_APIC_BASE	Thread	See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Running count of SMI events since last RESET.
		63:32		Reserved
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64Processor (R/W) See Table 2-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 2-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDC-TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDC and TDP Limits for Turbo mode are programmable. When set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 133.33MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		23:16		Reserved

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		24		Interrupt filtering enable (R/W) When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.
		25		C3 state auto demotion enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		C1 state auto demotion enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.
		63:19		Reserved
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Thread	See Table 2-2.
		7:0		Event Select
		15:8		UMask
		16		USR
		17		OS
		18		Edge
		19		PC
		20		INT
		21		AnyThread
		22		EN
		23		INV
		31:24		CMASK
		63:32		Reserved
		187H	391	IA32_PERFEVTSEL1
188H	392	IA32_PERFEVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Core	See Table 2-2.
		15:0		Current Performance State Value.
		63:16		Reserved
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved
		3:1		On demand Clock Modulation Duty Cycle (R/W)
		4		On demand Clock Modulation Enable (R/W)
		63:5		Reserved
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Thread	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.
		6:4		Reserved
		7	Thread	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Thread	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Thread	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 2-2.
		21:19		Reserved
		22	Thread	Limit CPUID Maxval (R/W) See Table 2-2.
23	Thread	xTPR Message Disable (R/W) See Table 2-2.		
33:24		Reserved		

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		34	Thread	XD Bit Disable (R/W) See Table 2-2.
		37:35		Reserved
		38	Package	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Thread	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.
		63:24		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1	Core	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.
		63:4		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0	Package	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.
		1	Thread	Energy/Performance Bias Enable (R/W) This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.(EAX=06h):ECX[3].
		63:2		Reserved
1ACH	428	MSR_TURBO_POWER_CURRENT_LIMIT		See http://biosbits.org .
		14:0	Package	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.
		15	Package	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.
		30:16	Package	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.
		31	Package	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.
		63:32		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
		63:9		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register See http://biosbits.org .
		0		Reserved
		1	Package	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MC0_CTL2	Package	See Table 2-2.
281H	641	IA32_MC1_CTL2	Package	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Core	See Table 2-2.
285H	645	IA32_MC5_CTL2	Core	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format See Table 2-2.
		6		PEBS Record Format
		7		PEBSSaveArchRegs See Table 2-2.
		11:8		PEBS_REC_FORMAT See Table 2-2.
		12		SMM_FREEZE See Table 2-2.
		63:13		Reserved
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Thread	Provides single-bit status used by software to query the overflow condition of each performance counter. (R/O)
		61		UNC_Ovf Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities." Allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Thread	(R/W)
		61		CLR_UNC_Ovf Set 1 to clear UNC_Ovf.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 20.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0 (R/W)
		1		Enable PEBS on IA32_PMC1 (R/W)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		2		Enable PEBS on IA32_PMC2 (R/W)
		3		Enable PEBS on IA32_PMC3 (R/W)
		31:4		Reserved
		32		Enable Load Latency on IA32_PMC0 (R/W)
		33		Enable Load Latency on IA32_PMC1 (R/W)
		34		Enable Load Latency on IA32_PMC2 (R/W)
		35		Enable Load Latency on IA32_PMC3 (R/W)
		63:36		Reserved
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 20.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
400H	1024	IA32_MCO_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
402H	1026	IA32_MCO_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MCO_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
406H	1030	IA32_MC1_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	IA32_MC1_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
40AH	1034	IA32_MC2_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40BH	1035	IA32_MC2_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	IA32_MC3_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40FH	1039	IA32_MC3_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
410H	1040	IA32_MC4_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	IA32_MC4_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC4_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
414H	1044	IA32_MC5_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	IA32_MC5_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 17.
41AH	1050	IA32_MC6_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	IA32_MC6_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	IA32_MC7_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 17.
41EH	1054	IA32_MC7_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	IA32_MC7_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 17.
422H	1058	IA32_MC8_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
483H	1155	IA32_VMX_EXIT_CTL5	Thread	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL5	Thread	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Thread	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Thread	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Thread	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.9.1 and record format in Section 18.4.8.1.
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	Last Branch Record 0 To Ip (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID Register (R/O)
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version Register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority Register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority Register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI Register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination Register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector Register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service Register Bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service Register Bits [63:32] (R/O)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service Register Bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service Register Bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service Register Bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service Register Bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service Register Bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service Register Bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode Register Bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode Register Bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode Register Bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode Register Bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode Register Bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode Register Bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode Register Bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode Register Bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request Register Bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request Register Bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request Register Bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request Register Bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request Register Bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request Register Bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request Register Bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request Register Bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status Register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command Register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt Register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt Register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor Register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 Register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 Register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error Register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count Register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count Register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration Register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI Register (w/O)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 18.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

2.8.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

The Intel Xeon Processor 5500 and 3400 series supports additional model-specific registers listed in Table 2-16. These MSRs also apply to the Intel Core i7 and i5 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH; see Table 2-1.

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Actual maximum turbo frequency is multiplied by 133.33MHz. (Not available in model 06_2EH.)
		7:0		Maximum Turbo Ratio Limit 1C (R/O) Maximum Turbo mode ratio limit with 1 core active.
		15:8		Maximum Turbo Ratio Limit 2C (R/O) Maximum Turbo mode ratio limit with 2 cores active.
		23:16		Maximum Turbo Ratio Limit 3C (R/O) Maximum Turbo mode ratio limit with 3 cores active.
		31:24		Maximum Turbo Ratio Limit 4C (R/O) Maximum Turbo mode ratio limit with 4 cores active.
		63:32		Reserved
301H	769	MSR_GQ_SNOOP_MESF	Package	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		From M to S (R/W)
		1		From E to S (R/W)
		2		From S to S (R/W)
		3		From F to S (R/W)
		4		From M to I (R/W)
		5		From E to I (R/W)
		6		From S to I (R/W)
		7		From F to I (R/W)
		63:8		Reserved
391H	913	MSR_UNCORE_PERF_GLOBAL_CTRL	Package	See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."
392H	914	MSR_UNCORE_PERF_GLOBAL_STATUS	Package	See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."
393H	915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	Package	See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."
394H	916	MSR_UNCORE_FIXED_CTRL0	Package	See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."
395H	917	MSR_UNCORE_FIXED_CTRL_CTRL	Package	See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."
396H	918	MSR_UNCORE_ADDR_OPCODE_MATCH	Package	See Section 20.3.1.2.3, "Uncore Address/Opcode Match MSR."
3B0H	960	MSR_UNCORE_PMC0	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B1H	961	MSR_UNCORE_PMC1	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B2H	962	MSR_UNCORE_PMC2	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B3H	963	MSR_UNCORE_PMC3	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B4H	964	MSR_UNCORE_PMC4	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B6H	966	MSR_UNCORE_PMC6	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B7H	967	MSR_UNCORE_PMC7	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C0H	944	MSR_UNCORE_PERFEVTSELO	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C1H	945	MSR_UNCORE_PERFEVTSSEL1	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3C2H	946	MSR_UNCORE_PERFEVTSEL2	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C3H	947	MSR_UNCORE_PERFEVTSEL3	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C4H	948	MSR_UNCORE_PERFEVTSEL4	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C5H	949	MSR_UNCORE_PERFEVTSEL5	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C6H	950	MSR_UNCORE_PERFEVTSEL6	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C7H	951	MSR_UNCORE_PERFEVTSEL7	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."

2.8.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

The Intel Xeon Processor 7500 series supports MSRs listed in Table 2-15 (except MSR address 1ADH) and additional model-specific registers listed in Table 2-17. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2EH.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved Attempt to read/write will cause #UD.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
394H	816	MSR_W_PMON_FIXED_CTR	Package	Uncore W-box perfmon fixed counter.
395H	817	MSR_W_PMON_FIXED_CTR_CTL	Package	Uncore U-box perfmon fixed counter control MSR.
424H	1060	IA32_MC9_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
425H	1061	IA32_MC9_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
426H	1062	IA32_MC9_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
427H	1063	IA32_MC9_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."
428H	1064	IA32_MC10_CTL	Package	See Section 16.3.2.1, "IA32_MCI_CTL MSRS."
429H	1065	IA32_MC10_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
42AH	1066	IA32_MC10_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
42BH	1067	IA32_MC10_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."
42CH	1068	IA32_MC11_CTL	Package	See Section 16.3.2.1, "IA32_MCI_CTL MSRS."
42DH	1069	IA32_MC11_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
42EH	1070	IA32_MC11_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
42FH	1071	IA32_MC11_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."
430H	1072	IA32_MC12_CTL	Package	See Section 16.3.2.1, "IA32_MCI_CTL MSRS."
431H	1073	IA32_MC12_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
432H	1074	IA32_MC12_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
433H	1075	IA32_MC12_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."
434H	1076	IA32_MC13_CTL	Package	See Section 16.3.2.1, "IA32_MCI_CTL MSRS."
435H	1077	IA32_MC13_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
436H	1078	IA32_MC13_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
437H	1079	IA32_MC13_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."
438H	1080	IA32_MC14_CTL	Package	See Section 16.3.2.1, "IA32_MCI_CTL MSRS."
439H	1081	IA32_MC14_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
43AH	1082	IA32_MC14_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
43BH	1083	IA32_MC14_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."
43CH	1084	IA32_MC15_CTL	Package	See Section 16.3.2.1, "IA32_MCI_CTL MSRS."
43DH	1085	IA32_MC15_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
43EH	1086	IA32_MC15_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
43FH	1087	IA32_MC15_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."
440H	1088	IA32_MC16_CTL	Package	See Section 16.3.2.1, "IA32_MCI_CTL MSRS."
441H	1089	IA32_MC16_STATUS	Package	See Section 16.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 17.
442H	1090	IA32_MC16_ADDR	Package	See Section 16.3.2.3, "IA32_MCI_ADDR MSRS."
443H	1091	IA32_MC16_MISC	Package	See Section 16.3.2.4, "IA32_MCI_MISC MSRS."

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
444H	1092	IA32_MC17_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
446H	1094	IA32_MC17_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
44AH	1098	IA32_MC18_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
44EH	1102	IA32_MC19_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
450H	1104	IA32_MC20_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	IA32_MC20_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
452H	1106	IA32_MC20_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
453H	1107	IA32_MC20_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
454H	1108	IA32_MC21_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
455H	1109	IA32_MC21_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
456H	1110	IA32_MC21_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
457H	1111	IA32_MC21_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
C00H	3072	MSR_U_PMON_GLOBAL_CTRL	Package	Uncore U-box perfmon global control MSR.
C01H	3073	MSR_U_PMON_GLOBAL_STATUS	Package	Uncore U-box perfmon global status MSR.
C02H	3074	MSR_U_PMON_GLOBAL_OVF_CTRL	Package	Uncore U-box perfmon global overflow control MSR.
C10H	3088	MSR_U_PMON_EVNT_SEL	Package	Uncore U-box perfmon event select MSR.
C11H	3089	MSR_U_PMON_CTR	Package	Uncore U-box perfmon counter MSR.
C20H	3104	MSR_B0_PMON_BOX_CTRL	Package	Uncore B-box 0 perfmon local box control MSR.
C21H	3105	MSR_B0_PMON_BOX_STATUS	Package	Uncore B-box 0 perfmon local box status MSR.
C22H	3106	MSR_B0_PMON_BOX_OVF_CTRL	Package	Uncore B-box 0 perfmon local box overflow control MSR.
C30H	3120	MSR_B0_PMON_EVNT_SELO	Package	Uncore B-box 0 perfmon event select MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C31H	3121	MSR_B0_PMON_CTRL0	Package	Uncore B-box 0 perfmon counter MSR.
C32H	3122	MSR_B0_PMON_EVNT_SEL1	Package	Uncore B-box 0 perfmon event select MSR.
C33H	3123	MSR_B0_PMON_CTRL1	Package	Uncore B-box 0 perfmon counter MSR.
C34H	3124	MSR_B0_PMON_EVNT_SEL2	Package	Uncore B-box 0 perfmon event select MSR.
C35H	3125	MSR_B0_PMON_CTRL2	Package	Uncore B-box 0 perfmon counter MSR.
C36H	3126	MSR_B0_PMON_EVNT_SEL3	Package	Uncore B-box 0 perfmon event select MSR.
C37H	3127	MSR_B0_PMON_CTRL3	Package	Uncore B-box 0 perfmon counter MSR.
C40H	3136	MSR_S0_PMON_BOX_CTRL	Package	Uncore S-box 0 perfmon local box control MSR.
C41H	3137	MSR_S0_PMON_BOX_STATUS	Package	Uncore S-box 0 perfmon local box status MSR.
C42H	3138	MSR_S0_PMON_BOX_OVF_CTRL	Package	Uncore S-box 0 perfmon local box overflow control MSR.
C50H	3152	MSR_S0_PMON_EVNT_SELO	Package	Uncore S-box 0 perfmon event select MSR.
C51H	3153	MSR_S0_PMON_CTRL0	Package	Uncore S-box 0 perfmon counter MSR.
C52H	3154	MSR_S0_PMON_EVNT_SEL1	Package	Uncore S-box 0 perfmon event select MSR.
C53H	3155	MSR_S0_PMON_CTRL1	Package	Uncore S-box 0 perfmon counter MSR.
C54H	3156	MSR_S0_PMON_EVNT_SEL2	Package	Uncore S-box 0 perfmon event select MSR.
C55H	3157	MSR_S0_PMON_CTRL2	Package	Uncore S-box 0 perfmon counter MSR.
C56H	3158	MSR_S0_PMON_EVNT_SEL3	Package	Uncore S-box 0 perfmon event select MSR.
C57H	3159	MSR_S0_PMON_CTRL3	Package	Uncore S-box 0 perfmon counter MSR.
C60H	3168	MSR_B1_PMON_BOX_CTRL	Package	Uncore B-box 1 perfmon local box control MSR.
C61H	3169	MSR_B1_PMON_BOX_STATUS	Package	Uncore B-box 1 perfmon local box status MSR.
C62H	3170	MSR_B1_PMON_BOX_OVF_CTRL	Package	Uncore B-box 1 perfmon local box overflow control MSR.
C70H	3184	MSR_B1_PMON_EVNT_SELO	Package	Uncore B-box 1 perfmon event select MSR.
C71H	3185	MSR_B1_PMON_CTRL0	Package	Uncore B-box 1 perfmon counter MSR.
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR.
C73H	3187	MSR_B1_PMON_CTRL1	Package	Uncore B-box 1 perfmon counter MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C74H	3188	MSR_B1_PMON_EVNT_SEL2	Package	Uncore B-box 1 perfmon event select MSR.
C75H	3189	MSR_B1_PMON_CTR2	Package	Uncore B-box 1 perfmon counter MSR.
C76H	3190	MSR_B1_PMON_EVNT_SEL3	Package	Uncore B-box 1 vperfmon event select MSR.
C77H	3191	MSR_B1_PMON_CTR3	Package	Uncore B-box 1 perfmon counter MSR.
C80H	3120	MSR_W_PMON_BOX_CTRL	Package	Uncore W-box perfmon local box control MSR.
C81H	3121	MSR_W_PMON_BOX_STATUS	Package	Uncore W-box perfmon local box status MSR.
C82H	3122	MSR_W_PMON_BOX_OVF_CTRL	Package	Uncore W-box perfmon local box overflow control MSR.
C90H	3136	MSR_W_PMON_EVNT_SELO	Package	Uncore W-box perfmon event select MSR.
C91H	3137	MSR_W_PMON_CTR0	Package	Uncore W-box perfmon counter MSR.
C92H	3138	MSR_W_PMON_EVNT_SEL1	Package	Uncore W-box perfmon event select MSR.
C93H	3139	MSR_W_PMON_CTR1	Package	Uncore W-box perfmon counter MSR.
C94H	3140	MSR_W_PMON_EVNT_SEL2	Package	Uncore W-box perfmon event select MSR.
C95H	3141	MSR_W_PMON_CTR2	Package	Uncore W-box perfmon counter MSR.
C96H	3142	MSR_W_PMON_EVNT_SEL3	Package	Uncore W-box perfmon event select MSR.
C97H	3143	MSR_W_PMON_CTR3	Package	Uncore W-box perfmon counter MSR.
CA0H	3232	MSR_M0_PMON_BOX_CTRL	Package	Uncore M-box 0 perfmon local box control MSR.
CA1H	3233	MSR_M0_PMON_BOX_STATUS	Package	Uncore M-box 0 perfmon local box status MSR.
CA2H	3234	MSR_M0_PMON_BOX_OVF_CTRL	Package	Uncore M-box 0 perfmon local box overflow control MSR.
CA4H	3236	MSR_M0_PMON_TIMESTAMP	Package	Uncore M-box 0 perfmon time stamp unit select MSR.
CA5H	3237	MSR_M0_PMON_DSP	Package	Uncore M-box 0 perfmon DSP unit select MSR.
CA6H	3238	MSR_M0_PMON_ISS	Package	Uncore M-box 0 perfmon ISS unit select MSR.
CA7H	3239	MSR_M0_PMON_MAP	Package	Uncore M-box 0 perfmon MAP unit select MSR.
CA8H	3240	MSR_M0_PMON_MSC_THR	Package	Uncore M-box 0 perfmon MIC THR select MSR.
CA9H	3241	MSR_M0_PMON_PGT	Package	Uncore M-box 0 perfmon PGT unit select MSR.
CAAH	3242	MSR_M0_PMON_PLD	Package	Uncore M-box 0 perfmon PLD unit select MSR.
CABH	3243	MSR_M0_PMON_ZDP	Package	Uncore M-box 0 perfmon ZDP unit select MSR.
CB0H	3248	MSR_M0_PMON_EVNT_SELO	Package	Uncore M-box 0 perfmon event select MSR.
CB1H	3249	MSR_M0_PMON_CTR0	Package	Uncore M-box 0 perfmon counter MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CB2H	3250	MSR_M0_PMON_EVNT_SEL1	Package	Uncore M-box 0 perfmon event select MSR.
CB3H	3251	MSR_M0_PMON_CTR1	Package	Uncore M-box 0 perfmon counter MSR.
CB4H	3252	MSR_M0_PMON_EVNT_SEL2	Package	Uncore M-box 0 perfmon event select MSR.
CB5H	3253	MSR_M0_PMON_CTR2	Package	Uncore M-box 0 perfmon counter MSR.
CB6H	3254	MSR_M0_PMON_EVNT_SEL3	Package	Uncore M-box 0 perfmon event select MSR.
CB7H	3255	MSR_M0_PMON_CTR3	Package	Uncore M-box 0 perfmon counter MSR.
CB8H	3256	MSR_M0_PMON_EVNT_SEL4	Package	Uncore M-box 0 perfmon event select MSR.
CB9H	3257	MSR_M0_PMON_CTR4	Package	Uncore M-box 0 perfmon counter MSR.
CBAH	3258	MSR_M0_PMON_EVNT_SEL5	Package	Uncore M-box 0 perfmon event select MSR.
CBBH	3259	MSR_M0_PMON_CTR5	Package	Uncore M-box 0 perfmon counter MSR.
CC0H	3264	MSR_S1_PMON_BOX_CTRL	Package	Uncore S-box 1 perfmon local box control MSR.
CC1H	3265	MSR_S1_PMON_BOX_STATUS	Package	Uncore S-box 1 perfmon local box status MSR.
CC2H	3266	MSR_S1_PMON_BOX_OVF_CTRL	Package	Uncore S-box 1 perfmon local box overflow control MSR.
CDOH	3280	MSR_S1_PMON_EVNT_SELO	Package	Uncore S-box 1 perfmon event select MSR.
CD1H	3281	MSR_S1_PMON_CTR0	Package	Uncore S-box 1 perfmon counter MSR.
CD2H	3282	MSR_S1_PMON_EVNT_SEL1	Package	Uncore S-box 1 perfmon event select MSR.
CD3H	3283	MSR_S1_PMON_CTR1	Package	Uncore S-box 1 perfmon counter MSR.
CD4H	3284	MSR_S1_PMON_EVNT_SEL2	Package	Uncore S-box 1 perfmon event select MSR.
CD5H	3285	MSR_S1_PMON_CTR2	Package	Uncore S-box 1 perfmon counter MSR.
CD6H	3286	MSR_S1_PMON_EVNT_SEL3	Package	Uncore S-box 1 perfmon event select MSR.
CD7H	3287	MSR_S1_PMON_CTR3	Package	Uncore S-box 1 perfmon counter MSR.
CE0H	3296	MSR_M1_PMON_BOX_CTRL	Package	Uncore M-box 1 perfmon local box control MSR.
CE1H	3297	MSR_M1_PMON_BOX_STATUS	Package	Uncore M-box 1 perfmon local box status MSR.
CE2H	3298	MSR_M1_PMON_BOX_OVF_CTRL	Package	Uncore M-box 1 perfmon local box overflow control MSR.
CE4H	3300	MSR_M1_PMON_TIMESTAMP	Package	Uncore M-box 1 perfmon time stamp unit select MSR.
CE5H	3301	MSR_M1_PMON_DSP	Package	Uncore M-box 1 perfmon DSP unit select MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CE6H	3302	MSR_M1_PMON_ISS	Package	Uncore M-box 1 perfmon ISS unit select MSR.
CE7H	3303	MSR_M1_PMON_MAP	Package	Uncore M-box 1 perfmon MAP unit select MSR.
CE8H	3304	MSR_M1_PMON_MSC_THR	Package	Uncore M-box 1 perfmon MIC THR select MSR.
CE9H	3305	MSR_M1_PMON_PGT	Package	Uncore M-box 1 perfmon PGT unit select MSR.
CEAH	3306	MSR_M1_PMON_PLD	Package	Uncore M-box 1 perfmon PLD unit select MSR.
CEBH	3307	MSR_M1_PMON_ZDP	Package	Uncore M-box 1 perfmon ZDP unit select MSR.
CF0H	3312	MSR_M1_PMON_EVNT_SEL0	Package	Uncore M-box 1 perfmon event select MSR.
CF1H	3313	MSR_M1_PMON_CTRL0	Package	Uncore M-box 1 perfmon counter MSR.
CF2H	3314	MSR_M1_PMON_EVNT_SEL1	Package	Uncore M-box 1 perfmon event select MSR.
CF3H	3315	MSR_M1_PMON_CTRL1	Package	Uncore M-box 1 perfmon counter MSR.
CF4H	3316	MSR_M1_PMON_EVNT_SEL2	Package	Uncore M-box 1 perfmon event select MSR.
CF5H	3317	MSR_M1_PMON_CTRL2	Package	Uncore M-box 1 perfmon counter MSR.
CF6H	3318	MSR_M1_PMON_EVNT_SEL3	Package	Uncore M-box 1 perfmon event select MSR.
CF7H	3319	MSR_M1_PMON_CTRL3	Package	Uncore M-box 1 perfmon counter MSR.
CF8H	3320	MSR_M1_PMON_EVNT_SEL4	Package	Uncore M-box 1 perfmon event select MSR.
CF9H	3321	MSR_M1_PMON_CTRL4	Package	Uncore M-box 1 perfmon counter MSR.
CFAH	3322	MSR_M1_PMON_EVNT_SEL5	Package	Uncore M-box 1 perfmon event select MSR.
CFBH	3323	MSR_M1_PMON_CTRL5	Package	Uncore M-box 1 perfmon counter MSR.
D00H	3328	MSR_CO_PMON_BOX_CTRL	Package	Uncore C-box 0 perfmon local box control MSR.
D01H	3329	MSR_CO_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon local box status MSR.
D02H	3330	MSR_CO_PMON_BOX_OVF_CTRL	Package	Uncore C-box 0 perfmon local box overflow control MSR.
D10H	3344	MSR_CO_PMON_EVNT_SEL0	Package	Uncore C-box 0 perfmon event select MSR.
D11H	3345	MSR_CO_PMON_CTRL0	Package	Uncore C-box 0 perfmon counter MSR.
D12H	3346	MSR_CO_PMON_EVNT_SEL1	Package	Uncore C-box 0 perfmon event select MSR.
D13H	3347	MSR_CO_PMON_CTRL1	Package	Uncore C-box 0 perfmon counter MSR.
D14H	3348	MSR_CO_PMON_EVNT_SEL2	Package	Uncore C-box 0 perfmon event select MSR.
D15H	3349	MSR_CO_PMON_CTRL2	Package	Uncore C-box 0 perfmon counter MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D16H	3350	MSR_C0_PMON_EVNT_SEL3	Package	Uncore C-box 0 perfmon event select MSR.
D17H	3351	MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter MSR.
D18H	3352	MSR_C0_PMON_EVNT_SEL4	Package	Uncore C-box 0 perfmon event select MSR.
D19H	3353	MSR_C0_PMON_CTR4	Package	Uncore C-box 0 perfmon counter MSR.
D1AH	3354	MSR_C0_PMON_EVNT_SEL5	Package	Uncore C-box 0 perfmon event select MSR.
D1BH	3355	MSR_C0_PMON_CTR5	Package	Uncore C-box 0 perfmon counter MSR.
D20H	3360	MSR_C4_PMON_BOX_CTRL	Package	Uncore C-box 4 perfmon local box control MSR.
D21H	3361	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon local box status MSR.
D22H	3362	MSR_C4_PMON_BOX_OVF_CTRL	Package	Uncore C-box 4 perfmon local box overflow control MSR.
D30H	3376	MSR_C4_PMON_EVNT_SELO	Package	Uncore C-box 4 perfmon event select MSR.
D31H	3377	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter MSR.
D32H	3378	MSR_C4_PMON_EVNT_SEL1	Package	Uncore C-box 4 perfmon event select MSR.
D33H	3379	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter MSR.
D34H	3380	MSR_C4_PMON_EVNT_SEL2	Package	Uncore C-box 4 perfmon event select MSR.
D35H	3381	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter MSR.
D36H	3382	MSR_C4_PMON_EVNT_SEL3	Package	Uncore C-box 4 perfmon event select MSR.
D37H	3383	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter MSR.
D38H	3384	MSR_C4_PMON_EVNT_SEL4	Package	Uncore C-box 4 perfmon event select MSR.
D39H	3385	MSR_C4_PMON_CTR4	Package	Uncore C-box 4 perfmon counter MSR.
D3AH	3386	MSR_C4_PMON_EVNT_SEL5	Package	Uncore C-box 4 perfmon event select MSR.
D3BH	3387	MSR_C4_PMON_CTR5	Package	Uncore C-box 4 perfmon counter MSR.
D40H	3392	MSR_C2_PMON_BOX_CTRL	Package	Uncore C-box 2 perfmon local box control MSR.
D41H	3393	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon local box status MSR.
D42H	3394	MSR_C2_PMON_BOX_OVF_CTRL	Package	Uncore C-box 2 perfmon local box overflow control MSR.
D50H	3408	MSR_C2_PMON_EVNT_SELO	Package	Uncore C-box 2 perfmon event select MSR.
D51H	3409	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D52H	3410	MSR_C2_PMON_EVNT_SEL1	Package	Uncore C-box 2 perfmon event select MSR.
D53H	3411	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter MSR.
D54H	3412	MSR_C2_PMON_EVNT_SEL2	Package	Uncore C-box 2 perfmon event select MSR.
D55H	3413	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter MSR.
D56H	3414	MSR_C2_PMON_EVNT_SEL3	Package	Uncore C-box 2 perfmon event select MSR.
D57H	3415	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter MSR.
D58H	3416	MSR_C2_PMON_EVNT_SEL4	Package	Uncore C-box 2 perfmon event select MSR.
D59H	3417	MSR_C2_PMON_CTR4	Package	Uncore C-box 2 perfmon counter MSR.
D5AH	3418	MSR_C2_PMON_EVNT_SEL5	Package	Uncore C-box 2 perfmon event select MSR.
D5BH	3419	MSR_C2_PMON_CTR5	Package	Uncore C-box 2 perfmon counter MSR.
D60H	3424	MSR_C6_PMON_BOX_CTRL	Package	Uncore C-box 6 perfmon local box control MSR.
D61H	3425	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon local box status MSR.
D62H	3426	MSR_C6_PMON_BOX_OVF_CTRL	Package	Uncore C-box 6 perfmon local box overflow control MSR.
D70H	3440	MSR_C6_PMON_EVNT_SELO	Package	Uncore C-box 6 perfmon event select MSR.
D71H	3441	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter MSR.
D72H	3442	MSR_C6_PMON_EVNT_SEL1	Package	Uncore C-box 6 perfmon event select MSR.
D73H	3443	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter MSR.
D74H	3444	MSR_C6_PMON_EVNT_SEL2	Package	Uncore C-box 6 perfmon event select MSR.
D75H	3445	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter MSR.
D76H	3446	MSR_C6_PMON_EVNT_SEL3	Package	Uncore C-box 6 perfmon event select MSR.
D77H	3447	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter MSR.
D78H	3448	MSR_C6_PMON_EVNT_SEL4	Package	Uncore C-box 6 perfmon event select MSR.
D79H	3449	MSR_C6_PMON_CTR4	Package	Uncore C-box 6 perfmon counter MSR.
D7AH	3450	MSR_C6_PMON_EVNT_SEL5	Package	Uncore C-box 6 perfmon event select MSR.
D7BH	3451	MSR_C6_PMON_CTR5	Package	Uncore C-box 6 perfmon counter MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D80H	3456	MSR_C1_PMON_BOX_CTRL	Package	Uncore C-box 1 perfmon local box control MSR.
D81H	3457	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon local box status MSR.
D82H	3458	MSR_C1_PMON_BOX_OVF_CTRL	Package	Uncore C-box 1 perfmon local box overflow control MSR.
D90H	3472	MSR_C1_PMON_EVNT_SELO	Package	Uncore C-box 1 perfmon event select MSR.
D91H	3473	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter MSR.
D92H	3474	MSR_C1_PMON_EVNT_SEL1	Package	Uncore C-box 1 perfmon event select MSR.
D93H	3475	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter MSR.
D94H	3476	MSR_C1_PMON_EVNT_SEL2	Package	Uncore C-box 1 perfmon event select MSR.
D95H	3477	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter MSR.
D96H	3478	MSR_C1_PMON_EVNT_SEL3	Package	Uncore C-box 1 perfmon event select MSR.
D97H	3479	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter MSR.
D98H	3480	MSR_C1_PMON_EVNT_SEL4	Package	Uncore C-box 1 perfmon event select MSR.
D99H	3481	MSR_C1_PMON_CTR4	Package	Uncore C-box 1 perfmon counter MSR.
D9AH	3482	MSR_C1_PMON_EVNT_SEL5	Package	Uncore C-box 1 perfmon event select MSR.
D9BH	3483	MSR_C1_PMON_CTR5	Package	Uncore C-box 1 perfmon counter MSR.
DA0H	3488	MSR_C5_PMON_BOX_CTRL	Package	Uncore C-box 5 perfmon local box control MSR.
DA1H	3489	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon local box status MSR.
DA2H	3490	MSR_C5_PMON_BOX_OVF_CTRL	Package	Uncore C-box 5 perfmon local box overflow control MSR.
DB0H	3504	MSR_C5_PMON_EVNT_SELO	Package	Uncore C-box 5 perfmon event select MSR.
DB1H	3505	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter MSR.
DB2H	3506	MSR_C5_PMON_EVNT_SEL1	Package	Uncore C-box 5 perfmon event select MSR.
DB3H	3507	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter MSR.
DB4H	3508	MSR_C5_PMON_EVNT_SEL2	Package	Uncore C-box 5 perfmon event select MSR.
DB5H	3509	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter MSR.
DB6H	3510	MSR_C5_PMON_EVNT_SEL3	Package	Uncore C-box 5 perfmon event select MSR.
DB7H	3511	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DB8H	3512	MSR_C5_PMON_EVNT_SEL4	Package	Uncore C-box 5 perfmon event select MSR.
DB9H	3513	MSR_C5_PMON_CTR4	Package	Uncore C-box 5 perfmon counter MSR.
DBAH	3514	MSR_C5_PMON_EVNT_SEL5	Package	Uncore C-box 5 perfmon event select MSR.
DBBH	3515	MSR_C5_PMON_CTR5	Package	Uncore C-box 5 perfmon counter MSR.
DC0H	3520	MSR_C3_PMON_BOX_CTRL	Package	Uncore C-box 3 perfmon local box control MSR.
DC1H	3521	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon local box status MSR.
DC2H	3522	MSR_C3_PMON_BOX_OVF_CTRL	Package	Uncore C-box 3 perfmon local box overflow control MSR.
DD0H	3536	MSR_C3_PMON_EVNT_SELO	Package	Uncore C-box 3 perfmon event select MSR.
DD1H	3537	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter MSR.
DD2H	3538	MSR_C3_PMON_EVNT_SEL1	Package	Uncore C-box 3 perfmon event select MSR.
DD3H	3539	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter MSR.
DD4H	3540	MSR_C3_PMON_EVNT_SEL2	Package	Uncore C-box 3 perfmon event select MSR.
DD5H	3541	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter MSR.
DD6H	3542	MSR_C3_PMON_EVNT_SEL3	Package	Uncore C-box 3 perfmon event select MSR.
DD7H	3543	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter MSR.
DD8H	3544	MSR_C3_PMON_EVNT_SEL4	Package	Uncore C-box 3 perfmon event select MSR.
DD9H	3545	MSR_C3_PMON_CTR4	Package	Uncore C-box 3 perfmon counter MSR.
DDAH	3546	MSR_C3_PMON_EVNT_SEL5	Package	Uncore C-box 3 perfmon event select MSR.
DDBH	3547	MSR_C3_PMON_CTR5	Package	Uncore C-box 3 perfmon counter MSR.
DE0H	3552	MSR_C7_PMON_BOX_CTRL	Package	Uncore C-box 7 perfmon local box control MSR.
DE1H	3553	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon local box status MSR.
DE2H	3554	MSR_C7_PMON_BOX_OVF_CTRL	Package	Uncore C-box 7 perfmon local box overflow control MSR.
DF0H	3568	MSR_C7_PMON_EVNT_SELO	Package	Uncore C-box 7 perfmon event select MSR.
DF1H	3569	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter MSR.
DF2H	3570	MSR_C7_PMON_EVNT_SEL1	Package	Uncore C-box 7 perfmon event select MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DF3H	3571	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter MSR.
DF4H	3572	MSR_C7_PMON_EVNT_SEL2	Package	Uncore C-box 7 perfmon event select MSR.
DF5H	3573	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter MSR.
DF6H	3574	MSR_C7_PMON_EVNT_SEL3	Package	Uncore C-box 7 perfmon event select MSR.
DF7H	3575	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter MSR.
DF8H	3576	MSR_C7_PMON_EVNT_SEL4	Package	Uncore C-box 7 perfmon event select MSR.
DF9H	3577	MSR_C7_PMON_CTR4	Package	Uncore C-box 7 perfmon counter MSR.
DFAH	3578	MSR_C7_PMON_EVNT_SEL5	Package	Uncore C-box 7 perfmon event select MSR.
DFBH	3579	MSR_C7_PMON_CTR5	Package	Uncore C-box 7 perfmon counter MSR.
E00H	3584	MSR_R0_PMON_BOX_CTRL	Package	Uncore R-box 0 perfmon local box control MSR.
E01H	3585	MSR_R0_PMON_BOX_STATUS	Package	Uncore R-box 0 perfmon local box status MSR.
E02H	3586	MSR_R0_PMON_BOX_OVF_CTRL	Package	Uncore R-box 0 perfmon local box overflow control MSR.
E04H	3588	MSR_R0_PMON_IPERF0_P0	Package	Uncore R-box 0 perfmon IPERF0 unit Port 0 select MSR.
E05H	3589	MSR_R0_PMON_IPERF0_P1	Package	Uncore R-box 0 perfmon IPERF0 unit Port 1 select MSR.
E06H	3590	MSR_R0_PMON_IPERF0_P2	Package	Uncore R-box 0 perfmon IPERF0 unit Port 2 select MSR.
E07H	3591	MSR_R0_PMON_IPERF0_P3	Package	Uncore R-box 0 perfmon IPERF0 unit Port 3 select MSR.
E08H	3592	MSR_R0_PMON_IPERF0_P4	Package	Uncore R-box 0 perfmon IPERF0 unit Port 4 select MSR.
E09H	3593	MSR_R0_PMON_IPERF0_P5	Package	Uncore R-box 0 perfmon IPERF0 unit Port 5 select MSR.
E0AH	3594	MSR_R0_PMON_IPERF0_P6	Package	Uncore R-box 0 perfmon IPERF0 unit Port 6 select MSR.
E0BH	3595	MSR_R0_PMON_IPERF0_P7	Package	Uncore R-box 0 perfmon IPERF0 unit Port 7 select MSR.
E0CH	3596	MSR_R0_PMON_QLX_P0	Package	Uncore R-box 0 perfmon QLX unit Port 0 select MSR.
E0DH	3597	MSR_R0_PMON_QLX_P1	Package	Uncore R-box 0 perfmon QLX unit Port 1 select MSR.
E0EH	3598	MSR_R0_PMON_QLX_P2	Package	Uncore R-box 0 perfmon QLX unit Port 2 select MSR.
E0FH	3599	MSR_R0_PMON_QLX_P3	Package	Uncore R-box 0 perfmon QLX unit Port 3 select MSR.
E10H	3600	MSR_R0_PMON_EVNT_SELO	Package	Uncore R-box 0 perfmon event select MSR.
E11H	3601	MSR_R0_PMON_CTR0	Package	Uncore R-box 0 perfmon counter MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E12H	3602	MSR_R0_PMON_EVNT_SEL1	Package	Uncore R-box 0 perfmon event select MSR.
E13H	3603	MSR_R0_PMON_CTR1	Package	Uncore R-box 0 perfmon counter MSR.
E14H	3604	MSR_R0_PMON_EVNT_SEL2	Package	Uncore R-box 0 perfmon event select MSR.
E15H	3605	MSR_R0_PMON_CTR2	Package	Uncore R-box 0 perfmon counter MSR.
E16H	3606	MSR_R0_PMON_EVNT_SEL3	Package	Uncore R-box 0 perfmon event select MSR.
E17H	3607	MSR_R0_PMON_CTR3	Package	Uncore R-box 0 perfmon counter MSR.
E18H	3608	MSR_R0_PMON_EVNT_SEL4	Package	Uncore R-box 0 perfmon event select MSR.
E19H	3609	MSR_R0_PMON_CTR4	Package	Uncore R-box 0 perfmon counter MSR.
E1AH	3610	MSR_R0_PMON_EVNT_SEL5	Package	Uncore R-box 0 perfmon event select MSR.
E1BH	3611	MSR_R0_PMON_CTR5	Package	Uncore R-box 0 perfmon counter MSR.
E1CH	3612	MSR_R0_PMON_EVNT_SEL6	Package	Uncore R-box 0 perfmon event select MSR.
E1DH	3613	MSR_R0_PMON_CTR6	Package	Uncore R-box 0 perfmon counter MSR.
E1EH	3614	MSR_R0_PMON_EVNT_SEL7	Package	Uncore R-box 0 perfmon event select MSR.
E1FH	3615	MSR_R0_PMON_CTR7	Package	Uncore R-box 0 perfmon counter MSR.
E20H	3616	MSR_R1_PMON_BOX_CTRL	Package	Uncore R-box 1 perfmon local box control MSR.
E21H	3617	MSR_R1_PMON_BOX_STATUS	Package	Uncore R-box 1 perfmon local box status MSR.
E22H	3618	MSR_R1_PMON_BOX_OVF_CTRL	Package	Uncore R-box 1 perfmon local box overflow control MSR.
E24H	3620	MSR_R1_PMON_IPERF1_P8	Package	Uncore R-box 1 perfmon IPERF1 unit Port 8 select MSR.
E25H	3621	MSR_R1_PMON_IPERF1_P9	Package	Uncore R-box 1 perfmon IPERF1 unit Port 9 select MSR.
E26H	3622	MSR_R1_PMON_IPERF1_P10	Package	Uncore R-box 1 perfmon IPERF1 unit Port 10 select MSR.
E27H	3623	MSR_R1_PMON_IPERF1_P11	Package	Uncore R-box 1 perfmon IPERF1 unit Port 11 select MSR.
E28H	3624	MSR_R1_PMON_IPERF1_P12	Package	Uncore R-box 1 perfmon IPERF1 unit Port 12 select MSR.
E29H	3625	MSR_R1_PMON_IPERF1_P13	Package	Uncore R-box 1 perfmon IPERF1 unit Port 13 select MSR.
E2AH	3626	MSR_R1_PMON_IPERF1_P14	Package	Uncore R-box 1 perfmon IPERF1 unit Port 14 select MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2BH	3627	MSR_R1_PMON_IPERF1_P15	Package	Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR.
E2CH	3628	MSR_R1_PMON_QLX_P4	Package	Uncore R-box 1 perfmon QLX unit Port 4 select MSR.
E2DH	3629	MSR_R1_PMON_QLX_P5	Package	Uncore R-box 1 perfmon QLX unit Port 5 select MSR.
E2EH	3630	MSR_R1_PMON_QLX_P6	Package	Uncore R-box 1 perfmon QLX unit Port 6 select MSR.
E2FH	3631	MSR_R1_PMON_QLX_P7	Package	Uncore R-box 1 perfmon QLX unit Port 7 select MSR.
E30H	3632	MSR_R1_PMON_EVNT_SEL8	Package	Uncore R-box 1 perfmon event select MSR.
E31H	3633	MSR_R1_PMON_CTR8	Package	Uncore R-box 1 perfmon counter MSR.
E32H	3634	MSR_R1_PMON_EVNT_SEL9	Package	Uncore R-box 1 perfmon event select MSR.
E33H	3635	MSR_R1_PMON_CTR9	Package	Uncore R-box 1 perfmon counter MSR.
E34H	3636	MSR_R1_PMON_EVNT_SEL10	Package	Uncore R-box 1 perfmon event select MSR.
E35H	3637	MSR_R1_PMON_CTR10	Package	Uncore R-box 1 perfmon counter MSR.
E36H	3638	MSR_R1_PMON_EVNT_SEL11	Package	Uncore R-box 1 perfmon event select MSR.
E37H	3639	MSR_R1_PMON_CTR11	Package	Uncore R-box 1 perfmon counter MSR.
E38H	3640	MSR_R1_PMON_EVNT_SEL12	Package	Uncore R-box 1 perfmon event select MSR.
E39H	3641	MSR_R1_PMON_CTR12	Package	Uncore R-box 1 perfmon counter MSR.
E3AH	3642	MSR_R1_PMON_EVNT_SEL13	Package	Uncore R-box 1 perfmon event select MSR.
E3BH	3643	MSR_R1_PMON_CTR13	Package	Uncore R-box 1 perfmon counter MSR.
E3CH	3644	MSR_R1_PMON_EVNT_SEL14	Package	Uncore R-box 1 perfmon event select MSR.
E3DH	3645	MSR_R1_PMON_CTR14	Package	Uncore R-box 1 perfmon counter MSR.
E3EH	3646	MSR_R1_PMON_EVNT_SEL15	Package	Uncore R-box 1 perfmon event select MSR.
E3FH	3647	MSR_R1_PMON_CTR15	Package	Uncore R-box 1 perfmon counter MSR.
E45H	3653	MSR_B0_PMON_MATCH	Package	Uncore B-box 0 perfmon local box match MSR.
E46H	3654	MSR_B0_PMON_MASK	Package	Uncore B-box 0 perfmon local box mask MSR.
E49H	3657	MSR_S0_PMON_MATCH	Package	Uncore S-box 0 perfmon local box match MSR.
E4AH	3658	MSR_S0_PMON_MASK	Package	Uncore S-box 0 perfmon local box mask MSR.
E4DH	3661	MSR_B1_PMON_MATCH	Package	Uncore B-box 1 perfmon local box match MSR.
E4EH	3662	MSR_B1_PMON_MASK	Package	Uncore B-box 1 perfmon local box mask MSR.
E54H	3668	MSR_M0_PMON_MM_CONFIG	Package	Uncore M-box 0 perfmon local box address match/mask config MSR.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E55H	3669	MSR_M0_PMON_ADDR_MATCH	Package	Uncore M-box 0 perfmon local box address match MSR.
E56H	3670	MSR_M0_PMON_ADDR_MASK	Package	Uncore M-box 0 perfmon local box address mask MSR.
E59H	3673	MSR_S1_PMON_MATCH	Package	Uncore S-box 1 perfmon local box match MSR.
E5AH	3674	MSR_S1_PMON_MASK	Package	Uncore S-box 1 perfmon local box mask MSR.
E5CH	3676	MSR_M1_PMON_MM_CONFIG	Package	Uncore M-box 1 perfmon local box address match/mask config MSR.
E5DH	3677	MSR_M1_PMON_ADDR_MATCH	Package	Uncore M-box 1 perfmon local box address match MSR.
E5EH	3678	MSR_M1_PMON_ADDR_MASK	Package	Uncore M-box 1 perfmon local box address mask MSR.
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."

2.9 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor 5600 Series is based on Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15, Table 2-16, plus additional MSRs listed in Table 2-18. These MSRs apply to the Intel Core i7, i5, and i3 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_25H or 06_2CH; see Table 2-1.

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		63:48		Reserved
1BOH	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.

2.10 MSRS IN THE INTEL® XEON® PROCESSOR E7 FAMILY BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor E7 Family is based on the Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15 (except MSR address 1ADH), Table 2-16, plus additional MSRs listed in Table 2-19. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2FH.

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:2		Reserved
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved Attempt to read/write will cause #UD.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
F40H	3904	MSR_C8_PMON_BOX_CTRL	Package	Uncore C-box 8 perfmon local box control MSR.
F41H	3905	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-box 8 perfmon local box status MSR.
F42H	3906	MSR_C8_PMON_BOX_OVF_CTRL	Package	Uncore C-box 8 perfmon local box overflow control MSR.
F50H	3920	MSR_C8_PMON_EVNT_SELO	Package	Uncore C-box 8 perfmon event select MSR.
F51H	3921	MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter MSR.
F52H	3922	MSR_C8_PMON_EVNT_SEL1	Package	Uncore C-box 8 perfmon event select MSR.
F53H	3923	MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter MSR.
F54H	3924	MSR_C8_PMON_EVNT_SEL2	Package	Uncore C-box 8 perfmon event select MSR.
F55H	3925	MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter MSR.
F56H	3926	MSR_C8_PMON_EVNT_SEL3	Package	Uncore C-box 8 perfmon event select MSR.
F57H	3927	MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter MSR.
F58H	3928	MSR_C8_PMON_EVNT_SEL4	Package	Uncore C-box 8 perfmon event select MSR.
F59H	3929	MSR_C8_PMON_CTR4	Package	Uncore C-box 8 perfmon counter MSR.
F5AH	3930	MSR_C8_PMON_EVNT_SEL5	Package	Uncore C-box 8 perfmon event select MSR.
F5BH	3931	MSR_C8_PMON_CTR5	Package	Uncore C-box 8 perfmon counter MSR.
FC0H	4032	MSR_C9_PMON_BOX_CTRL	Package	Uncore C-box 9 perfmon local box control MSR.
FC1H	4033	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-box 9 perfmon local box status MSR.
FC2H	4034	MSR_C9_PMON_BOX_OVF_CTRL	Package	Uncore C-box 9 perfmon local box overflow control MSR.
FD0H	4048	MSR_C9_PMON_EVNT_SELO	Package	Uncore C-box 9 perfmon event select MSR.
FD1H	4049	MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter MSR.
FD2H	4050	MSR_C9_PMON_EVNT_SEL1	Package	Uncore C-box 9 perfmon event select MSR.
FD3H	4051	MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter MSR.

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
FD4H	4052	MSR_C9_PMON_EVNT_SEL2	Package	Uncore C-box 9 perfmon event select MSR.
FD5H	4053	MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter MSR.
FD6H	4054	MSR_C9_PMON_EVNT_SEL3	Package	Uncore C-box 9 perfmon event select MSR.
FD7H	4055	MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter MSR.
FD8H	4056	MSR_C9_PMON_EVNT_SEL4	Package	Uncore C-box 9 perfmon event select MSR.
FD9H	4057	MSR_C9_PMON_CTR4	Package	Uncore C-box 9 perfmon counter MSR.
FDAH	4058	MSR_C9_PMON_EVNT_SEL5	Package	Uncore C-box 9 perfmon event select MSR.
FDBH	4059	MSR_C9_PMON_CTR5	Package	Uncore C-box 9 perfmon counter MSR.

2.11 MSRS IN THE INTEL® PROCESSOR FAMILY BASED ON SANDY BRIDGE MICROARCHITECTURE

Table 2-20 lists model-specific registers (MSRs) that are common to the Intel® processor family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH or 06_2DH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH are listed in Table 2-21.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 2.23, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.23, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 9.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 18.17, “Time-Stamp Counter,” and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 11.4.4, “Local APIC Status and Location,” and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Count SMIs.
		63:32		Reserved.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 2-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 2-2.
C5H	197	IA32_PMC4	Core	Performance Counter Register (if core not shared by threads)
C6H	198	IA32_PMC5	Core	Performance Counter Register (if core not shared by threads)
C7H	199	IA32_PMC6	Core	Performance Counter Register (if core not shared by threads)
C8H	200	IA32_PMC7	Core	Performance Counter Register (if core not shared by threads)
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.
		28		Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.
		63:29		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		C-State Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.
		63:19		Reserved
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSEL0	Thread	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 2-2.
18AH	394	IA32_PERFEVTSEL4	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 4.
18BH	395	IA32_PERFEVTSEL5	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 5.
18CH	396	IA32_PERFEVTSEL6	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 6.
18DH	397	IA32_PERFEVTSEL7	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 7.
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15:0		Current Performance State Value
		63:16		Reserved
198H	408	MSR_PERF_STATUS	Package	Performance Status
		47:32		Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2 ¹³).
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		3:0		On demand Clock Modulation Duty Cycle (R/W) In 6.25% increment.
		4		On demand Clock Modulation Enable (R/W)
		63:5		Reserved
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal Threshold #1 Status (R/O) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.		

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		10		Power Limitation Status (R/O) See Table 2-2.
		11		Power Limitation Log (R/WCO) See Table 2-2.
		15:12		Reserved
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.
		31		Reading Valid (R/O) See Table 2-2.
		63:32		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 2-2
		6:1		Reserved
		7	Thread	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Thread	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Thread	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Thread	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
		22	Thread	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Thread	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
34	Thread	XD Bit Disable (R/W) See Table 2-2.		

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		37:35		Reserved
		38	Package	<p>Turbo Mode Disable (R/W)</p> <p>When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0).</p> <p>When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled.</p> <p>Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.</p>
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Unique	Temperature Target
		15:0		Reserved
		23:16		<p>Temperature Target (R)</p> <p>The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.</p>
		63:24		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	<p>L2 Hardware Prefetcher Disable (R/W)</p> <p>If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.</p>
		1	Core	<p>L2 Adjacent Cache Line Prefetcher Disable (R/W)</p> <p>If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).</p>
		2	Core	<p>DCU Hardware Prefetcher Disable (R/W)</p> <p>If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.</p>
		3	Core	<p>DCU IP Prefetcher Disable (R/W)</p> <p>If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.</p>
		63:4		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1A7H	422	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		<p>Miscellaneous Power Management Control</p> <p>Various model specific features enumeration. See http://biosbits.org.</p>

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
63:9		Reserved		
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.
		0		LBR: Last Branch Record
		1		BTF
		5:2		Reserved
		6		TR: Branch Trace
		7		BTS: Log Branch Trace Message to BTS buffer
		8		BTINT
		9		BTS_OFF_OS
		10		BTS_OFF_USER
		11		FREEZE_LBR_ON_PMI
		12		FREEZE_PERFMON_ON_PMI
		13		ENABLE_UNCORE_PMI
		14		FREEZE_WHILE_SMM
63:15		Reserved		
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org .
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MCO_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	Always 0 (CMCI not supported).
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format See Table 2-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs See Table 2-2.
		11:8		PEBS_REC_FORMAT See Table 2-2.
		12		SMM_FREEZE See Table 2-2.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
		0	Thread	Ovf_PMC0
		1	Thread	Ovf_PMC1
		2	Thread	Ovf_PMC2
		3	Thread	Ovf_PMC3
		4	Core	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5	Core	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Core	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)
		7	Core	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)
		31:8		Reserved
		32	Thread	Ovf_FixedCtr0
		33	Thread	Ovf_FixedCtr1
		34	Thread	Ovf_FixedCtr2
		60:35		Reserved
		61	Thread	Ovf_Uncore
		62	Thread	Ovf_BufDSSAVE
		63	Thread	CondChgd
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
		0	Thread	Set 1 to enable PMC0 to count.
		1	Thread	Set 1 to enable PMC1 to count.
		2	Thread	Set 1 to enable PMC2 to count.
		3	Thread	Set 1 to enable PMC3 to count.
		4	Core	Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4).
		5	Core	Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5).
		6	Core	Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6).
		7	Core	Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to enable FixedCtr0 to count.
		33	Thread	Set 1 to enable FixedCtr1 to count.
		34	Thread	Set 1 to enable FixedCtr2 to count.
		63:35		Reserved
390H	912	IA32_PERF_GLOBAL_OVF_CTRL		See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
		0	Thread	Set 1 to clear Ovf_PMC0.
		1	Thread	Set 1 to clear Ovf_PMC1.
		2	Thread	Set 1 to clear Ovf_PMC2.
		3	Thread	Set 1 to clear Ovf_PMC3.
		4	Core	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).
		5	Core	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).
6	Core	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).		

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7	Core	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to clear Ovf_FixedCtr0.
		33	Thread	Set 1 to clear Ovf_FixedCtr1.
		34	Thread	Set 1 to clear Ovf_FixedCtr2.
		60:35		Reserved
		61	Thread	Set 1 to clear Ovf_Uncore.
		62	Thread	Set 1 to clear Ovf_BufDSSAVE.
		63	Thread	Set 1 to clear CondChgd.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 20.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		62:36		Reserved
		63		Enable Precise Store (R/W)
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 20.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
3FEH	1022	MSR_CORE_C7_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C7 Residency Counter (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.
400H	1024	IA32_MCO_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
402H	1026	IA32_MCO_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
403H	1027	IA32_MCO_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
406H	1030	IA32_MC1_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
407H	1031	IA32_MC1_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
409H	1033	IA32_MC2_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 17.
40AH	1034	IA32_MC2_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
40BH	1035	IA32_MC2_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 17.
40EH	1038	IA32_MC3_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
40FH	1039	IA32_MC3_MISC	Core	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
		0		PCU Hardware Error (R/W) When set, enables signaling of PCU hardware detected errors.
		1		PCU Controller Error (R/W) When set, enables signaling of PCU controller detected errors.
		2		PCU Firmware Error (R/W) When set, enables signaling of PCU firmware detected errors.
		63:2		Reserved
411H	1041	IA32_MC4_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 17.
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Thread	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Thread	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Thread	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Thread	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLSS2	Thread	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Thread	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLSS	Thread	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLSS	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLSS	Thread	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLSS	Thread	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 2-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 2-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 2-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 2-2.
4C8H	1224	IA32_A_PMC7	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."
60AH	1546	MSR_PKGC3_IRTL	Package	Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60BH	1547	MSR_PKG_C6_IRTL	Package	Package C6 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C6 to a C0 state, where an interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W) See Section 15.10.3, "Package RAPL Domain."
638H	1592	MSR_PPO_POWER_LIMIT	Package	PPO RAPL Power Limit Control (R/W) See Section 15.10.4, "PPO/PP1 RAPL Domains."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.9.1 and record format in Section 18.4.8.1.
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table 2-2.
802H-83FH	2050-2111	X2APIC MSRs	Thread	See Table 2-2.
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 18.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

2.11.1 MSRs in the 2nd Generation Intel® Core™ Processor Family Based on Sandy Bridge Microarchitecture

Table 2-21 and Table 2-22 list model-specific registers (MSRs) that are specific to the 2nd generation Intel® Core™ processor family based on the Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH; see Table 2-1.

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60CH	1548	MSR_PKG_C7_IRTL	Package	Package C7 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."
63AH	1594	MSR_PP0_POLICY	Package	PP0 Balance Policy (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	PP1 RAPL Power Limit Control (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	PP1 Balance Policy (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."
See Table 2-20, Table 2-21, and Table 2-22 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.				

Table 2-22 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4 select.
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
	63:32		Reserved	
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Report the number of C-Box units with performance counters, including processor cores and processor graphics.
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, Counter 1 Event Select MSR

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
702H	1794	MSR_UNC_CBO_0_PERFEVTSEL2	Package	Uncore C-Box 0, Counter 2 Event Select MSR
703H	1795	MSR_UNC_CBO_0_PERFEVTSEL3	Package	Uncore C-Box 0, Counter 3 Event Select MSR
705H	1797	MSR_UNC_CBO_0_UNIT_STATUS	Package	Uncore C-Box 0, Unit Status for Counter 0-3
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
708H	1800	MSR_UNC_CBO_0_PERFCTR2	Package	Uncore C-Box 0, Performance Counter 2
709H	1801	MSR_UNC_CBO_0_PERFCTR3	Package	Uncore C-Box 0, Performance Counter 3
710H	1808	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
712H	1810	MSR_UNC_CBO_1_PERFEVTSEL2	Package	Uncore C-Box 1, Counter 2 Event Select MSR
713H	1811	MSR_UNC_CBO_1_PERFEVTSEL3	Package	Uncore C-Box 1, Counter 3 Event Select MSR
715H	1813	MSR_UNC_CBO_1_UNIT_STATUS	Package	Uncore C-Box 1, Unit Status for Counter 0-3
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
718H	1816	MSR_UNC_CBO_1_PERFCTR2	Package	Uncore C-Box 1, Performance Counter 2
719H	1817	MSR_UNC_CBO_1_PERFCTR3	Package	Uncore C-Box 1, Performance Counter 3
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
722H	1826	MSR_UNC_CBO_2_PERFEVTSEL2	Package	Uncore C-Box 2, Counter 2 Event Select MSR
723H	1827	MSR_UNC_CBO_2_PERFEVTSEL3	Package	Uncore C-Box 2, Counter 3 Event Select MSR
725H	1829	MSR_UNC_CBO_2_UNIT_STATUS	Package	Uncore C-Box 2, Unit Status for Counter 0-3
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
728H	1832	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, Performance Counter 2
729H	1833	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, Performance Counter 3
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
732H	1842	MSR_UNC_CBO_3_PERFEVTSEL2	Package	Uncore C-Box 3, Counter 2 Event Select MSR
733H	1843	MSR_UNC_CBO_3_PERFEVTSEL3	Package	Uncore C-Box 3, counter 3 Event Select MSR
735H	1845	MSR_UNC_CBO_3_UNIT_STATUS	Package	Uncore C-Box 3, Unit Status for Counter 0-3
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
738H	1848	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, Performance Counter 2
739H	1849	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, Performance Counter 3
740H	1856	MSR_UNC_CBO_4_PERFEVTSELO	Package	Uncore C-Box 4, Counter 0 Event Select MSR

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
741H	1857	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
742H	1858	MSR_UNC_CBO_4_PERFEVTSEL2	Package	Uncore C-Box 4, Counter 2 Event Select MSR
743H	1859	MSR_UNC_CBO_4_PERFEVTSEL3	Package	Uncore C-Box 4, Counter 3 Event Select MSR
745H	1861	MSR_UNC_CBO_4_UNIT_STATUS	Package	Uncore C-Box 4, Unit status for Counter 0-3
746H	1862	MSR_UNC_CBO_4_PERFCTRO	Package	Uncore C-Box 4, Performance Counter 0
747H	1863	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
748H	1864	MSR_UNC_CBO_4_PERFCTR2	Package	Uncore C-Box 4, Performance Counter 2
749H	1865	MSR_UNC_CBO_4_PERFCTR3	Package	Uncore C-Box 4, Performance Counter 3

2.11.2 MSRs in the Intel® Xeon® Processor E5 Family Based on Sandy Bridge Microarchitecture

Table 2-23 lists additional model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH, and also support MSRs listed in Table 2-20 and Table 2-24.

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, R/W if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 cores active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 cores active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 cores active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 cores active.

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 cores active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 cores active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 cores active.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
39CH	924	MSR_PEBS_NUM_ALT	Package	ENABLE_PEBS_NUM_ALT (R/W)
		0		ENABLE_PEBS_NUM_ALT (R/W) Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see https://perfmon-events.intel.com/ .
		63:1		Reserved, must be zero.
414H	1044	IA32_MC5_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
416H	1046	IA32_MC5_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
41AH	1050	IA32_MC6_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	IA32_MC6_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
41DH	1053	IA32_MC7_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
41EH	1054	IA32_MC7_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	IA32_MC7_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
422H	1058	IA32_MC8_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	IA32_MC9_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	IA32_MC9_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
426H	1062	IA32_MC9_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	IA32_MC9_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	IA32_MC10_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	IA32_MC10_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
42AH	1066	IA32_MC10_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	IA32_MC10_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	IA32_MC11_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	IA32_MC11_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
42EH	1070	IA32_MC11_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	IA32_MC11_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	IA32_MC12_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	IA32_MC12_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
432H	1074	IA32_MC12_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	IA32_MC12_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	IA32_MC13_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	IA32_MC13_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
436H	1078	IA32_MC13_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	IA32_MC13_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	IA32_MC14_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	IA32_MC14_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
43AH	1082	IA32_MC14_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	IA32_MC14_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
43CH	1084	IA32_MC15_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	IA32_MC15_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
43EH	1086	IA32_MC15_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	IA32_MC15_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	IA32_MC16_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	IA32_MC16_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
442H	1090	IA32_MC16_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	IA32_MC16_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	IA32_MC17_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
446H	1094	IA32_MC17_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
44AH	1098	IA32_MC18_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.
44EH	1102	IA32_MC19_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 16.3.2.4, "IA32_MCi_MISC MSRs."
613H	1555	MSR_PKG_PERF_STATUS	Package	Package RAPL Perf Status (R/O)
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 15.10.4, "PPO/PP1 RAPL Domains."

See Table 2-20, Table 2-23, and Table 2-24 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.

2.11.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 Family

Intel Xeon Processor E5 family is based on the Sandy Bridge microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C08H	3080	MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-box UCLK Fixed Counter Control
C09H	3081	MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-box UCLK Fixed Counter
C10H	3088	MSR_U_PMON_EVNTSELO	Package	Uncore U-box Perfmon Event Select for U-box Counter 0
C11H	3089	MSR_U_PMON_EVNTSEL1	Package	Uncore U-box Perfmon Event Select for U-box Counter 1
C16H	3094	MSR_U_PMON_CTR0	Package	Uncore U-box Perfmon Counter 0
C17H	3095	MSR_U_PMON_CTR1	Package	Uncore U-box Perfmon Counter 1
C24H	3108	MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU Perfmon for PCU-box-wide Control
C30H	3120	MSR_PCU_PMON_EVNTSELO	Package	Uncore PCU Perfmon Event Select for PCU Counter 0
C31H	3121	MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU Perfmon Event Select for PCU Counter 1
C32H	3122	MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU Perfmon Event Select for PCU Counter 2
C33H	3123	MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU Perfmon Event Select for PCU Counter 3
C34H	3124	MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU Perfmon box-wide Filter
C36H	3126	MSR_PCU_PMON_CTR0	Package	Uncore PCU Perfmon Counter 0
C37H	3127	MSR_PCU_PMON_CTR1	Package	Uncore PCU Perfmon Counter 1
C38H	3128	MSR_PCU_PMON_CTR2	Package	Uncore PCU Perfmon Counter 2
C39H	3129	MSR_PCU_PMON_CTR3	Package	Uncore PCU Perfmon Counter 3
D04H	3332	MSR_CO_PMON_BOX_CTL	Package	Uncore C-box 0 Perfmon Local Box Wide Control
D10H	3344	MSR_CO_PMON_EVNTSELO	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 0
D11H	3345	MSR_CO_PMON_EVNTSEL1	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 1
D12H	3346	MSR_CO_PMON_EVNTSEL2	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 2
D13H	3347	MSR_CO_PMON_EVNTSEL3	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 3
D14H	3348	MSR_CO_PMON_BOX_FILTER	Package	Uncore C-box 0 Perfmon Box Wide Filter
D16H	3350	MSR_CO_PMON_CTR0	Package	Uncore C-box 0 Perfmon Counter 0
D17H	3351	MSR_CO_PMON_CTR1	Package	Uncore C-box 0 Perfmon Counter 1
D18H	3352	MSR_CO_PMON_CTR2	Package	Uncore C-box 0 Perfmon Counter 2
D19H	3353	MSR_CO_PMON_CTR3	Package	Uncore C-box 0 Perfmon Counter 3
D24H	3364	MSR_C1_PMON_BOX_CTL	Package	Uncore C-box 1 Perfmon Local Box Wide Control

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D30H	3376	MSR_C1_PMON_EVNTSEL0	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 0
D31H	3377	MSR_C1_PMON_EVNTSEL1	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 1
D32H	3378	MSR_C1_PMON_EVNTSEL2	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 2
D33H	3379	MSR_C1_PMON_EVNTSEL3	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 3
D34H	3380	MSR_C1_PMON_BOX_FILTER	Package	Uncore C-box 1 Perfmon Box Wide Filter
D36H	3382	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 Perfmon Counter 0
D37H	3383	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 Perfmon Counter 1
D38H	3384	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 Perfmon Counter 2
D39H	3385	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 Perfmon Counter 3
D44H	3396	MSR_C2_PMON_BOX_CTL	Package	Uncore C-box 2 Perfmon Local Box Wide Control
D50H	3408	MSR_C2_PMON_EVNTSEL0	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 0
D51H	3409	MSR_C2_PMON_EVNTSEL1	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 1
D52H	3410	MSR_C2_PMON_EVNTSEL2	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 2
D53H	3411	MSR_C2_PMON_EVNTSEL3	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 3
D54H	3412	MSR_C2_PMON_BOX_FILTER	Package	Uncore C-box 2 Perfmon Box Wide Filter
D56H	3414	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 Perfmon Counter 0
D57H	3415	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 Perfmon Counter 1
D58H	3416	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 Perfmon Counter 2
D59H	3417	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 Perfmon Counter 3
D64H	3428	MSR_C3_PMON_BOX_CTL	Package	Uncore C-box 3 Perfmon Local Box Wide Control
D70H	3440	MSR_C3_PMON_EVNTSEL0	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 0
D71H	3441	MSR_C3_PMON_EVNTSEL1	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 1
D72H	3442	MSR_C3_PMON_EVNTSEL2	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 2
D73H	3443	MSR_C3_PMON_EVNTSEL3	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 3
D74H	3444	MSR_C3_PMON_BOX_FILTER	Package	Uncore C-box 3 Perfmon Box Wide Filter
D76H	3446	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 Perfmon Counter 0
D77H	3447	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 Perfmon Counter 1
D78H	3448	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 Perfmon Counter 2
D79H	3449	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 Perfmon Counter 3

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D84H	3460	MSR_C4_PMON_BOX_CTL	Package	Uncore C-box 4 Perfmon Local Box Wide Control
D90H	3472	MSR_C4_PMON_EVNTSEL0	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 0
D91H	3473	MSR_C4_PMON_EVNTSEL1	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 1
D92H	3474	MSR_C4_PMON_EVNTSEL2	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 2
D93H	3475	MSR_C4_PMON_EVNTSEL3	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 3
D94H	3476	MSR_C4_PMON_BOX_FILTER	Package	Uncore C-box 4 Perfmon Box Wide Filter
D96H	3478	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 Perfmon Counter 0
D97H	3479	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 Perfmon Counter 1
D98H	3480	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 Perfmon Counter 2
D99H	3481	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 Perfmon Counter 3
DA4H	3492	MSR_C5_PMON_BOX_CTL	Package	Uncore C-box 5 Perfmon Local Box Wide Control
DB0H	3504	MSR_C5_PMON_EVNTSEL0	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 0
DB1H	3505	MSR_C5_PMON_EVNTSEL1	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 1
DB2H	3506	MSR_C5_PMON_EVNTSEL2	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 2
DB3H	3507	MSR_C5_PMON_EVNTSEL3	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 3
DB4H	3508	MSR_C5_PMON_BOX_FILTER	Package	Uncore C-box 5 Perfmon Box Wide Filter
DB6H	3510	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 Perfmon Counter 0
DB7H	3511	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 Perfmon Counter 1
DB8H	3512	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 Perfmon Counter 2
DB9H	3513	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 Perfmon Counter 3
DC4H	3524	MSR_C6_PMON_BOX_CTL	Package	Uncore C-box 6 Perfmon Local Box Wide Control
DD0H	3536	MSR_C6_PMON_EVNTSEL0	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 0
DD1H	3537	MSR_C6_PMON_EVNTSEL1	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 1
DD2H	3538	MSR_C6_PMON_EVNTSEL2	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 2
DD3H	3539	MSR_C6_PMON_EVNTSEL3	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 3
DD4H	3540	MSR_C6_PMON_BOX_FILTER	Package	Uncore C-box 6 Perfmon Box Wide Filter
DD6H	3542	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 Perfmon Counter 0
DD7H	3543	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 Perfmon Counter 1
DD8H	3544	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 Perfmon Counter 2

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD9H	3545	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 Perfmon Counter 3
DE4H	3556	MSR_C7_PMON_BOX_CTL	Package	Uncore C-box 7 Perfmon Local Box Wide Control
DF0H	3568	MSR_C7_PMON_EVNTSEL0	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 0
DF1H	3569	MSR_C7_PMON_EVNTSEL1	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 1
DF2H	3570	MSR_C7_PMON_EVNTSEL2	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 2
DF3H	3571	MSR_C7_PMON_EVNTSEL3	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 3
DF4H	3572	MSR_C7_PMON_BOX_FILTER	Package	Uncore C-box 7 Perfmon Box Wide Filter
DF6H	3574	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 Perfmon Counter 0
DF7H	3575	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 Perfmon Counter 1
DF8H	3576	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 Perfmon Counter 2
DF9H	3577	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 Perfmon Counter 3

2.12 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY BASED ON IVY BRIDGE MICROARCHITECTURE

The 3rd generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v2 product family based on Ivy Bridge microarchitecture support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-25. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates that TDP Limit for Turbo mode is not programmable.
		31:30		Reserved
		32	Package	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.
		34:33	Package	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved
		39:35		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		55:48	Package	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.
		28		Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.
		63:29		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O)
		7:0		Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O)
		14:0		PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.
		15		Reserved
		23:16		Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		47		Reserved
		62:48		PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O)
		14:0		PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W)
		1:0		TDP_LEVEL (RW/L) System BIOS can program this field.
		30:2		Reserved.
		31		Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		7:0		MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.
		30:8		Reserved
		31		TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
See Table 2-20, Table 2-21, and Table 2-22 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.				

2.12.1 MSRs in the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture

Table 2-26 lists model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 v2 Product Family (based on Ivy Bridge-E microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH; see Table 2-1. These processors supports the MSR interfaces listed in Table 2-20 and Table 2-26.

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/W) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		22:16		Reserved
		23	Package	PPIN_CAP (R/O) When set to 1, indicates that Protected Processor Inventory Number (PPIN) capability can be enabled for a privileged system inventory agent to read PPIN from MSR_PPIN. When set to 0, PPIN capability is not supported. An attempt to access MSR_PPIN_CTL or MSR_PPIN will cause #GP.
		27:24		Reserved
	28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		30	Package	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:16		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		Reserved
		26		MCG_ELOG_P
		63:27		Reserved
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R/O) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.
		27:24		TCC Activation Offset (R/W) Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO.[30] is set.
		63:28		Reserved
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		63:32		Reserved
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
296H	662	IA32_MC22_CTL2	Package	See Table 2-2.
297H	663	IA32_MC23_CTL2	Package	See Table 2-2.
298H	664	IA32_MC24_CTL2	Package	See Table 2-2.
299H	665	IA32_MC25_CTL2	Package	See Table 2-2.
29AH	666	IA32_MC26_CTL2	Package	See Table 2-2.
29BH	667	IA32_MC27_CTL2	Package	See Table 2-2.
29CH	668	IA32_MC28_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
43CH	1084	IA32_MC15_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
458H	1112	IA32_MC22_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
459H	1113	IA32_MC22_STATUS	Package	
45AH	1114	IA32_MC22_ADDR	Package	
45BH	1115	IA32_MC22_MISC	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
45CH	1116	IA32_MC23_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
45DH	1117	IA32_MC23_STATUS	Package	
45EH	1118	IA32_MC23_ADDR	Package	
45FH	1119	IA32_MC23_MISC	Package	
460H	1120	IA32_MC24_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
461H	1121	IA32_MC24_STATUS	Package	
462H	1122	IA32_MC24_ADDR	Package	
463H	1123	IA32_MC24_MISC	Package	
464H	1124	IA32_MC25_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
465H	1125	IA32_MC25_STATUS	Package	
466H	1126	IA32_MC25_ADDR	Package	
467H	1127	IA32_MC25_MISC	Package	
468H	1128	IA32_MC26_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
469H	1129	IA32_MC26_STATUS	Package	
46AH	1130	IA32_MC26_ADDR	Package	
46BH	1131	IA32_MC26_MISC	Package	
46CH	1132	IA32_MC27_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
46DH	1133	IA32_MC27_STATUS	Package	
46EH	1134	IA32_MC27_ADDR	Package	
46FH	1135	IA32_MC27_MISC	Package	
470H	1136	IA32_MC28_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
471H	1137	IA32_MC28_STATUS	Package	
472H	1138	IA32_MC28_ADDR	Package	
473H	1139	IA32_MC28_MISC	Package	
613H	1555	MSR_PKG_PERF_STATUS	Package	Package RAPL Perf Status (R/O)
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 15.10.4, "PPO/PP1 RAPL Domains."

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
See Table 2-20, for other MSR definitions applicable to Intel Xeon processor E5 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.				

2.12.2 Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family

The Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH supports the MSR interfaces listed in Table 2-20, Table 2-26, and Table 2-27.

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)
		63:16		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		63:25		Reserved
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status (R/WO)
		0		RIPV
		1		EIPV
		2		MCIP
		3		LMCE Signaled
		63:4		Reserved

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10core active.
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		39:32	Package	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.
		47:40	Package	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.
		55:48	Package	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.
		62:56		Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
29DH	669	IA32_MC29_CTL2	Package	See Table 2-2.
29EH	670	IA32_MC30_CTL2	Package	See Table 2-2.
29FH	671	IA32_MC31_CTL2	Package	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 20.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		$n:0$		Enable PEBS on IA32_PMCx. (R/W)
		$31:n+1$		Reserved
		$32+m:32$		Enable Load Latency on IA32_PMCx. (R/W)
		$63:33+m$		Reserved
41BH	1051	IA32_MC6_MISC	Package	Misc MAC Information of Integrated I/O (R/O) See Section 16.3.2.4.
		5:0		Recoverable Address LSB
		8:6		Address Mode
		15:9		Reserved
		31:16		PCI Express Requestor ID

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32		PCI Express Segment Number
		63:32		Reserved
474H	1140	IA32_MC29_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
475H	1141	IA32_MC29_STATUS	Package	
476H	1142	IA32_MC29_ADDR	Package	
477H	1143	IA32_MC29_MISC	Package	
478H	1144	IA32_MC30_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
479H	1145	IA32_MC30_STATUS	Package	
47AH	1146	IA32_MC30_ADDR	Package	
47BH	1147	IA32_MC30_MISC	Package	
47CH	1148	IA32_MC31_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
47DH	1149	IA32_MC31_STATUS	Package	
47EH	1150	IA32_MC31_ADDR	Package	
47FH	1147	IA32_MC31_MISC	Package	
See Table 2-20, Table 2-26 for other MSR definitions applicable to Intel Xeon processor E7 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.				

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.12.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Intel Xeon Processor E5 v2 and E7 v2 families are based on the Ivy Bridge-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24 and Table 2-28. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 v2 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C00H	3072	MSR_PMON_GLOBAL_CTL	Package	Uncore Perfmon Per-Socket Global Control
C01H	3073	MSR_PMON_GLOBAL_STATUS	Package	Uncore Perfmon Per-Socket Global Status
C06H	3078	MSR_PMON_GLOBAL_CONFIG	Package	Uncore Perfmon Per-Socket Global Configuration
C15H	3093	MSR_U_PMON_BOX_STATUS	Package	Uncore U-box Perfmon U-Box Wide Status
C35H	3125	MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU Perfmon Box Wide Status
D1AH	3354	MSR_C0_PMON_BOX_FILTER1	Package	Uncore C-Box 0 Perfmon Box Wide Filter1
D3AH	3386	MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-Box 1 Perfmon Box Wide Filter1

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D5AH	3418	MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-Box 2 Perfmon Box Wide Filter1
D7AH	3450	MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-Box 3 Perfmon Box Wide Filter1
D9AH	3482	MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-Box 4 Perfmon Box Wide Filter1
DBAH	3514	MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-Box 5 Perfmon Box Wide Filter1
DDAH	3546	MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-Box 6 Perfmon Box Wide Filter1
DFAH	3578	MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-Box 7 Perfmon Box Wide Filter1
E04H	3588	MSR_C8_PMON_BOX_CTL	Package	Uncore C-Box 8 Perfmon Local Box Wide Control
E10H	3600	MSR_C8_PMON_EVNTSELO	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0
E11H	3601	MSR_C8_PMON_EVNTSEL1	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1
E12H	3602	MSR_C8_PMON_EVNTSEL2	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2
E13H	3603	MSR_C8_PMON_EVNTSEL3	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3
E14H	3604	MSR_C8_PMON_BOX_FILTER	Package	Uncore C-Box 8 Perfmon Box Wide Filter
E16H	3606	MSR_C8_PMON_CTR0	Package	Uncore C-Box 8 Perfmon Counter 0
E17H	3607	MSR_C8_PMON_CTR1	Package	Uncore C-Box 8 Perfmon Counter 1
E18H	3608	MSR_C8_PMON_CTR2	Package	Uncore C-Box 8 Perfmon Counter 2
E19H	3609	MSR_C8_PMON_CTR3	Package	Uncore C-Box 8 Perfmon Counter 3
E1AH	3610	MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-Box 8 Perfmon Box Wide Filter1
E24H	3620	MSR_C9_PMON_BOX_CTL	Package	Uncore C-Box 9 Perfmon Local Box Wide Control
E30H	3632	MSR_C9_PMON_EVNTSELO	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 0
E31H	3633	MSR_C9_PMON_EVNTSEL1	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 1
E32H	3634	MSR_C9_PMON_EVNTSEL2	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 2
E33H	3635	MSR_C9_PMON_EVNTSEL3	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 3
E34H	3636	MSR_C9_PMON_BOX_FILTER	Package	Uncore C-Box 9 Perfmon Box Wide Filter
E36H	3638	MSR_C9_PMON_CTR0	Package	Uncore C-Box 9 Perfmon Counter 0
E37H	3639	MSR_C9_PMON_CTR1	Package	Uncore C-Box 9 Perfmon Counter 1
E38H	3640	MSR_C9_PMON_CTR2	Package	Uncore C-Box 9 Perfmon Counter 2
E39H	3641	MSR_C9_PMON_CTR3	Package	Uncore C-Box 9 Perfmon Counter 3
E3AH	3642	MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-Box 9 Perfmon Box Wide Filter1
E44H	3652	MSR_C10_PMON_BOX_CTL	Package	Uncore C-Box 10 Perfmon Local Box Wide Control
E50H	3664	MSR_C10_PMON_EVNTSELO	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E51H	3665	MSR_C10_PMON_EVNTSEL1	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1
E52H	3666	MSR_C10_PMON_EVNTSEL2	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2
E53H	3667	MSR_C10_PMON_EVNTSEL3	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3
E54H	3668	MSR_C10_PMON_BOX_FILTER	Package	Uncore C-Box 10 Perfmon Box Wide Filter
E56H	3670	MSR_C10_PMON_CTR0	Package	Uncore C-Box 10 Perfmon Counter 0
E57H	3671	MSR_C10_PMON_CTR1	Package	Uncore C-Box 10 Perfmon Counter 1
E58H	3672	MSR_C10_PMON_CTR2	Package	Uncore C-Box 10 Perfmon Counter 2
E59H	3673	MSR_C10_PMON_CTR3	Package	Uncore C-Box 10 Perfmon Counter 3
E5AH	3674	MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-Box 10 Perfmon Box Wide Filter1
E64H	3684	MSR_C11_PMON_BOX_CTL	Package	Uncore C-Box 11 Perfmon Local Box Wide Control
E70H	3696	MSR_C11_PMON_EVNTSELO	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0
E71H	3697	MSR_C11_PMON_EVNTSEL1	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1
E72H	3698	MSR_C11_PMON_EVNTSEL2	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2
E73H	3699	MSR_C11_PMON_EVNTSEL3	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 3
E74H	3700	MSR_C11_PMON_BOX_FILTER	Package	Uncore C-Box 11 Perfmon Box Wide Filter
E76H	3702	MSR_C11_PMON_CTR0	Package	Uncore C-Box 11 Perfmon Counter 0
E77H	3703	MSR_C11_PMON_CTR1	Package	Uncore C-Box 11 Perfmon Counter 1
E78H	3704	MSR_C11_PMON_CTR2	Package	Uncore C-Box 11 Perfmon Counter 2
E79H	3705	MSR_C11_PMON_CTR3	Package	Uncore C-Box 11 Perfmon Counter 3
E7AH	3706	MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-Box 11 Perfmon Box Wide Filter1
E84H	3716	MSR_C12_PMON_BOX_CTL	Package	Uncore C-Box 12 Perfmon Local Box Wide Control
E90H	3728	MSR_C12_PMON_EVNTSELO	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0
E91H	3729	MSR_C12_PMON_EVNTSEL1	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1
E92H	3730	MSR_C12_PMON_EVNTSEL2	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2
E93H	3731	MSR_C12_PMON_EVNTSEL3	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3
E94H	3732	MSR_C12_PMON_BOX_FILTER	Package	Uncore C-Box 12 Perfmon Box Wide Filter
E96H	3734	MSR_C12_PMON_CTR0	Package	Uncore C-Box 12 Perfmon Counter 0
E97H	3735	MSR_C12_PMON_CTR1	Package	Uncore C-Box 12 Perfmon Counter 1
E98H	3736	MSR_C12_PMON_CTR2	Package	Uncore C-Box 12 Perfmon Counter 2

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E99H	3737	MSR_C12_PMON_CTR3	Package	Uncore C-Box 12 Perfmon Counter 3
E9AH	3738	MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-Box 12 Perfmon Box Wide Filter1
EA4H	3748	MSR_C13_PMON_BOX_CTL	Package	Uncore C-Box 13 Perfmon Local Box Wide Control
EBOH	3760	MSR_C13_PMON_EVNTSELO	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0
EB1H	3761	MSR_C13_PMON_EVNTSEL1	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1
EB2H	3762	MSR_C13_PMON_EVNTSEL2	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2
EB3H	3763	MSR_C13_PMON_EVNTSEL3	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3
EB4H	3764	MSR_C13_PMON_BOX_FILTER	Package	Uncore C-Box 13 Perfmon Box Wide Filter
EB6H	3766	MSR_C13_PMON_CTR0	Package	Uncore C-Box 13 Perfmon Counter 0
EB7H	3767	MSR_C13_PMON_CTR1	Package	Uncore C-Box 13 Perfmon Counter 1
EB8H	3768	MSR_C13_PMON_CTR2	Package	Uncore C-Box 13 Perfmon Counter 2
EB9H	3769	MSR_C13_PMON_CTR3	Package	Uncore C-Box 13 Perfmon Counter 3
EBAH	3770	MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-Box 13 Perfmon Box Wide Filter1
EC4H	3780	MSR_C14_PMON_BOX_CTL	Package	Uncore C-Box 14 Perfmon Local Box Wide Control
EDOH	3792	MSR_C14_PMON_EVNTSELO	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0
ED1H	3793	MSR_C14_PMON_EVNTSEL1	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1
ED2H	3794	MSR_C14_PMON_EVNTSEL2	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2
ED3H	3795	MSR_C14_PMON_EVNTSEL3	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3
ED4H	3796	MSR_C14_PMON_BOX_FILTER	Package	Uncore C-Box 14 Perfmon Box Wide Filter
ED6H	3798	MSR_C14_PMON_CTR0	Package	Uncore C-Box 14 Perfmon Counter 0
ED7H	3799	MSR_C14_PMON_CTR1	Package	Uncore C-Box 14 Perfmon Counter 1
ED8H	3800	MSR_C14_PMON_CTR2	Package	Uncore C-Box 14 Perfmon Counter 2
ED9H	3801	MSR_C14_PMON_CTR3	Package	Uncore C-Box 14 Perfmon Counter 3
EDAH	3802	MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-Box 14 Perfmon Box Wide Filter1

2.13 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS BASED ON HASWELL MICROARCHITECTURE

The 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H, support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-29. For an MSR listed in Table 2-20 that also appears in Table 2-29, Table 2-29 supersedes Table 2-20.

The MSRs listed in Table 2-29 also apply to processors based on Haswell-E microarchitecture (see Section 2.14).

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3BH	59	IA32_TSC_ADJUST	Thread	Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		31:30		Reserved
		32	Package	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.
		34:33	Package	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved
		39:35		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		55:48	Package	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved
186H	390	IA32_PERFVTSELO	Thread	Performance Event Select for Counter 0 (R/W) Supports all fields described in Table 2-2 and the fields below.

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		32		IN_TX: See Section 20.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
187H	391	IA32_PERFEVTSEL1	Thread	Performance Event Select for Counter 1 (R/W) Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: See Section 20.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
188H	392	IA32_PERFEVTSEL2	Thread	Performance Event Select for Counter 2 (R/W) Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: See Section 20.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
		33		IN_TXCP: See Section 20.3.6.5.1. When IN_TXCP=1 & IN_TX=1 and in sampling, a spurious PMI may occur and transactions may continuously abort near overflow conditions. Software should favor using IN_TXCP for counting over sampling. If sampling, software should use large "sample-after" value after clearing the counter configured to use IN_TXCP and also always reset the counter even when no overflow condition was reported.
189H	393	IA32_PERFEVTSEL3	Thread	Performance Event Select for Counter 3 (R/W) Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: See Section 20.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W)
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
9		EN_CALL_STACK		

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:9		Reserved
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.
		0		LBR: Last Branch Record
		1		BTF
		5:2		Reserved
		6		TR: Branch Trace
		7		BTS: Log Branch Trace Message to BTS Buffer
		8		BTINT
		9		BTS_OFF_OS
		10		BTS_OFF_USER
		11		FREEZE_LBR_ON_PMI
		12		FREEZE_PERFMON_ON_PMI
		13		ENABLE_UNCORE_PMI
		14		FREEZE_WHILE_SMM
		15		RTM_DEBUG
		63:15		Reserved
491H	1169	IA32_VMX_VMFUNC	Thread	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.
60BH	1548	MSR_PKG_C7_IRTL1	Package	Package C6/C7 Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the shorter-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
60CH	1548	MSR_PKGC_IRTL2	Package	Package C6/C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the longer-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt response time limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 15.10.3, "Package RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O)
		7:0		Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 Ratio and Power Level (R/O)
		14:0		PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15		Reserved
		23:16		Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.
		62:47		PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 Ratio and Power Level (R/O)
		14:0		PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.
		62:47		PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W)
		1:0		TDP_LEVEL (RW/L) System BIOS can program this field.
		30:2		Reserved
		31		Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		7:0		MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.
		30:8		Reserved
		31		TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:32		Reserved
C80H	3200	IA32_DEBUG_INTERFACE	Package	Silicon Debug Feature Control (R/W) See Table 2-2.

2.13.1 MSRs in the 4th Generation Intel® Core™ Processor Family Based on Haswell Microarchitecture

Table 2-30 lists model-specific registers (MSRs) that are specific to the 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H; see Table 2-1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		3:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s Package C states C7 are not available to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH.
		9:4		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/WO)
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W)

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		63:29		Reserved
17DH	381	MSR_SMM_MCA_CAP	THREAD	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Core 0 select.
		1		Core 1 select.
		2		Core 2 select.
		3		Core 3 select.
		18:4		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Encoded number of C-Box, derive value by "-1".
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Core 0 select.
		1		Core 1 select.
		2		Core 2 select.
		3		Core 3 select.
		18:4		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
63:32		Reserved		
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.
		0		Lock (SMM-RWO) When set to '1' locks this register from further changes.
		1		Reserved
		2		SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is OFFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:N		Reserved
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 15.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{ESU}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 15.10.1, "RAPL Interfaces."
		63:20		Reserved
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 15.10.4, "PPO/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	PP1 RAPL Power Limit Control (R/W) See Section 15.10.4, "PPO/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 15.10.4, "PPO/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	PP1 Balance Policy (R/W) See Section 15.10.4, "PPO/PP1 RAPL Domains."
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved
		4		Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		9		Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		12		Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.
		13		Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		15:14		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		20		Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)
		0		PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved
		4		Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		9		Graphics Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		11		Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved
		20		Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)
		0		PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		5:2		Reserved
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9		Reserved
		10		Package-Level Power Limiting PL1 Status (RO) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		Package-Level PL2 Power Limiting Status (RO) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1824	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
736H	1846	MSR_UNC_CBO_3_PERFCTRO	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1

See Table 2-20, Table 2-21, Table 2-22, Table 2-25, and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 063CH or 06_46H.

2.13.2 Additional Residency MSRs Supported in 4th Generation Intel® Core™ Processors

The 4th generation Intel® Core™ processor family (based on Haswell microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-30, and Table 2-31.

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		3:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10
		9:4		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/W0)

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		63:29		Reserved
630H	1584	MSR_PKG_C8_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C8 Residency Counter (R/O) Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.
		63:60		Reserved
631H	1585	MSR_PKG_C9_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C9 Residency Counter (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.
		63:60		Reserved
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C10 Residency Counter (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.
		63:60		Reserved

See Table 2-20, Table 2-21, Table 2-22, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.

2.14 MSRS IN THE INTEL® XEON® PROCESSOR E5 V3 AND E7 V3 PRODUCT FAMILY

The Intel® Xeon® processor E5 v3 family and the Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_3F). These processors support the MSR interfaces listed in Table 2-20, Table 2-29, and Table 2-32.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
35H	53	MSR_CORE_THREAD_COUNT	Package	Configured State of Enabled Processor Core Count and Logical Processor Count (R/O) <ul style="list-style-type: none"> After a Power-On RESET, enumerates factory configuration of the number of processor cores and logical processors in the physical package. Following the sequence of (i) BIOS modified a Configuration Mask which selects a subset of processor cores to be active post RESET and (ii) a RESET event after the modification, enumerates the current configuration of enabled processor core count and logical processor count in the physical package.
		15:0		THREAD_COUNT (R/O) The number of logical processors that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		31:16		Core_COUNT (R/O) The number of processor cores that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		63:32		Reserved
53H	83	MSR_THREAD_ID_INFO	Thread	A Hardware Assigned ID for the Logical Processor (R/O)
		7:0		Logical_Processor_ID (R/O) An implementation-specific numerical value physically assigned to each logical processor. This ID is not related to Initial APIC ID or x2APIC ID, it is unique within a physical package.
		63:8		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/W0)
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		MCG_EM_P
		26		MCG_ELOG_P
		63:27		Reserved
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		39:32	Package	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.
		47:40	Package	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.
		55:48	Package	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:56	Package	Maximum Ratio Limit for 16C Maximum turbo ratio limit of 16 core active.
1AFH	431	MSR_TURBO_RATIO_LIMIT2	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 17C Maximum turbo ratio limit of 17 core active.
		15:8	Package	Maximum Ratio Limit for 18C Maximum turbo ratio limit of 18 core active.
		62:16	Package	Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).
414H	1044	IA32_MC5_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
428H	1064	IA32_MC10_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
44CH	1100	IA32_MC19_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 15.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 15.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy Consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."
61EH	1566	MSR_PCIE_PLL_RATIO	Package	Configuration of PCIE PLL Relative to BCLK(R/W)

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1:0	Package	PCIE Ratio (R/W) 00b: Use 5:5 mapping for 100MHz operation (default). 01b: Use 5:4 mapping for 125MHz operation. 10b: Use 5:3 mapping for 166MHz operation. 11b: Use 5:2 mapping for 250MHz operation.
		2	Package	LPLL Select (R/W) If 1, use configured setting of PCIE Ratio.
		3	Package	LONG RESET (R/W) If 1, wait an additional time-out before re-locking Gen2/Gen3 PLLs.
		63:4		Reserved
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (RO) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (RO) When set, frequency is reduced below the operating system request due to a thermal event.
		2		Power Budget Management Status (RO) When set, frequency is reduced below the operating system request due to PBM limit
		3		Platform Configuration Services Status (RO) When set, frequency is reduced below the operating system request due to PCS limit
		4		Reserved

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		9		Reserved
		10		Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.
		12:11		Reserved
		13		Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.
		14		Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.
		15		Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		19		Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28:27		Reserved
		29		Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31		Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:32		Reserved
C8DH	3213	IA32_QM_EVTSEL	THREAD	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (R/W) Event encoding: 0x0: No monitoring. 0x1: L3 occupancy monitoring. All other encoding reserved.
		31:8		Reserved
		41:32		RMID (R/W)
		63:42		Reserved
C8EH	3214	IA32_QM_CTR	THREAD	Monitoring Counter Register (R/O) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		61:0		Resource Monitored Data
		62		Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.
		63		Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		63: 10		Reserved
See Table 2-20 and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.				

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.14.1 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

The Intel Xeon Processor E5 v3 and E7 v3 families are based on Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-33. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
700H	1792	MSR_PMON_GLOBAL_CTL	Package	Uncore Perfmon Per-Socket Global Control
701H	1793	MSR_PMON_GLOBAL_STATUS	Package	Uncore Perfmon Per-Socket Global Status
702H	1794	MSR_PMON_GLOBAL_CONFIG	Package	Uncore Perfmon Per-Socket Global Configuration
703H	1795	MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-Box UCLK Fixed Counter Control
704H	1796	MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-Box UCLK Fixed Counter
705H	1797	MSR_U_PMON_EVNTSELO	Package	Uncore U-Box Perfmon Event Select for U-Box Counter 0
706H	1798	MSR_U_PMON_EVNTSEL1	Package	Uncore U-Box Perfmon Event Select for U-Box Counter 1
708H	1800	MSR_U_PMON_BOX_STATUS	Package	Uncore U-Box Perfmon U-Box Wide Status
709H	1801	MSR_U_PMON_CTR0	Package	Uncore U-Box Perfmon Counter 0
70AH	1802	MSR_U_PMON_CTR1	Package	Uncore U-Box Perfmon Counter 1
710H	1808	MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU Perfmon for PCU-Box-Wide Control
711H	1809	MSR_PCU_PMON_EVNTSELO	Package	Uncore PCU Perfmon Event Select for PCU Counter 0
712H	1810	MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU Perfmon Event Select for PCU Counter 1
713H	1811	MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU Perfmon Event Select for PCU Counter 2
714H	1812	MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU Perfmon Event Select for PCU Counter 3
715H	1813	MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU Perfmon Box-Wide Filter
716H	1814	MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU Perfmon Box Wide Status
717H	1815	MSR_PCU_PMON_CTR0	Package	Uncore PCU Perfmon Counter 0
718H	1816	MSR_PCU_PMON_CTR1	Package	Uncore PCU Perfmon Counter 1
719H	1817	MSR_PCU_PMON_CTR2	Package	Uncore PCU Perfmon Counter 2
71AH	1818	MSR_PCU_PMON_CTR3	Package	Uncore PCU Perfmon Counter 3
720H	1824	MSR_S0_PMON_BOX_CTL	Package	Uncore SBo 0 Perfmon for SBo 0 Box-Wide Control
721H	1825	MSR_S0_PMON_EVNTSELO	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 0
722H	1826	MSR_S0_PMON_EVNTSEL1	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 1
723H	1827	MSR_S0_PMON_EVNTSEL2	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 2
724H	1828	MSR_S0_PMON_EVNTSEL3	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 3
725H	1829	MSR_S0_PMON_BOX_FILTER	Package	Uncore SBo 0 Perfmon Box-Wide Filter
726H	1830	MSR_S0_PMON_CTR0	Package	Uncore SBo 0 Perfmon Counter 0
727H	1831	MSR_S0_PMON_CTR1	Package	Uncore SBo 0 Perfmon Counter 1
728H	1832	MSR_S0_PMON_CTR2	Package	Uncore SBo 0 Perfmon Counter 2
729H	1833	MSR_S0_PMON_CTR3	Package	Uncore SBo 0 Perfmon Counter 3
72AH	1834	MSR_S1_PMON_BOX_CTL	Package	Uncore SBo 1 Perfmon for SBo 1 Box-Wide Control

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
72BH	1835	MSR_S1_PMON_EVNTSELO	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 0
72CH	1836	MSR_S1_PMON_EVNTSEL1	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 1
72DH	1837	MSR_S1_PMON_EVNTSEL2	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 2
72EH	1838	MSR_S1_PMON_EVNTSEL3	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 3
72FH	1839	MSR_S1_PMON_BOX_FILTER	Package	Uncore SBo 1 Perfmon Box-Wide Filter
730H	1840	MSR_S1_PMON_CTRL0	Package	Uncore SBo 1 Perfmon Counter 0
731H	1841	MSR_S1_PMON_CTRL1	Package	Uncore SBo 1 Perfmon Counter 1
732H	1842	MSR_S1_PMON_CTRL2	Package	Uncore SBo 1 Perfmon Counter 2
733H	1843	MSR_S1_PMON_CTRL3	Package	Uncore SBo 1 Perfmon Counter 3
734H	1844	MSR_S2_PMON_BOX_CTL	Package	Uncore SBo 2 Perfmon for SBo 2 Box-Wide Control
735H	1845	MSR_S2_PMON_EVNTSELO	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 0
736H	1846	MSR_S2_PMON_EVNTSEL1	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 1
737H	1847	MSR_S2_PMON_EVNTSEL2	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 2
738H	1848	MSR_S2_PMON_EVNTSEL3	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 3
739H	1849	MSR_S2_PMON_BOX_FILTER	Package	Uncore SBo 2 Perfmon Box-Wide Filter
73AH	1850	MSR_S2_PMON_CTRL0	Package	Uncore SBo 2 Perfmon Counter 0
73BH	1851	MSR_S2_PMON_CTRL1	Package	Uncore SBo 2 Perfmon Counter 1
73CH	1852	MSR_S2_PMON_CTRL2	Package	Uncore SBo 2 Perfmon Counter 2
73DH	1853	MSR_S2_PMON_CTRL3	Package	Uncore SBo 2 Perfmon Counter 3
73EH	1854	MSR_S3_PMON_BOX_CTL	Package	Uncore SBo 3 Perfmon for SBo 3 Box-Wide Control
73FH	1855	MSR_S3_PMON_EVNTSELO	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 0
740H	1856	MSR_S3_PMON_EVNTSEL1	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 1
741H	1857	MSR_S3_PMON_EVNTSEL2	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 2
742H	1858	MSR_S3_PMON_EVNTSEL3	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 3
743H	1859	MSR_S3_PMON_BOX_FILTER	Package	Uncore SBo 3 Perfmon Box-Wide Filter
744H	1860	MSR_S3_PMON_CTRL0	Package	Uncore SBo 3 Perfmon Counter 0
745H	1861	MSR_S3_PMON_CTRL1	Package	Uncore SBo 3 Perfmon Counter 1
746H	1862	MSR_S3_PMON_CTRL2	Package	Uncore SBo 3 Perfmon Counter 2
747H	1863	MSR_S3_PMON_CTRL3	Package	Uncore SBo 3 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E00H	3584	MSR_CO_PMON_BOX_CTL	Package	Uncore C-Box 0 Perfmon for Box-Wide Control
E01H	3585	MSR_CO_PMON_EVNTSELO	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 0
E02H	3586	MSR_CO_PMON_EVNTSEL1	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 1
E03H	3587	MSR_CO_PMON_EVNTSEL2	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 2
E04H	3588	MSR_CO_PMON_EVNTSEL3	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 3
E05H	3589	MSR_CO_PMON_BOX_FILTER0	Package	Uncore C-Box 0 Perfmon Box Wide Filter 0
E06H	3590	MSR_CO_PMON_BOX_FILTER1	Package	Uncore C-Box 0 Perfmon Box Wide Filter 1
E07H	3591	MSR_CO_PMON_BOX_STATUS	Package	Uncore C-Box 0 Perfmon Box Wide Status
E08H	3592	MSR_CO_PMON_CTR0	Package	Uncore C-Box 0 Perfmon Counter 0
E09H	3593	MSR_CO_PMON_CTR1	Package	Uncore C-Box 0 Perfmon Counter 1
E0AH	3594	MSR_CO_PMON_CTR2	Package	Uncore C-Box 0 Perfmon Counter 2
E0BH	3595	MSR_CO_PMON_CTR3	Package	Uncore C-Box 0 Perfmon Counter 3
E10H	3600	MSR_C1_PMON_BOX_CTL	Package	Uncore C-Box 1 Perfmon for Box-Wide Control
E11H	3601	MSR_C1_PMON_EVNTSELO	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 0
E12H	3602	MSR_C1_PMON_EVNTSEL1	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 1
E13H	3603	MSR_C1_PMON_EVNTSEL2	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 2
E14H	3604	MSR_C1_PMON_EVNTSEL3	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 3
E15H	3605	MSR_C1_PMON_BOX_FILTER0	Package	Uncore C-Box 1 Perfmon Box Wide Filter 0
E16H	3606	MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-Box 1 Perfmon Box Wide Filter 1
E17H	3607	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-Box 1 Perfmon Box Wide Status
E18H	3608	MSR_C1_PMON_CTR0	Package	Uncore C-Box 1 Perfmon Counter 0
E19H	3609	MSR_C1_PMON_CTR1	Package	Uncore C-Box 1 Perfmon Counter 1
E1AH	3610	MSR_C1_PMON_CTR2	Package	Uncore C-Box 1 Perfmon Counter 2
E1BH	3611	MSR_C1_PMON_CTR3	Package	Uncore C-Box 1 Perfmon Counter 3
E20H	3616	MSR_C2_PMON_BOX_CTL	Package	Uncore C-Box 2 Perfmon for Box-Wide Control
E21H	3617	MSR_C2_PMON_EVNTSELO	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 0
E22H	3618	MSR_C2_PMON_EVNTSEL1	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 1
E23H	3619	MSR_C2_PMON_EVNTSEL2	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 2
E24H	3620	MSR_C2_PMON_EVNTSEL3	Package	Uncore C-Box 2 Perfmon Event select for C-Box 2 Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E25H	3621	MSR_C2_PMON_BOX_FILTER0	Package	Uncore C-Box 2 Perfmon Box Wide Filter 0
E26H	3622	MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-Box 2 Perfmon Box Wide Filter1
E27H	3623	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-Box 2 Perfmon Box Wide Status
E28H	3624	MSR_C2_PMON_CTR0	Package	Uncore C-Box 2 Perfmon Counter 0
E29H	3625	MSR_C2_PMON_CTR1	Package	Uncore C-Box 2 Perfmon Counter 1
E2AH	3626	MSR_C2_PMON_CTR2	Package	Uncore C-Box 2 Perfmon Counter 2
E2BH	3627	MSR_C2_PMON_CTR3	Package	Uncore C-Box 2 Perfmon Counter 3
E30H	3632	MSR_C3_PMON_BOX_CTL	Package	Uncore C-Box 3 Perfmon for Box-Wide Control
E31H	3633	MSR_C3_PMON_EVNTSELO	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 0
E32H	3634	MSR_C3_PMON_EVNTSEL1	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 1
E33H	3635	MSR_C3_PMON_EVNTSEL2	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 2
E34H	3636	MSR_C3_PMON_EVNTSEL3	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 3
E35H	3637	MSR_C3_PMON_BOX_FILTER0	Package	Uncore C-Box 3 Perfmon Box Wide Filter 0
E36H	3638	MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-Box 3 Perfmon Box Wide Filter1
E37H	3639	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-Box 3 Perfmon Box Wide Status
E38H	3640	MSR_C3_PMON_CTR0	Package	Uncore C-Box 3 Perfmon Counter 0
E39H	3641	MSR_C3_PMON_CTR1	Package	Uncore C-Box 3 Perfmon Counter 1
E3AH	3642	MSR_C3_PMON_CTR2	Package	Uncore C-Box 3 Perfmon Counter 2
E3BH	3643	MSR_C3_PMON_CTR3	Package	Uncore C-Box 3 Perfmon Counter 3
E40H	3648	MSR_C4_PMON_BOX_CTL	Package	Uncore C-Box 4 Perfmon for Box-Wide Control
E41H	3649	MSR_C4_PMON_EVNTSELO	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 0
E42H	3650	MSR_C4_PMON_EVNTSEL1	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 1
E43H	3651	MSR_C4_PMON_EVNTSEL2	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 2
E44H	3652	MSR_C4_PMON_EVNTSEL3	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 3
E45H	3653	MSR_C4_PMON_BOX_FILTER0	Package	Uncore C-Box 4 Perfmon Box Wide Filter 0
E46H	3654	MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-Box 4 Perfmon Box Wide Filter1
E47H	3655	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-Box 4 Perfmon Box Wide Status
E48H	3656	MSR_C4_PMON_CTR0	Package	Uncore C-Box 4 Perfmon Counter 0
E49H	3657	MSR_C4_PMON_CTR1	Package	Uncore C-Box 4 Perfmon Counter 1
E4AH	3658	MSR_C4_PMON_CTR2	Package	Uncore C-Box 4 Perfmon Counter 2
E4BH	3659	MSR_C4_PMON_CTR3	Package	Uncore C-Box 4 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E50H	3664	MSR_C5_PMON_BOX_CTL	Package	Uncore C-Box 5 Perfmon for Box-Wide Control
E51H	3665	MSR_C5_PMON_EVNTSELO	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 0
E52H	3666	MSR_C5_PMON_EVNTSEL1	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 1
E53H	3667	MSR_C5_PMON_EVNTSEL2	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 2
E54H	3668	MSR_C5_PMON_EVNTSEL3	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 3
E55H	3669	MSR_C5_PMON_BOX_FILTER0	Package	Uncore C-Box 5 Perfmon Box Wide Filter 0
E56H	3670	MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-Box 5 Perfmon Box Wide Filter 1
E57H	3671	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-Box 5 Perfmon Box Wide Status
E58H	3672	MSR_C5_PMON_CTR0	Package	Uncore C-Box 5 Perfmon Counter 0
E59H	3673	MSR_C5_PMON_CTR1	Package	Uncore C-Box 5 Perfmon Counter 1
E5AH	3674	MSR_C5_PMON_CTR2	Package	Uncore C-Box 5 Perfmon Counter 2
E5BH	3675	MSR_C5_PMON_CTR3	Package	Uncore C-Box 5 Perfmon Counter 3
E60H	3680	MSR_C6_PMON_BOX_CTL	Package	Uncore C-Box 6 Perfmon for Box-Wide Control
E61H	3681	MSR_C6_PMON_EVNTSELO	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 0
E62H	3682	MSR_C6_PMON_EVNTSEL1	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 1
E63H	3683	MSR_C6_PMON_EVNTSEL2	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 2
E64H	3684	MSR_C6_PMON_EVNTSEL3	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 3
E65H	3685	MSR_C6_PMON_BOX_FILTER0	Package	Uncore C-Box 6 Perfmon Box Wide Filter 0
E66H	3686	MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-Box 6 Perfmon Box Wide Filter 1
E67H	3687	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-Box 6 Perfmon Box Wide Status
E68H	3688	MSR_C6_PMON_CTR0	Package	Uncore C-Box 6 Perfmon Counter 0
E69H	3689	MSR_C6_PMON_CTR1	Package	Uncore C-Box 6 Perfmon Counter 1
E6AH	3690	MSR_C6_PMON_CTR2	Package	Uncore C-Box 6 Perfmon Counter 2
E6BH	3691	MSR_C6_PMON_CTR3	Package	Uncore C-Box 6 Perfmon Counter 3
E70H	3696	MSR_C7_PMON_BOX_CTL	Package	Uncore C-Box 7 Perfmon for Box-Wide Control
E71H	3697	MSR_C7_PMON_EVNTSELO	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 0
E72H	3698	MSR_C7_PMON_EVNTSEL1	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 1
E73H	3699	MSR_C7_PMON_EVNTSEL2	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 2
E74H	3700	MSR_C7_PMON_EVNTSEL3	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E75H	3701	MSR_C7_PMON_BOX_FILTER0	Package	Uncore C-Box 7 Perfmon Box Wide Filter 0
E76H	3702	MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-Box 7 Perfmon Box Wide Filter 1
E77H	3703	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-Box 7 Perfmon Box Wide Status
E78H	3704	MSR_C7_PMON_CTR0	Package	Uncore C-Box 7 Perfmon Counter 0
E79H	3705	MSR_C7_PMON_CTR1	Package	Uncore C-Box 7 Perfmon Counter 1
E7AH	3706	MSR_C7_PMON_CTR2	Package	Uncore C-Box 7 Perfmon Counter 2
E7BH	3707	MSR_C7_PMON_CTR3	Package	Uncore C-Box 7 Perfmon Counter 3
E80H	3712	MSR_C8_PMON_BOX_CTL	Package	Uncore C-Box 8 Perfmon Local Box Wide Control
E81H	3713	MSR_C8_PMON_EVNTSELO	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0
E82H	3714	MSR_C8_PMON_EVNTSEL1	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1
E83H	3715	MSR_C8_PMON_EVNTSEL2	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2
E84H	3716	MSR_C8_PMON_EVNTSEL3	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3
E85H	3717	MSR_C8_PMON_BOX_FILTER0	Package	Uncore C-Box 8 Perfmon Box Wide Filter 0
E86H	3718	MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-Box 8 Perfmon Box Wide Filter 1
E87H	3719	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-Box 8 Perfmon Box Wide Status
E88H	3720	MSR_C8_PMON_CTR0	Package	Uncore C-Box 8 Perfmon Counter 0
E89H	3721	MSR_C8_PMON_CTR1	Package	Uncore C-Box 8 Perfmon Counter 1
E8AH	3722	MSR_C8_PMON_CTR2	Package	Uncore C-Box 8 Perfmon Counter 2
E8BH	3723	MSR_C8_PMON_CTR3	Package	Uncore C-Box 8 Perfmon Counter 3
E90H	3728	MSR_C9_PMON_BOX_CTL	Package	Uncore C-Box 9 Perfmon Local Box Wide Control
E91H	3729	MSR_C9_PMON_EVNTSELO	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 0
E92H	3730	MSR_C9_PMON_EVNTSEL1	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 1
E93H	3731	MSR_C9_PMON_EVNTSEL2	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 2
E94H	3732	MSR_C9_PMON_EVNTSEL3	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 3
E95H	3733	MSR_C9_PMON_BOX_FILTER0	Package	Uncore C-Box 9 Perfmon Box Wide Filter 0
E96H	3734	MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-Box 9 Perfmon Box Wide Filter 1
E97H	3735	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-Box 9 Perfmon Box Wide Status
E98H	3736	MSR_C9_PMON_CTR0	Package	Uncore C-Box 9 Perfmon Counter 0
E99H	3737	MSR_C9_PMON_CTR1	Package	Uncore C-Box 9 Perfmon Counter 1
E9AH	3738	MSR_C9_PMON_CTR2	Package	Uncore C-Box 9 Perfmon Counter 2
E9BH	3739	MSR_C9_PMON_CTR3	Package	Uncore C-Box 9 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EA0H	3744	MSR_C10_PMON_BOX_CTL	Package	Uncore C-Box 10 Perfmon Local Box Wide Control
EA1H	3745	MSR_C10_PMON_EVNTSELO	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0
EA2H	3746	MSR_C10_PMON_EVNTSEL1	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1
EA3H	3747	MSR_C10_PMON_EVNTSEL2	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2
EA4H	3748	MSR_C10_PMON_EVNTSEL3	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3
EA5H	3749	MSR_C10_PMON_BOX_FILTER0	Package	Uncore C-Box 10 Perfmon Box Wide Filter 0
EA6H	3750	MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-Box 10 Perfmon Box Wide Filter 1
EA7H	3751	MSR_C10_PMON_BOX_STATUS	Package	Uncore C-Box 10 Perfmon Box Wide Status
EA8H	3752	MSR_C10_PMON_CTR0	Package	Uncore C-Box 10 Perfmon Counter 0
EA9H	3753	MSR_C10_PMON_CTR1	Package	Uncore C-Box 10 perfmon Counter 1
EAAH	3754	MSR_C10_PMON_CTR2	Package	Uncore C-Box 10 Perfmon Counter 2
EABH	3755	MSR_C10_PMON_CTR3	Package	Uncore C-Box 10 Perfmon Counter 3
EBOH	3760	MSR_C11_PMON_BOX_CTL	Package	Uncore C-Box 11 Perfmon Local Box Wide Control
EB1H	3761	MSR_C11_PMON_EVNTSELO	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0
EB2H	3762	MSR_C11_PMON_EVNTSEL1	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1
EB3H	3763	MSR_C11_PMON_EVNTSEL2	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2
EB4H	3764	MSR_C11_PMON_EVNTSEL3	Package	Uncore C-box 11 Perfmon Event Select for C-Box 11 Counter 3
EB5H	3765	MSR_C11_PMON_BOX_FILTER0	Package	Uncore C-Box 11 Perfmon Box Wide Filter 0
EB6H	3766	MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-Box 11 Perfmon Box Wide Filter 1
EB7H	3767	MSR_C11_PMON_BOX_STATUS	Package	Uncore C-Box 11 Perfmon Box Wide Status
EB8H	3768	MSR_C11_PMON_CTR0	Package	Uncore C-Box 11 Perfmon Counter 0
EB9H	3769	MSR_C11_PMON_CTR1	Package	Uncore C-Box 11 Perfmon Counter 1
EBAH	3770	MSR_C11_PMON_CTR2	Package	Uncore C-Box 11 Perfmon Counter 2
EBBH	3771	MSR_C11_PMON_CTR3	Package	Uncore C-Box 11 Perfmon Counter 3
EC0H	3776	MSR_C12_PMON_BOX_CTL	Package	Uncore C-Box 12 Perfmon Local Box Wide Control
EC1H	3777	MSR_C12_PMON_EVNTSELO	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0
EC2H	3778	MSR_C12_PMON_EVNTSEL1	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1
EC3H	3779	MSR_C12_PMON_EVNTSEL2	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2
EC4H	3780	MSR_C12_PMON_EVNTSEL3	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EC5H	3781	MSR_C12_PMON_BOX_FILTER0	Package	Uncore C-Box 12 Perfmon Box Wide Filter 0
EC6H	3782	MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-Box 12 Perfmon Box Wide Filter 1
EC7H	3783	MSR_C12_PMON_BOX_STATUS	Package	Uncore C-Box 12 Perfmon Box Wide Status
EC8H	3784	MSR_C12_PMON_CTR0	Package	Uncore C-Box 12 Perfmon Counter 0
EC9H	3785	MSR_C12_PMON_CTR1	Package	Uncore C-Box 12 Perfmon Counter 1
ECAH	3786	MSR_C12_PMON_CTR2	Package	Uncore C-Box 12 Perfmon Counter 2
ECBH	3787	MSR_C12_PMON_CTR3	Package	Uncore C-Box 12 Perfmon Counter 3
ED0H	3792	MSR_C13_PMON_BOX_CTL	Package	Uncore C-Box 13 Perfmon local box wide control.
ED1H	3793	MSR_C13_PMON_EVNTSELO	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0
ED2H	3794	MSR_C13_PMON_EVNTSEL1	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1
ED3H	3795	MSR_C13_PMON_EVNTSEL2	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2
ED4H	3796	MSR_C13_PMON_EVNTSEL3	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3
ED5H	3797	MSR_C13_PMON_BOX_FILTER0	Package	Uncore C-Box 13 Perfmon Box Wide Filter 0
ED6H	3798	MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-Box 13 Perfmon Box Wide Filter 1
ED7H	3799	MSR_C13_PMON_BOX_STATUS	Package	Uncore C-Box 13 Perfmon Box Wide Status
ED8H	3800	MSR_C13_PMON_CTR0	Package	Uncore C-Box 13 Perfmon Counter 0
ED9H	3801	MSR_C13_PMON_CTR1	Package	Uncore C-Box 13 Perfmon Counter 1
EDAH	3802	MSR_C13_PMON_CTR2	Package	Uncore C-Box 13 Perfmon Counter 2
EDBH	3803	MSR_C13_PMON_CTR3	Package	Uncore C-Box 13 Perfmon Counter 3
EE0H	3808	MSR_C14_PMON_BOX_CTL	Package	Uncore C-Box 14 Perfmon Local Box Wide Control
EE1H	3809	MSR_C14_PMON_EVNTSELO	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0
EE2H	3810	MSR_C14_PMON_EVNTSEL1	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1
EE3H	3811	MSR_C14_PMON_EVNTSEL2	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2
EE4H	3812	MSR_C14_PMON_EVNTSEL3	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3
EE5H	3813	MSR_C14_PMON_BOX_FILTER	Package	Uncore C-Box 14 Perfmon Box Wide Filter 0
EE6H	3814	MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-Box 14 Perfmon Box Wide Filter 1
EE7H	3815	MSR_C14_PMON_BOX_STATUS	Package	Uncore C-Box 14 Perfmon Box Wide Status
EE8H	3816	MSR_C14_PMON_CTR0	Package	Uncore C-Box 14 Perfmon Counter 0
EE9H	3817	MSR_C14_PMON_CTR1	Package	Uncore C-Box 14 Perfmon Counter 1
EEAH	3818	MSR_C14_PMON_CTR2	Package	Uncore C-Box 14 Perfmon Counter 2
EEBH	3819	MSR_C14_PMON_CTR3	Package	Uncore C-Box 14 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EF0H	3824	MSR_C15_PMON_BOX_CTL	Package	Uncore C-Box 15 Perfmon Local Box Wide Control
EF1H	3825	MSR_C15_PMON_EVNTSELO	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 0
EF2H	3826	MSR_C15_PMON_EVNTSEL1	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 1
EF3H	3827	MSR_C15_PMON_EVNTSEL2	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 2
EF4H	3828	MSR_C15_PMON_EVNTSEL3	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 3
EF5H	3829	MSR_C15_PMON_BOX_FILTER0	Package	Uncore C-Box 15 Perfmon Box Wide Filter 0
EF6H	3830	MSR_C15_PMON_BOX_FILTER1	Package	Uncore C-Box 15 Perfmon Box Wide Filter 1
EF7H	3831	MSR_C15_PMON_BOX_STATUS	Package	Uncore C-Box 15 Perfmon Box Wide Status
EF8H	3832	MSR_C15_PMON_CTR0	Package	Uncore C-Box 15 Perfmon Counter 0
EF9H	3833	MSR_C15_PMON_CTR1	Package	Uncore C-Box 15 Perfmon Counter 1
EFAH	3834	MSR_C15_PMON_CTR2	Package	Uncore C-Box 15 Perfmon Counter 2
EFBH	3835	MSR_C15_PMON_CTR3	Package	Uncore C-Box 15 Perfmon Counter 3
F00H	3840	MSR_C16_PMON_BOX_CTL	Package	Uncore C-Box 16 Perfmon for Box-Wide Control
F01H	3841	MSR_C16_PMON_EVNTSELO	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 0
F02H	3842	MSR_C16_PMON_EVNTSEL1	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 1
F03H	3843	MSR_C16_PMON_EVNTSEL2	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 2
F04H	3844	MSR_C16_PMON_EVNTSEL3	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 3
F05H	3845	MSR_C16_PMON_BOX_FILTER0	Package	Uncore C-Box 16 Perfmon Box Wide Filter 0
F06H	3846	MSR_C16_PMON_BOX_FILTER1	Package	Uncore C-Box 16 Perfmon Box Wide Filter 1
F07H	3847	MSR_C16_PMON_BOX_STATUS	Package	Uncore C-Box 16 Perfmon Box Wide Status
F08H	3848	MSR_C16_PMON_CTR0	Package	Uncore C-Box 16 Perfmon Counter 0
F09H	3849	MSR_C16_PMON_CTR1	Package	Uncore C-Box 16 Perfmon Counter 1
F0AH	3850	MSR_C16_PMON_CTR2	Package	Uncore C-Box 16 Perfmon Counter 2
FOBH	3851	MSR_C16_PMON_CTR3	Package	Uncore C-Box 16 Perfmon Counter 3
F10H	3856	MSR_C17_PMON_BOX_CTL	Package	Uncore C-Box 17 Perfmon for Box-Wide Control
F11H	3857	MSR_C17_PMON_EVNTSELO	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 0
F12H	3858	MSR_C17_PMON_EVNTSEL1	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 1
F13H	3859	MSR_C17_PMON_EVNTSEL2	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 2
F14H	3860	MSR_C17_PMON_EVNTSEL3	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 3

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
F15H	3861	MSR_C17_PMON_BOX_FILTER0	Package	Uncore C-Box 17 Perfmon Box Wide Filter 0
F16H	3862	MSR_C17_PMON_BOX_FILTER1	Package	Uncore C-Box 17 Perfmon Box Wide Filter1
F17H	3863	MSR_C17_PMON_BOX_STATUS	Package	Uncore C-Box 17 Perfmon Box Wide Status
F18H	3864	MSR_C17_PMON_CTRL0	Package	Uncore C-Box 17 Perfmon Counter 0
F19H	3865	MSR_C17_PMON_CTRL1	Package	Uncore C-Box 17 Perfmon Counter 1
F1AH	3866	MSR_C17_PMON_CTRL2	Package	Uncore C-Box 17 Perfmon Counter 2
F1BH	3867	MSR_C17_PMON_CTRL3	Package	Uncore C-Box 17 Perfmon Counter 3

2.15 MSRS IN THE INTEL® CORE™ M PROCESSORS AND THE 5TH GENERATION INTEL® CORE™ PROCESSORS

The Intel® Core™ M-5xxx processors, 5th generation Intel® Core™ Processors, and the Intel® Xeon® Processor E3-1200 v4 family are based on Broadwell microarchitecture. The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH. The Intel® Xeon® Processor E3-1200 v4 family and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_47H. Processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH or 06_47H support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, Table 2-34, and Table 2-35. For an MSR listed in Table 2-35 that also appears in the model-specific tables of prior generations, Table 2-35 supersedes prior generation tables.

Table 2-34 lists MSRs that are common to processors based on the Broadwell microarchitectures (including CPUID Signature DisplayFamily_DisplayModel values of 06_3DH, 06_47H, 06_4FH, and 06_56H).

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
		0		Ovf_PMC0
		1		Ovf_PMC1
		2		Ovf_PMC2
		3		Ovf_PMC3
		31:4		Reserved
		32		Ovf_FixedCtr0
		33		Ovf_FixedCtr1
		34		Ovf_FixedCtr2
		54:35		Reserved
		55		Trace_ToPA_PMI See Section 33.2.7.2, "Table of Physical Addresses (ToPA)."
		60:56		Reserved

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		61		Ovf_Uncore
		62		Ovf_BufDSSAVE
		63		CondChgd
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."
		0		Set 1 to clear Ovf_PMC0
		1		Set 1 to clear Ovf_PMC1
		2		Set 1 to clear Ovf_PMC2
		3		Set 1 to clear Ovf_PMC3
		31:4		Reserved
		32		Set 1 to clear Ovf_FixedCtr0
		33		Set 1 to clear Ovf_FixedCtr1
		34		Set 1 to clear Ovf_FixedCtr2
		54:35		Reserved.
		55		Set 1 to clear Trace_ToPA_PMI. See Section 33.2.7.2, "Table of Physical Addresses (ToPA)."
		60:56		Reserved
		61		Set 1 to clear Ovf_Uncore
		62		Set 1 to clear Ovf_BufDSSAVE
		63		Set 1 to clear CondChgd
560H	1376	IA32_RTIT_OUTPUT_BASE	THREAD	Trace Output Base Register (R/W)
		6:0		Reserved
		MAXPHYADDR ¹ -1:7		Base physical address.
		63:MAXPHYADDR		Reserved
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	THREAD	Trace Output Mask Pointers Register (R/W)
		6:0		Reserved
		31:7		MaskOrTableOffset
		63:32		Output Offset.
570H	1392	IA32_RTIT_CTL	Thread	Trace Control Register (R/W)
		0		TraceEn
		1		Reserved, must be zero.
		2		OS
		3		User
		6:4		Reserved, must be zero.
		7		CR3Filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9		Reserved, must be zero.
		10		TSCEn
		11		DisRETC
		12		Reserved, must be zero.
		13		Reserved; writing 0 will #GP if also setting TraceEn.
		63:14		Reserved, must be zero.
571H	1393	IA32_RTIT_STATUS	Thread	Tracing Status Register (R/W)
		0		Reserved, writes ignored.
		1		ContexEn, writes ignored.
		2		TriggerEn, writes ignored.
		3		Reserved
		4		Error (R/W)
		5		Stopped
		63:6		Reserved, must be zero.
572H	1394	IA32_RTIT_CR3_MATCH	THREAD	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match.
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.

NOTES:

1. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

Table 2-35 lists MSRs that are specific to Intel Core M processors and 5th Generation Intel Core Processors.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description	
Hex	Dec				
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .	
				3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10
				9:4	Reserved
				10	I/O MWAIT Redirection Enable (R/W)
				14:11	Reserved
				15	CFG Lock (R/WO)
				24:16	Reserved
				25	C3 State Auto Demotion Enable (R/W)
				26	C1 State Auto Demotion Enable (R/W)
				27	Enable C3 Undemotion (R/W)
				28	Enable C1 Undemotion (R/W)
				29	Enable Package C-State Auto-Demotion (R/W)
				30	Enable Package C-State Undemotion (R/W)
				63:31	Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.	
				7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
				15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6core active.
		63:48		Reserved
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 15.10.4, "PPO/PP1 RAPL Domains."

See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, and Table 2-34 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH.

2.16 MSRS IN THE INTEL® XEON® PROCESSOR E5 V4 FAMILY

The MSRs listed in Table 2-36 are available and common to the Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H) and to the Intel Xeon processors E5 v4 and E7 v4 families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). These processors are based on Broadwell microarchitecture.

See Section 2.16.1 for lists of tables of MSRs that are supported by the Intel® Xeon® Processor D Family.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/W/O) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) See Table 2-26.
		22:16		Reserved.
		23	Package	PPIN_CAP (R/O) See Table 2-26.
		27:24		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.
		30	Package	Programmable TJ OFFSET (R/O) See Table 2-26.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) See Table 2-26.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/WO)

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		16		Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).
		24:17		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		MCG_EM_P
		26		MCG_ELOG_P
		63:27		Reserved
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal Threshold #1 Status (R/O) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.
		10		Power Limitation Status (R/O) See Table 2-2.
		11		Power Limitation Log (R/WCO) See Table 2-2.
		12		Current Limit Status (R/O) See Table 2-2.
		13		Current Limit Log (R/WCO) See Table 2-2.
		14		Cross Domain Limit Status (R/O) See Table 2-2.
		15		Cross Domain Limit Log (R/WCO) See Table 2-2.
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31		Reading Valid (R/O) See Table 2-2.
		63:32		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R/O) See Table 2-26.
		27:24		TCC Activation Offset (R/W) See Table 2-26.
		63:28		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C
		15:8	Package	Maximum Ratio Limit for 2C
		23:16	Package	Maximum Ratio Limit for 3C
		31:24	Package	Maximum Ratio Limit for 4C
		39:32	Package	Maximum Ratio Limit for 5C
		47:40	Package	Maximum Ratio Limit for 6C
		55:48	Package	Maximum Ratio Limit for 7C
63:56	Package	Maximum Ratio Limit for 8C		
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C
		15:8	Package	Maximum Ratio Limit for 10C
		23:16	Package	Maximum Ratio Limit for 11C
		31:24	Package	Maximum Ratio Limit for 12C
		39:32	Package	Maximum Ratio Limit for 13C
		47:40	Package	Maximum Ratio Limit for 14C
		55:48	Package	Maximum Ratio Limit for 15C
		63:56	Package	Maximum Ratio Limit for 16C
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 15.10.1, "RAPL Interfaces."
		7:4	Package	Reserved

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 15.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		2		Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit.
		3		Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit.
		4		Reserved
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		9		Reserved
		10		Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.
		12:11		Reserved
		13		Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.
		14		Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.
		15		Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		28:27		Reserved
		29		Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		31		Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:32		Reserved
770H	1904	IA32_PM_ENABLE	Package	See Section 15.4.2, "Enabling HWP."
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities."
774H	1908	IA32_HWP_REQUEST	Thread	See Section 15.4.4, "Managing HWP."
		7:0		Minimum Performance (R/W)
		15:8		Maximum Performance (R/W)
		23:16		Desired Performance (R/W)
		63:24		Reserved
777H	1911	IA32_HWP_STATUS	Thread	See Section 15.4.5, "HWP Feedback."
		1:0		Reserved
		2		Excursion to Minimum (R/O)
		63:3		Reserved
C8DH	3213	IA32_QM_EVTSEL	Thread	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31:8		Reserved
		41:32		RMID (R/W)
		63:42		Reserved
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		31:10		Reserved
		51:32		COS (R/W)
		63: 52		Reserved
C90H	3216	IA32_L3_QOS_MASK_0	Package	L3 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:19		CBM: Bit vector of available L3 ways for COS 0 enforcement.
		63:20		Reserved
C91H	3217	IA32_L3_QOS_MASK_1	Package	L3 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:19		CBM: Bit vector of available L3 ways for COS 1 enforcement.
		63:20		Reserved
C92H	3218	IA32_L3_QOS_MASK_2	Package	L3 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:19		CBM: Bit vector of available L3 ways for COS 2 enforcement.
		63:20		Reserved
C93H	3219	IA32_L3_QOS_MASK_3	Package	L3 Class Of Service Mask - COS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L3 ways for COS 3 enforcement.
		63:20		Reserved
C94H	3220	IA32_L3_QOS_MASK_4	Package	L3 Class Of Service Mask - COS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.
		0:19		CBM: Bit vector of available L3 ways for COS 4 enforcement.
		63:20		Reserved
C95H	3221	IA32_L3_QOS_MASK_5	Package	L3 Class Of Service Mask - COS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.
		0:19		CBM: Bit vector of available L3 ways for COS 5 enforcement.
		63:20		Reserved

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C96H	3222	IA32_L3_QOS_MASK_6	Package	L3 Class Of Service Mask - COS 6 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=6.
		0:19		CBM: Bit vector of available L3 ways for COS 6 enforcement.
		63:20		Reserved
C97H	3223	IA32_L3_QOS_MASK_7	Package	L3 Class Of Service Mask - COS 7 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=7.
		0:19		CBM: Bit vector of available L3 ways for COS 7 enforcement.
		63:20		Reserved
C98H	3224	IA32_L3_QOS_MASK_8	Package	L3 Class Of Service Mask - COS 8 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=8.
		0:19		CBM: Bit vector of available L3 ways for COS 8 enforcement.
		63:20		Reserved
C99H	3225	IA32_L3_QOS_MASK_9	Package	L3 Class Of Service Mask - COS 9 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=9.
		0:19		CBM: Bit vector of available L3 ways for COS 9 enforcement.
		63:20		Reserved
C9AH	3226	IA32_L3_QOS_MASK_10	Package	L3 Class Of Service Mask - COS 10 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=10.
		0:19		CBM: Bit vector of available L3 ways for COS 10 enforcement.
		63:20		Reserved
C9BH	3227	IA32_L3_QOS_MASK_11	Package	L3 Class Of Service Mask - COS 11 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=11.
		0:19		CBM: Bit vector of available L3 ways for COS 11 enforcement.
		63:20		Reserved
C9CH	3228	IA32_L3_QOS_MASK_12	Package	L3 Class Of Service Mask - COS 12 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=12.
		0:19		CBM: Bit vector of available L3 ways for COS 12 enforcement.
		63:20		Reserved
C9DH	3229	IA32_L3_QOS_MASK_13	Package	L3 Class Of Service Mask - COS 13 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=13.
		0:19		CBM: Bit vector of available L3 ways for COS 13 enforcement.
		63:20		Reserved

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C9EH	3230	IA32_L3_QOS_MASK_14	Package	L3 Class Of Service Mask - COS 14 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=14.
		0:19		CBM: Bit vector of available L3 ways for COS 14 enforcement.
		63:20		Reserved
C9FH	3231	IA32_L3_QOS_MASK_15	Package	L3 Class Of Service Mask - COS 15 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=15.
		0:19		CBM: Bit vector of available L3 ways for COS 15 enforcement.
		63:20		Reserved

2.16.1 Additional MSRs Supported in the Intel® Xeon® Processor D Product Family

The MSRs listed in Table 2-37 are available to Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H). The Intel® Xeon® processor D product family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-29, Table 2-34, Table 2-36, and Table 2-37.

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		62:0	Package	Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
418H	1048	IA32_MC6_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
See Table 2-20, Table 2-29, Table 2-34, and Table 2-36 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_56H.				

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.16.2 Additional MSRs Supported in Intel® Xeon® Processors E5 v4 and E7 v4 Families

The MSRs listed in Table 2-37 are available to the Intel® Xeon® Processor E5 v4 and E7 v4 Families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). The Intel® Xeon® processor E5 v4 family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-34, Table 2-36, and Table 2-38.

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		62:0	Package	Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
418H	1048	IA32_MC6_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
438H	1080	IA32_MC14_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
C81H	3201	IA32_L3_QOS_CFG	Package	Cache Allocation Technology Configuration (R/W)
		0		CAT Enable. Set 1 to enable Cache Allocation Technology.
		63:1		Reserved

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
See Table 2-20, Table 2-21, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.				

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.17 MSRS IN THE 6TH GENERATION, 7TH GENERATION, 8TH GENERATION, 9TH GENERATION, 10TH GENERATION, 11TH GENERATION, 12TH GENERATION, AND 13TH GENERATION INTEL® CORE™ PROCESSORS, INTEL® XEON® SCALABLE PROCESSOR FAMILY, 2ND, 3RD, AND 4TH GENERATION INTEL® XEON® SCALABLE PROCESSOR FAMILY, 8TH GENERATION INTEL® CORE™ I3 PROCESSORS, AND INTEL® XEON® E PROCESSORS

6th generation Intel® Core™ processors are based on Skylake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH or 06_5EH.

The Intel® Xeon® Scalable Processor Family based on the Skylake microarchitecture, the 2nd generation Intel® Xeon® Scalable Processor Family based on the Cascade Lake product, and the 3rd generation Intel® Xeon® Scalable Processor Family based on the Cooper Lake product all have a CPUID Signature DisplayFamily_DisplayModel value of 06_55H.

7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture, 8th generation and 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on Coffee Lake microarchitecture; these processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH.

8th generation Intel® Core™ i3 processors are based on Cannon Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

10th generation Intel® Core™ processors are based on Comet Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_A5H or 06_A6H) and Ice Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_7DH or 06_7EH).

11th generation Intel® Core™ processors are based on Tiger Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH.

The 3rd generation Intel® Xeon® Scalable Processor Family is based on Ice Lake microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH.

12th generation Intel® Core™ processors supporting the Alder Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_97H or 06_9AH.

13th generation Intel® Core™ processors supporting the Raptor Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_BAH, 06_B7H, or 06_BFH.

The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_8FH.

These processors support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-25, Table 2-29, Table 2-35, and Table 2-39¹. For an MSR listed in Table 2-39 that also appears in the model-specific tables of prior generations, Table 2-39 supersede prior generation tables.

Tables 2-40 through 2-52 list additional supported MSR interfaces for specific processors; see each table for additional details.

The notation of “Platform” in the Scope column (with respect to MSR_PLATFORM_ENERGY_COUNTER and MSR_PLATFORM_POWER_LIMIT) is limited to the power-delivery domain and the specifics of the power delivery integration may vary by platform vendor’s implementation.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	MTRR Capability (R/O, Architectural) See Table 2-2
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal threshold #1 Status (R/O) See Table 2-2.
		7		Thermal threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.
	10			Power Limitation Status (R/O) See Table 2-2.

- MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core: 3F7H. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core or P-core: 652H, 653H, 655H, 656H, DB0H, DB1H, DB2H, and D90H.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		11		Power Limitation Log (R/WC0) See Table 2-2.
		12		Current Limit Status (R/O) See Table 2-2.
		13		Current Limit Log (R/WC0) See Table 2-2.
		14		Cross Domain Limit Status (R/O) See Table 2-2.
		15		Cross Domain Limit Log (R/WC0) See Table 2-2.
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.
		31		Reading Valid (R/O) See Table 2-2.
		63:32		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, R/W if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register See http://biosbits.org .
		0		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1	Package	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		18:2		Reserved
		19		Disable Energy Efficiency Optimization (R/W) Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.
		20		Disable Race to Halt Optimization (R/W) Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.
		63:21		Reserved
300H	768	MSR_SGXOWNEREPOCH0	Package	Lower 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	768	MSR_SGXOWNEREPOCH1	Package	Upper 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 20.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Ovf_PMC0
		1	Thread	Ovf_PMC1
		2	Thread	Ovf_PMC2
		3	Thread	Ovf_PMC3
	4	Thread	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5	Thread	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Thread	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)
		7	Thread	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)
		31:8		Reserved
		32	Thread	Ovf_FixedCtr0
		33	Thread	Ovf_FixedCtr1
		34	Thread	Ovf_FixedCtr2
		54:35		Reserved
		55	Thread	Trace_ToPA_PMI
		57:56		Reserved
		58	Thread	LBR_Frz
		59	Thread	CTR_Frz
		60	Thread	ASCI
		61	Thread	Ovf_Uncore
		62	Thread	Ovf_BufDSSAVE
		63	Thread	CondChgd
390H	912	IA32_PERF_GLOBAL_STATUS_RESET		See Table 2-2. See Section 20.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Set 1 to clear Ovf_PMC0.
		1	Thread	Set 1 to clear Ovf_PMC1.
		2	Thread	Set 1 to clear Ovf_PMC2.
		3	Thread	Set 1 to clear Ovf_PMC3.
		4	Thread	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).
		5	Thread	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).
		6	Thread	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).
		7	Thread	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to clear Ovf_FixedCtr0.
		33	Thread	Set 1 to clear Ovf_FixedCtr1.
		34	Thread	Set 1 to clear Ovf_FixedCtr2.
		54:35		Reserved
		55	Thread	Set 1 to clear Trace_ToPA_PMI.
		57:56		Reserved
58	Thread	Set 1 to clear LBR_Frz.		
59	Thread	Set 1 to clear CTR_Frz.		

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		60	Thread	Set 1 to clear ASCI.
		61	Thread	Set 1 to clear Ovf_Uncore.
		62	Thread	Set 1 to clear Ovf_BufDSSAVE.
		63	Thread	Set 1 to clear CondChgd.
391H	913	IA32_PERF_GLOBAL_STATUS_SET		See Table 2-2. See Section 20.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Set 1 to cause Ovf_PMC0 = 1.
		1	Thread	Set 1 to cause Ovf_PMC1 = 1.
		2	Thread	Set 1 to cause Ovf_PMC2 = 1.
		3	Thread	Set 1 to cause Ovf_PMC3 = 1.
		4	Thread	Set 1 to cause Ovf_PMC4=1 (if CPUID.0AH:EAX[15:8] > 4).
		5	Thread	Set 1 to cause Ovf_PMC5=1 (if CPUID.0AH:EAX[15:8] > 5).
		6	Thread	Set 1 to cause Ovf_PMC6=1 (if CPUID.0AH:EAX[15:8] > 6).
		7	Thread	Set 1 to cause Ovf_PMC7=1 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to cause Ovf_FixedCtr0 = 1.
		33	Thread	Set 1 to cause Ovf_FixedCtr1 = 1.
		34	Thread	Set 1 to cause Ovf_FixedCtr2 = 1.
		54:35		Reserved
		55	Thread	Set 1 to cause Trace_ToPA_PMI = 1.
		57:56		Reserved
		58	Thread	Set 1 to cause LBR_Frz = 1.
		59	Thread	Set 1 to cause CTR_Frz = 1.
		60	Thread	Set 1 to cause ASCI = 1.
		61	Thread	Set 1 to cause Ovf_Uncore.
		62	Thread	Set 1 to cause Ovf_BufDSSAVE.
		63		Reserved
392H	913	IA32_PERF_GLOBAL_INUSE	Thread	See Table 2-2.
3F7H	1015	MSR_PEBS_FRONTEND	Thread	FrontEnd Precise Event Condition Select (R/W)
		2:0		Event Code Select
		3		Reserved
		4		Event Code Select High

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7:5		Reserved
		19:8		IDQ_Bubble_Length Specifier
		22:20		IDQ_Bubble_Width Specifier
		63:23		Reserved
500H	1280	IA32_SGX_SVN_STATUS	Thread	Status and SVN Threshold of SGX Support for ACM (R/O)
		0		Lock See Section 39.11.3, "Interactions with Authenticated Code Modules (ACMs)."
		15:1		Reserved
		23:16		SGX_SVN_SINIT See Section 39.11.3, "Interactions with Authenticated Code Modules (ACMs)."
		63:24		Reserved
560H	1376	IA32_RTIT_OUTPUT_BASE	Thread	Trace Output Base Register (R/W) See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Thread	Trace Output Mask Pointers Register (R/W) See Table 2-2.
570H	1392	IA32_RTIT_CTL	Thread	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		6:4		Reserved, must be zero.
		7		CR3Filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.
		9		MTCEn
		10		TSCEn
		11		DisRETC
		12		Reserved, must be zero.
		13		BranchEn
		17:14		MTCFreq
		18		Reserved, must be zero.
22:19		CycThresh		
23		Reserved, must be zero.		

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		27:24		PSBFreq
		31:28		Reserved, must be zero.
		35:32		ADDR0_CFG
		39:36		ADDR1_CFG
		63:40		Reserved, must be zero.
571H	1393	IA32_RTIT_STATUS	Thread	Tracing Status Register (R/W)
		0		FilterEn, writes ignored.
		1		ContexEn, writes ignored.
		2		TriggerEn, writes ignored.
		3		Reserved
		4		Error (R/W)
		5		Stopped
		31:6		Reserved, must be zero.
		48:32		PacketByteCnt
		63:49		Reserved, must be zero.
572H	1394	IA32_RTIT_CR3_MATCH	Thread	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match
580H	1408	IA32_RTIT_ADDR0_A	Thread	Region 0 Start Address (R/W)
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDR0_B	Thread	Region 0 End Address (R/W)
		63:0		See Table 2-2.
582H	1410	IA32_RTIT_ADDR1_A	Thread	Region 1 Start Address (R/W)
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Thread	Region 1 End Address (R/W)
		63:0		See Table 2-2.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."
64DH	1613	MSR_PLATFORM_ENERGY_COUNTER	Platform	Platform Energy Counter (R/O) This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it. This MSR will read 0 if not valid.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31:0		Total energy consumed by all devices in the platform that receive power from integrated power delivery mechanism, included platform devices are processor cores, SOC, memory, add-on or peripheral devices that get powered directly from the platform power delivery means. The energy units are specified in the MSR_RAPL_POWER_UNIT.Energy_Status_Unit.
		63:32		Reserved
64EH	1614	MSR_PPERF	Thread	Productive Performance Count (R/O)
		63:0		Hardware's view of workload scalability. See Section 15.4.5.1.
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved
		4		Residency State Regulation Status (R0) When set, frequency is reduced below the operating system request due to residency state regulation limit.
		5		Running Average Thermal Limit Status (R0) When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).
		7		VR Therm Design Current Status (R0) When set, frequency is reduced below the operating system request due to VR thermal design current limit.
		8		Other Status (R0) When set, frequency is reduced below the operating system request due to electrical or other constraints.
		9		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		10		Package/Platform-Level Power Limiting PL1 Status (RO) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.
		11		Package/Platform-Level PL2 Power Limiting Status (RO) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.
		12		Max Turbo Limit Status (RO) When set, frequency is reduced below the operating system request due to multi-core turbo limits.
		13		Turbo Transition Attenuation Status (RO) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		15:14		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		Residency State Regulation Log When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		VR Thermal Design Current Log When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24		Other Log When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
652H	1618	MSR_PKG_HDC_CONFIG	Package	HDC Configuration (R/W)

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		2:0		PKG_Cx_Monitor Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY.
		63:3		Reserved
653H	1619	MSR_CORE_HDC_RESIDENCY	Core	Core HDC Idle Residency (R/O)
		63:0		Core_Cx_Duty_Cycle_Cnt
655H	1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	Package	Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle (R/O)
		63:0		Pkg_C2_Duty_Cycle_Cnt
656H	1622	MSR_PKG_HDC_DEEP_RESIDENCY	Package	Package Cx HDC Idle Residency (R/O)
		63:0		Pkg_Cx_Duty_Cycle_Cnt
658H	1624	MSR_WEIGHTED_CORE_CO	Package	Core-count Weighted C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.
659H	1625	MSR_ANY_CORE_CO	Package	Any Core C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor core in the package is in C0.
65AH	1626	MSR_ANY_GFXE_CO	Package	Any Graphics Engine C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.
65BH	1627	MSR_CORE_GFXE_OVERLAP_CO	Package	Core and Graphics Engine Overlapped C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.
65CH	1628	MSR_PLATFORM_POWER_LIMIT	Platform	Platform Power Limit Control (R/W-L) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		14:0		Platform Power Limit #1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.
		15		Enable Platform Power Limit #1 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.
		16		Platform Clamping Limitation #1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID (EAX=6):EAX[4] is set.
		23:17		Time Window for Platform Power Limit #1 Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, The unit is specified in MSR_RAPLPOWER_UNIT[Time Unit].
		31:24		Reserved
		46:32		Platform Power Limit #2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit # 1).
		47		Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		48		Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.
		62:49		Reserved
		63		Lock. Setting this bit will lock all other bits of this MSR until system RESET.
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Thread	Last Branch Record 16 From IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.12.
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Thread	Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Thread	Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Thread	Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Thread	Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Thread	Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Thread	Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Thread	Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Thread	Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Thread	Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Thread	Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Thread	Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Thread	Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Thread	Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Thread	Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Thread	Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6BOH	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)
		0		PROCHOT Status (RO) When set, frequency is reduced due to assertion of external PROCHOT.
		1		Thermal Status (RO) When set, frequency is reduced due to a thermal event.
		4:2		Reserved.
		5		Running Average Thermal Limit Status (RO) When set, frequency is reduced due to running average thermal limit.
		6		VR Therm Alert Status (RO) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.
		7		VR Thermal Design Current Status (RO) When set, frequency is reduced due to VR TDC limit.
		8		Other Status (RO) When set, frequency is reduced due to electrical or other constraints.
		9		Reserved
		10		Package/Platform-Level Power Limiting PL1 Status (RO) When set, frequency is reduced due to package/platform-level power limiting PL1.
		11		Package/Platform-Level PL2 Power Limiting Status (RO) When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.
		12		Inefficient Operation Status (RO) When set, processor graphics frequency is operating below target frequency.
15:13	Reserved			

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20:18		Reserved.
		21		Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24		Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		27		Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Inefficient Operation Log When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:29		Reserved
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)
		0		PROCHOT Status (RO) When set, frequency is reduced due to assertion of external PROCHOT.
		1		Thermal Status (RO) When set, frequency is reduced due to a thermal event.
		4:2		Reserved
		5		Running Average Thermal Limit Status (RO) When set, frequency is reduced due to running average thermal limit.
		6		VR Therm Alert Status (RO) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.
		7		VR Thermal Design Current Status (RO) When set, frequency is reduced due to VR TDC limit.
		8		Other Status (RO) When set, frequency is reduced due to electrical or other constraints.
		9		Reserved
		10		Package/Platform-Level Power Limiting PL1 Status (RO) When set, frequency is reduced due to package/Platform-level power limiting PL1.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		11		Package/Platform-Level PL2 Power Limiting Status (RO) When set, frequency is reduced due to package/Platform-level power limiting PL2/PL3.
		15:12		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20:18		Reserved
		21		Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24		Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:28		Reserved
6D0H	1744	MSR_LASTBRANCH_16_TO_IP	Thread	Last Branch Record 16 To IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.12.
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Thread	Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Thread	Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Thread	Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Thread	Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Thread	Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Thread	Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Thread	Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Thread	Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Thread	Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Thread	Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Thread	Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Thread	Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Thread	Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Thread	Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Thread	Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
770H	1904	IA32_PM_ENABLE	Package	See Section 15.4.2, "Enabling HWP."
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities."
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Section 15.4.4, "Managing HWP."
773H	1907	IA32_HWP_INTERRUPT	Thread	See Section 15.4.6, "HWP Notifications."
774H	1908	IA32_HWP_REQUEST	Thread	See Section 15.4.4, "Managing HWP."
		7:0		Minimum Performance (R/W)
		15:8		Maximum Performance (R/W)
		23:16		Desired Performance (R/W)
		31:24		Energy/Performance Preference (R/W)
		41:32		Activity Window (R/W)
		42		Package Control (R/W)
		63:43		Reserved
777H	1911	IA32_HWP_STATUS	Thread	See Section 15.4.5, "HWP Feedback."
D90H	3472	IA32_BNDCFGS	Thread	See Table 2-2.
DA0H	3488	IA32_XSS	Thread	See Table 2-2.
DB0H	3504	IA32_PKG_HDC_CTL	Package	See Section 15.5.2, "Package level Enabling HDC."
DB1H	3505	IA32_PM_CTL1	Thread	See Section 15.5.3, "Logical-Processor Level HDC Control."
DB2H	3506	IA32_THREAD_STALL	Thread	See Section 15.5.4.1, "IA32_THREAD_STALL."
DC0H	3520	MSR_LBR_INFO_0	Thread	Last Branch Record 0 Additional Information (R/W) One of 32 triplet of last branch record registers on the last branch record stack. This part of the stack contains flag, TSX-related and elapsed cycle information. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.9.1, "LBR Stack."

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DC1H	3521	MSR_LBR_INFO_1	Thread	Last Branch Record 1 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC2H	3522	MSR_LBR_INFO_2	Thread	Last Branch Record 2 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC3H	3523	MSR_LBR_INFO_3	Thread	Last Branch Record 3 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC4H	3524	MSR_LBR_INFO_4	Thread	Last Branch Record 4 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC5H	3525	MSR_LBR_INFO_5	Thread	Last Branch Record 5 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC6H	3526	MSR_LBR_INFO_6	Thread	Last Branch Record 6 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC7H	3527	MSR_LBR_INFO_7	Thread	Last Branch Record 7 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC8H	3528	MSR_LBR_INFO_8	Thread	Last Branch Record 8 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC9H	3529	MSR_LBR_INFO_9	Thread	Last Branch Record 9 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCAH	3530	MSR_LBR_INFO_10	Thread	Last Branch Record 10 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCBH	3531	MSR_LBR_INFO_11	Thread	Last Branch Record 11 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCCH	3532	MSR_LBR_INFO_12	Thread	Last Branch Record 12 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCDH	3533	MSR_LBR_INFO_13	Thread	Last Branch Record 13 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCEH	3534	MSR_LBR_INFO_14	Thread	Last Branch Record 14 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCFH	3535	MSR_LBR_INFO_15	Thread	Last Branch Record 15 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD0H	3536	MSR_LBR_INFO_16	Thread	Last Branch Record 16 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD1H	3537	MSR_LBR_INFO_17	Thread	Last Branch Record 17 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD2H	3538	MSR_LBR_INFO_18	Thread	Last Branch Record 18 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD3H	3539	MSR_LBR_INFO_19	Thread	Last Branch Record 19 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD4H	3520	MSR_LBR_INFO_20	Thread	Last Branch Record 20 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD5H	3521	MSR_LBR_INFO_21	Thread	Last Branch Record 21 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD6H	3522	MSR_LBR_INFO_22	Thread	Last Branch Record 22 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD7H	3523	MSR_LBR_INFO_23	Thread	Last Branch Record 23 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD8H	3524	MSR_LBR_INFO_24	Thread	Last Branch Record 24 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD9H	3525	MSR_LBR_INFO_25	Thread	Last Branch Record 25 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDAH	3526	MSR_LBR_INFO_26	Thread	Last Branch Record 26 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDBH	3527	MSR_LBR_INFO_27	Thread	Last Branch Record 27 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDCH	3528	MSR_LBR_INFO_28	Thread	Last Branch Record 28 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDDH	3529	MSR_LBR_INFO_29	Thread	Last Branch Record 29 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDEH	3530	MSR_LBR_INFO_30	Thread	Last Branch Record 30 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDFH	3531	MSR_LBR_INFO_31	Thread	Last Branch Record 31 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-40 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH, 06_5EH, 06_8EH, 06_9EH, or 06_66H.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		43:0		Current count.
		63:44		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
E01H	3585	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4select.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved
E02H	3586	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved

2.17.1 MSRs Introduced in 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Table 2-41 lists additional MSRs for 7th generation and 8th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH. For an MSR listed in Table 2-41 that also appears in the model-specific tables of prior generations, Table 2-41 supersedes prior generation tables.

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
80H	128	MSR_TRACE_HUB_STH ACPIBAR_BASE	Package	NPK Address Used by AET Messages (R/W)
		0		Lock Bit If set, this MSR cannot be re-written anymore. Lock bit has to be set in order for the AET packets to be directed to NPK MMIO.
		17:1		Reserved
		63:18		ACPIBAR_BASE_ADDRESS AET target address in NPK MMIO space.
1F4H	500	MSR_PRMRR_PHYS_BASE	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MemType PRMRR BASE MemType.
		11:3		Reserved
		45:12		Base PRMRR Base Address.
		63:46		Reserved

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1F5H	501	MSR_PRMRR_PHYS_MASK	Core	Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)
		9:0		Reserved
		10		Lock Lock bit for the PRMRR.
		11		VLD Enable bit for the PRMRR.
		45:12		Mask PRMRR MASK bits.
		63:46		Reserved
1FBH	507	MSR_PRMRR_VALID_CONFIG	Core	Valid PRMRR Configurations (R/W)
		0		1M supported MEE size.
		4:1		Reserved
		5		32M supported MEE size.
		6		64M supported MEE size.
		7		128M supported MEE size.
		31:8		Reserved
		2F4H		756
11:0	Reserved			
PAWIDTH-1:12	Range Base This field corresponds to bits PAWIDTH-1:12 of the base address memory range which is allocated to PRMRR memory.			
63:PAWIDTH	Reserved			
2F5H	757	MSR_UNCORE_PRMRR_PHYS_MASK ¹	Package	(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.
		9:0		Reserved
		10		Lock Setting this bit locks all writeable settings in this register, including itself.
		11		Range_En Indicates whether the PRMRR range is enabled and valid.

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		38:12		Range_Mask This field indicates which address bits must match PRMRR base in order to qualify as an PRMRR access.
		63:39		Reserved
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		6:0		MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.
		7		Reserved
		14:8		MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.
		63:15		Reserved

NOTES:

1. This MSR is specific to 7th generation and 8th generation Intel® Core™ processors.

2.17.2 MSRs Specific to 8th Generation Intel® Core™ i3 Processors

Table 2-42 lists additional MSRs for 8th generation Intel Core i3 processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H. For an MSR listed in Table 2-42 that also appears in the model-specific tables of prior generations, Table 2-42 supersedes prior generation tables.

Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)

**Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		17		SGX Launch Control Enable (R/W/L) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Available only if CPUID.(EAX=07H, ECX=0H): ECX[30] = 1.
		18		SGX Global Functions Enable (R/W/L)
		63:21		Reserved
350H	848	MSR_BR_DETECT_CTRL		Branch Monitoring Global Control (R/W)
		0		EnMonitoring Global enable for branch monitoring.
		1		EnExcept Enable branch monitoring event signaling on threshold trip. The branch monitoring event handler is signaled via the existing PMI signaling mechanism as programmed from the corresponding local APIC LVT entry.
		2		EnLBRFrz Enable LBR freeze on threshold trip. This will cause the LBR frozen bit 58 to be set in IA32_PERF_GLOBAL_STATUS when a triggering condition occurs and this bit is enabled.
		3		DisableInGuest When set to '1', branch monitoring, event triggering and LBR freeze actions are disabled when operating at VMX non-root operation.
		7:4		Reserved
		17:8		WindowSize Window size defined by WindowCntSel. Values 0 - 1023 are supported. Once the Window counter reaches the WindowSize count both the Window Counter and all Branch Monitoring Counters are cleared.
		23:18		Reserved
		25:24		WindowCntSel Window event count select: '00 = Instructions retired. '01 = Branch instructions retired '10 = Return instructions retired. '11 = Indirect branch instructions retired.

**Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		<p>CntAndMode</p> <p>When set to '1', the overall branch monitoring event triggering condition is true only if all enabled counters' threshold conditions are true.</p> <p>When '0', the threshold tripping condition is true if any enabled counters' threshold is true.</p>
		63:27		Reserved
351H	849	MSR_BR_DETECT_STATUS		Branch Monitoring Global Status (R/W)
		0		<p>Branch Monitoring Event Signaled</p> <p>When set to '1', Branch Monitoring event signaling is blocked until this bit is cleared by software.</p>
		1		<p>LBRsValid</p> <p>This status bit is set to '1' if the LBR state is considered valid for sampling by branch monitoring software.</p>
		7:2		Reserved
		8		<p>CntrHit0</p> <p>Branch monitoring counter #0 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.</p>
		9		<p>CntrHit1</p> <p>Branch monitoring counter #1 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.</p>
		15:10		<p>Reserved</p> <p>Reserved for additional branch monitoring counters threshold hit status.</p>
		25:16		<p>CountWindow</p> <p>The current value of the window counter. The count value is frozen on a valid branch monitoring triggering condition. This is a 10-bit unsigned value.</p>
		31:26		<p>Reserved</p> <p>Reserved for future extension of CountWindow.</p>

**Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32		<p>Count0</p> <p>The current value of counter 0 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit0 will also be set). This is an 8-bit signed value (2's complement).</p> <p>Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256).</p> <p>RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).</p>
		47:40		<p>Count1</p> <p>The current value of counter 1 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit1 will also be set). This is an 8-bit signed value (2's complement).</p> <p>Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256).</p> <p>RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).</p>
		63:48		Reserved
354H - 355H	852 - 853	MSR_BR_DETECT_COUNTER_CONFIG_i		Branch Monitoring Detect Counter Configuration (R/W)
		0		<p>CntrEn</p> <p>Enable counter.</p>
		7:1		<p>CntrEvSel</p> <p>Event select (other values #GP)</p> <p>'0000000 = RETs.</p> <p>'0000001 = RET-CALL bias.</p> <p>'0000010 = RET mispredicts.</p> <p>'0000011 = Branch (all) mispredicts.</p> <p>'0000100 = Indirect branch mispredicts.</p> <p>'0000101 = Far branch instructions.</p>
		14:8		<p>CntrThreshold</p> <p>Threshold (an unsigned value of 0 to 127 supported). The value 0 of counter threshold will result in event signaled after every instruction. #GP if threshold is < 2.</p>

**Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15		MispredEventCnt Mispredict events counting behavior: '0 = Mispredict events are counted in a window. '1 = Mispredict events are counted based on a consecutive occurrence. CntrThreshold is treated as # of consecutive mispredicts. This control bit only applies to events specified by CntrEvSel that involve a prediction (0000010, 0000011, 0000100). Setting this bit for other events is ignored.
		63:16		Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Package C3 Residency Counter (R/O)
		63:0		Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		6:0		MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.
		7		Reserved
		14:8		MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.
		63:15		Reserved
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Core C1 Residency Counter (R/O)
		63:0		Value since last reset for the Core C1 residency. Counter rate is the Max Non-Turbo frequency (same as TSC). This counter counts in case both of the core's threads are in an idle state and at least one of the core's thread residency is in a C1 state or in one of its sub states. The counter is updated only after a core C state exit. Note: Always reads 0 if core C1 is unsupported. A value of zero indicates that this processor does not support core C1 or never entered core C1 level state.
662H	1634	MSR_CORE_C3_RESIDENCY	Core	Core C3 Residency Counter (R/O)
		63:0		Will always return 0.

Table 2-43 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Report the number of C-Box units with performance counters, including processor cores and processor graphics.
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, Counter 1 Event Select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
702H	1794	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
703H	1795	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
708H	1800	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
709H	1801	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
70AH	1802	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
70BH	1803	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
710H	1808	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
712H	1810	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
713H	1811	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
718H	1816	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
719H	1817	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
71AH	1818	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
71BH	1819	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
720H	1824	MSR_UNC_CBO_4_PERFEVTSELO	Package	Uncore C-Box 4, Counter 0 Event Select MSR

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
721H	1825	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
722H	1826	MSR_UNC_CBO_4_PERFCTRO	Package	Uncore C-Box 4, Performance Counter 0
723H	1827	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
728H	1832	MSR_UNC_CBO_5_PERFEVTSELO	Package	Uncore C-Box 5, Counter 0 Event Select MSR
729H	1833	MSR_UNC_CBO_5_PERFEVTSEL1	Package	Uncore C-Box 5, Counter 1 Event Select MSR
72AH	1834	MSR_UNC_CBO_5_PERFCTRO	Package	Uncore C-Box 5, Performance Counter 0
72BH	1835	MSR_UNC_CBO_5_PERFCTR1	Package	Uncore C-Box 5, Performance Counter 1
730H	1840	MSR_UNC_CBO_6_PERFEVTSELO	Package	Uncore C-Box 6, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_6_PERFEVTSEL1	Package	Uncore C-Box 6, Counter 1 Event Select MSR
732H	1842	MSR_UNC_CBO_6_PERFCTRO	Package	Uncore C-Box 6, Performance Counter 0
733H	1843	MSR_UNC_CBO_6_PERFCTR1	Package	Uncore C-Box 6, Performance Counter 1
738H	1848	MSR_UNC_CBO_7_PERFEVTSELO	Package	Uncore C-Box 7, Counter 0 Event Select MSR
739H	1849	MSR_UNC_CBO_7_PERFEVTSEL1	Package	Uncore C-Box 7, Counter 1 Event Select MSR
73AH	1850	MSR_UNC_CBO_7_PERFCTRO	Package	Uncore C-Box 7, Performance Counter 0
73BH	1851	MSR_UNC_CBO_7_PERFCTR1	Package	Uncore C-Box 7, Performance Counter 1
E01H	3585	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4select.
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
63:32		Reserved		
E02H	3586	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved

2.17.3 MSRs Introduced in 10th Generation Intel® Core™ Processors

Table 2-44 lists additional MSRs for 10th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7DH or 06_7EH. For an MSR listed in Table 2-44 that also appears in the model-specific tables of prior generations, Table 2-44 supersedes prior generation tables.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		28:0		Reserved.
		29		SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 9.1.2.3, "Features to Disable Bus Locks."
		30		Reserved.
		31		Reserved.
48H	72	IA32_SPEC_CTRL	Core	See Table 2-2.
49H	73	IA32_PREDICT_CMD	Thread	See Table 2-2.
8CH	140	IA32_SGXLEPUBKEYHASH0	Thread	See Table 2-2.
8DH	141	IA32_SGXLEPUBKEYHASH1	Thread	See Table 2-2.
8EH	142	IA32_SGXLEPUBKEYHASH2	Thread	See Table 2-2.
8FH	143	IA32_SGXLEPUBKEYHASH3	Thread	See Table 2-2.
A0H	160	MSR_BIOS_MCU_ERRORCODE	Package	BIOS MCU ERRORCODE (R/O) This MSR indicates if WRMSR 0x79 failed to configure PRM memory and gives a hint to debug BIOS.
		15:0	Package	Error Codes (R/O)
		30:16		Reserved.
		31	Thread	MCU Partial Success (R/O) When set to 1, WRMSR 0x79 skipped part of the functionality during BIOS.
A5H	165	MSR_FIT_BIOS_ERROR	Thread	FIT BIOS ERROR (R/W) Report error codes for debug in case the processor failed to parse the Firmware Table in BIOS. Can also be used to log BIOS information.
		7:0		Error Codes (R/W) Error codes for debug.
		15:8		Entry Type (R/W) Failed FIT entry type.
		16		FIT MCU Entry (R/W) FIT contains MCU entry.
		62:17		Reserved.
		63		LOCK (R/W) When set to 1, writes to this MSR will be skipped.
10BH	267	IA32_FLUSH_CMD	Thread	See Table 2-2.
151H	337	MSR_BIOS_DONE	Thread	BIOS Done (R/W/O)

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0	Thread	BIOS Done Indication (R/WO) Set by BIOS when it finishes programming the processor and wants to lock the memory configuration from changes by software that is running on this thread. Writes to the bit will be ignored if EAX[0] is 0.
		1	Package	Package BIOS Done Indication (R/O) When set to 1, all threads in the package have bit 0 of this MSR set.
		31:2		Reserved.
1F1H	497	MSR_CRASHLOG_CONTROL	Thread	Write Data to a Crash Log Configuration
		0		CDDIS: CrashDump_Disable If set, indicates that Crash Dump is disabled.
		63:1		Reserved.
2A0H	672	MSR_PLMRR_BASE_0	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MEMTYPE: PLMRR BASE Memory Type.
		3		CONFIGURED: PLMRR BASE Configured.
		11:4		Reserved.
		51:12		BASE: PLMRR Base Address.
63:52		Reserved.		
30CH	780	IA32_FIXED_CTR3	Thread	Fixed-Function Performance Counter Register 3 (R/W) Bit definitions are the same as found in IA32_FIXED_CTR0, offset 309H. See Table 2-2.
329H	809	MSR_PERF_METRICS	Thread	Performance Metrics (R/W) Reports metrics directly. Software can check (and/or expose to its guests) the availability of PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15).
		7:0		Retiring. Percent of utilized slots by uops that eventually retire (commit).
		15:8		Bad Speculation. Percent of wasted slots due to incorrect speculation, covering utilized by uops that do not retire, or recovery bubbles (unutilized slots).
		23:16		Frontend Bound. Percent of unutilized slots where front-end did not deliver a uop while back-end is ready.
		31:24		Backend Bound. Percent of unutilized slots where a uop was not delivered to back-end due to lack of back-end resources.
		63:25		Reserved.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3F2H	1010	MSR_PEBS_DATA_CFG	Thread	PEBS Data Configuration (R/W) Provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.
		0		Memory Info. Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.
		1		GPRs. Setting this bit will capture the contents of the General Purpose registers in the PEBS record.
		2		XMMs. Setting this bit will capture the contents of the XMM registers in the PEBS record.
		3		LBRs. Setting this bit will capture LBR TO, FROM, and INFO in the PEBS record.
		23:4		Reserved.
		31:24		LBR Entries. Set the field to the desired number of entries - 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, software can use LBR_entries that is multiple of 3.
541H	1345	MSR_CORE_UARCH_CTL	Core	Core Microarchitecture Control MSR (R/W)
		0		L1 Scrubbing Enable When set to 1, enable L1 scrubbing.
		31:1		Reserved.
657H	1623	MSR_FAST_UNCORE_MSRS_CTL	Thread	Fast WRMSR/RDMSR Control MSR (R/W)
		3:0		FAST_ACCESS_ENABLE: Bit 0: When set to '1', provides a hint for the hardware to enable fast access mode for the IA32_HWP_REQUEST MSR. This bit is sticky and is cleaned by the hardware only during reset time. This bit is valid only if FAST_UNCORE_MSRS_CAPABILITY[0] is set. Setting this bit will cause CPUID[6].EAX[18] to be set.
		31:4		Reserved.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
65EH	1630	MSR_FAST_UNCORE_MSRS_STATUS	Thread	Indication of Uncore MSRs, Post Write Activates
		0		Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor. Software can use the status of this bit to avoid overwriting IA32_HWP_REQUEST.
		31:1		Reserved.
65FH	1631	MSR_FAST_UNCORE_MSRS_CAPABILITY	Thread	Fast WRMSR/RDMSR Enumeration MSR (R/O)
		3:0		MSRS_CAPABILITY: Bit 0: If set to '1', hardware supports the fast access mode for the IA32_HWP_REQUEST MSR.
		31:4		Reserved.
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Table 2-2.
775H	1909	IA32_PECI_HWP_REQUEST_INFO	Thread	See Table 2-2.
777H	1911	IA32_HWP_STATUS	Thread	See Table 2-2.

2.17.4 MSRs Introduced in the 11th Generation Intel® Core™ Processors based on Tiger Lake Microarchitecture

Table 2-45 lists additional MSRs for 11th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH. The MSRs listed in Table 2-44 are also supported by these processors. For an MSR listed in Table 2-45 that also appears in the model-specific tables of prior generations, Table 2-45 supersedes prior generation tables.

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
A0H	160	MSR_BIOS_MCU_ERRORCODE	Package	BIOS MCU ERRORCODE (R/O)
		15:0		Error Codes
		31:16		Reserved
A7H	167	MSR_BIOS_DEBUG	Thread	BIOS DEBUG (R/O) This MSR indicates if WRMSR 79H failed to configure PRM memory and gives a hint to debug BIOS.
		30:0		Reserved

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31		MCU Partial Success When set to 1, WRMSR 79H skipped part of the functionality during BIOS.
		63:32		Reserved
CFH	207	IA32_CORE_CAPABILITIES	Package	IA32 Core Capabilities Register (R/O) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		1:0		Reserved
		2		FUSA_SUPPORTED
		3		RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.
		4		Reserved
		5		SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).
		31:6		Reserved
492H	1170	IA32_VMX_PROCBASED_CTL3	Core	IA32_VMX_PROCBASED_CTL3 This MSR enumerates the allowed 1-settings of the third set of processor-based controls. Specifically, VM entry allows bit X of the tertiary processor-based VM-execution controls to be 1 if and only if bit X of the MSR is set to 1. If bit X of the MSR is cleared to 0, VM entry fails if control X and the “activate tertiary controls” primary processor-based VM-execution control are both 1.
		0		LOADIWKEY This control determines whether executions of LOADIWKEY cause VM exits.
		63:1		Reserved
601H	1537	MSR_VR_CURRENT_CONFIG	Package	Power Limit 4 (PL4) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.
		12:0		PL4 Value PL4 value in 0.125 A increments. This field is locked by VR_CURRENT_CONFIG[LOCK]. When the LOCK bit is set to 1b, this field becomes Read Only.
		30:13		Reserved

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31		Lock Indication (LOCK) This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1b, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.
		62:32		Not in use.
		63		Reserved
6A0H	1696	IA32_U_CET		Configure User Mode CET (R/W) See Table 2-2.
6A2H	1698	IA32_S_CET		Configure Supervisor Mode CET (R/W) See Table 2-2.
6A4H	1700	IA32_PL0_SSP		Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.
6A5H	1701	IA32_PL1_SSP		Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.
6A6H	1702	IA32_PL2_SSP		Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.
6A7H	1703	IA32_PL3_SSP		Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.
6A8H	1704	IA32_INTERRUPT_SSP_TABLE_ADDR		Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.
981H	2433	IA32_TME_CAPABILITY		See Table 2-2.
982H	2434	IA32_TME_ACTIVATE		See Table 2-2.
983H	2435	IA32_TME_EXCLUDE_MASK		See Table 2-2.
984H	2436	IA32_TME_EXCLUDE_BASE		See Table 2-2.
990H	2448	IA32_COPY_STATUS ¹	Thread	See Table 2-2.
991H	2449	IA32_IWKEYBACKUP_STATUS ¹	Platform	See Table 2-2.
C82H	3202	IA32_L2_QOS_CFG	Core	IA32_CR_L2_QOS_CFG This MSR provides software an enumeration of the parameters that L2 QoS (Intel RDT) support in any particular implementation.

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		CDP_ENABLE When set to 1, it will enable the code and data prioritization for the L2 CAT/Intel RDT feature. When set to 0, code and data prioritization is disabled for L2 CAT/Intel RDT. See Chapter 18, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” for further details on CDP.
		31:1		Reserved
D10H - D17H	3220 - 3351	IA32_L2_QOS_MASK_[0-7]	Package	IA32_CR_L2_QOS_MASK_[0-7] Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 18, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”
		19:0		WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.4.2:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.
		31:20		Reserved
D91H	3473	IA32_COPY_LOCAL_TO_PLATFORM ¹	Thread	See Table 2-2.
D92H	3474	IA32_COPY_PLATFORM_TO_LOCAL ¹	Thread	See Table 2-2.

NOTES:

1. Further details on Key Locker and usage of this MSR can be found here:

<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>

2.17.5 MSRs Introduced in the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Table 2-46 lists additional MSRs for 12th and 13th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH. Table 2-47 lists the MSRs unique to the processor P-core. Table 2-48 lists the MSRs unique to the processor E-core.

The MSRs listed in Table 2-44¹ and Table 2-45 are also supported by these processors. For an MSR listed in Table 2-46, Table 2-47, or Table 2-48 that also appears in the model-specific tables of prior generations, Table 2-46, Table 2-47, and Table 2-48 supersede prior generation tables.

1. MSRs at the following addresses are not supported in the 12th and 13th generation Intel Core processor E-core: 30CH, 329H, 541H, and 657H. The MSR at address 657H is not supported in the 12th and 13th generation Intel Core processor P-core.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		26:0		Reserved.
		27		UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.
		28		UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 9.1.2.3, "Features to Disable Bus Locks."
		29		SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 9.1.2.3, "Features to Disable Bus Locks."
		30		Reserved.
		31		Reserved.
BCH	188	IA32_MISC_PACKAGE_CTL5	Package	Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.
C7H	199	IA32_PMC6	Core	General Performance Counter 6 (R/W) See Table 2-2.
C8H	200	IA32_PMC7	Core	General Performance Counter 7 (R/W) See Table 2-2.
CFH	207	IA32_CORE_CAPABILITIES	Package	IA32 Core Capabilities Register (R/O) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		0		STLB_QOS_SUPPORTED When set to 1, the STLB QoS feature is supported and the STLB QoS MSRs (1A8FH - 1A97H) are accessible. When set to 0, access to these MSRs will #GP.
		1		Reserved
		2		FUSA_SUPPORTED
		3		RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.
		4		UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5		SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.
		6		SNOOP_FILTER_QOS_SUPPORTED When set to 1, the Snoop Filter Qos Mask MSRs are supported. When set to 0, access to these MSRs will #GP.
		7		UC_STORE_THROTTLING_SUPPORTED When set 1, UC Store throttle capability exist through MSR_MEMORY_CTRL (33H) bit 27.
		31:8		Reserved
E1H	225	IA32_UMWAIT_CONTROL		UMWAIT Control (R/W) See Table 2-2.
10AH	266	IA32_ARCH_CAPABILITIES		Enumeration of Architectural Features (R/O) See Table 2-2.
18CH	396	IA32_PERFEVTSEL6	Core	See Table 2-20.
18DH	397	IA32_PERFEVTSEL7	Core	See Table 2-20.
195H	405	IA32_OVERCLOCKING_STATUS	Package	Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR. See Table 2-2.
1ADH	429	MSR_PRIMARY_TURBO_RATIO_LIMIT	Package	Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.
		7:0		MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.
		15:8		MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.
		23:16		MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.
		31:24		MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.
		39:32		MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.
		47:40		MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		55:48		MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.
		63:56		MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.
493H	1171	IA32_VMX_EXIT_CTLD2		See Table 2-2.
4C7H	1223	IA32_A_PMC6		Full Width Writable IA32_PMC6 Alias (R/W) See Table 2-2.
4C8H	1224	IA32_A_PMC7		Full Width Writable IA32_PMC7 Alias (R/W) See Table 2-2.
650H	1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	Package	Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.
		7:0		MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.
		15:8		MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.
		23:16		MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.
		31:24		MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.
		39:32		MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.
		47:40		MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.
		55:48		MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.
		63:56		MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.
6E1H	1761	IA32_PKRS		Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
776H	1910	IA32_HWP_CTL		See Table 2-2.
981H	2433	IA32_TME_CAPABILITY		Memory Encryption Capability MSR See Table 2-2.
1200H - 121FH	4608 - 4639	IA32_LBR_x_INFO		Last Branch Record Entry X Info Register (R/W) See Table 2-2.
14CEH	5326	IA32_LBR_CTL		Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.
14CFH	5327	IA32_LBR_DEPTH		Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.
1500H - 151FH	5376 - 5407	IA32_LBR_x_FROM_IP		Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.
1600H - 161FH	5632 - 5663	IA32_LBR_x_TO_IP		Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.
17D2H	6098	IA32_THREAD_FEEDBACK_CHAR		Thread Feedback Characteristics (R/O) See Table 2-2.
17D4H	6100	IA32_HW_FEEDBACK_THREAD_CONFIG		Hardware Feedback Thread Configuration (R/W) See Table 2-2.
17DAH	6106	IA32_HRESET_ENABLE		History Reset Enable (R/W) See Table 2-2.

The MSRs listed in Table 2-47 are unique to the 12th and 13th generation Intel Core processor P-core. These MSRs are not supported on the processor E-core.

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1A4H	420	MSR_PREFETCH_CONTROL		Prefetch Disable Bits (R/W)
		0		L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		2		DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3		DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.
		4		Reserved.
		5		AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.
		63:6		Reserved.
3F7H	1015	MSR_PEBS_FRONTEND	Thread	FrontEnd Precise Event Condition Select (R/W) See Table 2-39.
540H	1344	MSR_THREAD_UARCH_CTL	Thread	Thread Microarchitectural Control (R/W)
		0		WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).
		63:1		Reserved
541H	1345	MSR_CORE_UARCH_CTL	Core	Core Microarchitecture Control MSR (R/W) See Table 2-44.
D10H - D17H	3220 - 3351	IA32_L2_QOS_MASK_[0-7]	Core	IA32_CR_L2_QOS_MASK_[0-7] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 18, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”
		19:0		WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.4.2:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.
		31:20		Reserved

The MSRs listed in Table 2-48 are unique to the 12th and 13th generation Intel Core processor E-core. These MSRs are not supported on the processor P-core.

Table 2-48. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor E-core

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D10H - D1FH	3220 - 3359	IA32_L2_QOS_MASK_[0-15]	Module	IA32_CR_L2_QOS_MASK_[0-15] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 18, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features."
		19:0		WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.4.2:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.
		31:20		Reserved
1309H - 130BH	4873 - 4875	MSR_RELOAD_FIXED_CTRx		Reload value for IA32_FIXED_CTRx (R/W)
		47:0		Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.
		63:48		Reserved
14C1H - 14C6H	5313 - 5318	MSR_RELOAD_PMCx	Core	Reload value for IA32_PMCx (R/W)
		47:0		Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.
		63:48		Reserved

Table 2-49 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH.

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).
		63:4		Reserved
2000H	8192	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR
2001H	8193	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
2002H	8194	MSR_UNC_CBO_0_PERFCTRO	Package	Uncore C-Box 0, Performance Counter 0
2003H	8195	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
2008H	8200	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
2009H	8201	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
200AH	8202	MSR_UNC_CBO_1_PERFCTRO	Package	Uncore C-Box 1, Performance Counter 0
200BH	8203	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
2010H	8208	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
2011H	8209	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
2012H	8210	MSR_UNC_CBO_2_PERFCTRO	Package	Uncore C-Box 2, Performance Counter 0
2013H	8211	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
2018H	8216	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
2019H	8217	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
201AH	8218	MSR_UNC_CBO_3_PERFCTRO	Package	Uncore C-Box 3, Performance Counter 0
201BH	8219	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
2020H	8224	MSR_UNC_CBO_4_PERFEVTSELO	Package	Uncore C-Box 4, Counter 0 Event Select MSR
2021H	8225	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
2022H	8226	MSR_UNC_CBO_4_PERFCTRO	Package	Uncore C-Box 4, Performance Counter 0
2023H	8227	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
2028H	8232	MSR_UNC_CBO_5_PERFEVTSELO	Package	Uncore C-Box 5, Counter 0 Event Select MSR
2029H	8233	MSR_UNC_CBO_5_PERFEVTSEL1	Package	Uncore C-Box 5, Counter 1 Event Select MSR
202AH	8234	MSR_UNC_CBO_5_PERFCTRO	Package	Uncore C-Box 5, Performance Counter 0
202BH	8235	MSR_UNC_CBO_5_PERFCTR1	Package	Uncore C-Box 5, Performance Counter 1
2030H	8240	MSR_UNC_CBO_6_PERFEVTSELO	Package	Uncore C-Box 6, Counter 0 Event Select MSR
2031H	8241	MSR_UNC_CBO_6_PERFEVTSEL1	Package	Uncore C-Box 6, Counter 1 Event Select MSR
2032H	8242	MSR_UNC_CBO_6_PERFCTRO	Package	Uncore C-Box 6, Performance Counter 0
2033H	8243	MSR_UNC_CBO_6_PERFCTR1	Package	Uncore C-Box 6, Performance Counter 1
2038H	8248	MSR_UNC_CBO_7_PERFEVTSELO	Package	Uncore C-Box 7, Counter 0 Event Select MSR
2039H	8249	MSR_UNC_CBO_7_PERFEVTSEL1	Package	Uncore C-Box 7, Counter 1 Event Select MSR
203AH	8250	MSR_UNC_CBO_7_PERFCTRO	Package	Uncore C-Box 7, Performance Counter 0
203BH	8251	MSR_UNC_CBO_7_PERFCTR1	Package	Uncore C-Box 7, Performance Counter 1
2040H	8256	MSR_UNC_CBO_8_PERFEVTSELO	Package	Uncore C-Box 8, Counter 0 Event Select MSR
2041H	8257	MSR_UNC_CBO_8_PERFEVTSEL1	Package	Uncore C-Box 8, Counter 1 Event Select MSR
2042H	8258	MSR_UNC_CBO_8_PERFCTRO	Package	Uncore C-Box 8, Performance Counter 0
2043H	8259	MSR_UNC_CBO_8_PERFCTR1	Package	Uncore C-Box 8, Performance Counter 1
2048H	8264	MSR_UNC_CBO_9_PERFEVTSELO	Package	Uncore C-Box 9, Counter 0 Event Select MSR
2049H	8265	MSR_UNC_CBO_9_PERFEVTSEL1	Package	Uncore C-Box 9, Counter 1 Event Select MSR
204AH	8266	MSR_UNC_CBO_9_PERFCTRO	Package	Uncore C-Box 9, Performance Counter 0

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
204BH	8267	MSR_UNC_CBO_9_PERFCTR1	Package	Uncore C-Box 9, Performance Counter 1
2FD0H	12240	MSR_UNC_ARB_0_PERFEVTSELO	Package	Uncore Arb Unit 0, Counter 0 Event Select MSR
2FD1H	12241	MSR_UNC_ARB_0_PERFEVTSEL1	Package	Uncore Arb Unit 0, Counter 1 Event Select MSR
2FD2H	12242	MSR_UNC_ARB_0_PERFCTRO	Package	Uncore Arb Unit 0, Performance Counter 0
2FD3H	12243	MSR_UNC_ARB_0_PERFCTR1	Package	Uncore Arb Unit 0, Performance Counter 1
2FD4H	12244	MSR_UNC_ARB_0_PERF_STATUS	Package	Uncore Arb Unit 0, Performance Status
2FD5H	12245	MSR_UNC_ARB_0_PERF_CTRL	Package	Uncore Arb Unit 0, Performance Control
2FD8H	12248	MSR_UNC_ARB_1_PERFEVTSELO	Package	Uncore Arb Unit 1, Counter 0 Event Select MSR
2FD9H	12249	MSR_UNC_ARB_1_PERFEVTSEL1	Package	Uncore Arb Unit 1, Counter 1 Event Select MSR
2FDAH	12250	MSR_UNC_ARB_1_PERFCTRO	Package	Uncore Arb Unit 1, Performance Counter 0
2FDBH	12251	MSR_UNC_ARB_1_PERFCTR1	Package	Uncore Arb Unit 1, Performance Counter 1
2FDCH	12252	MSR_UNC_ARB_1_PERF_STATUS	Package	Uncore Arb Unit 1, Performance Status
2FDDH	12253	MSR_UNC_ARB_1_PERF_CTRL	Package	Uncore Arb Unit 1, Performance Control
2FDEH	12254	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
2DFDH	12255	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		43:0		Current count.
		63:44		Reserved
2FF0H	12272	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4 select.
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved
2FF2H	12274	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved

2.17.6 MSRs Introduced in the Intel® Xeon® Scalable Processor Family

The Intel® Xeon® Scalable Processor Family (CPUID Signature DisplayFamily_DisplayModel value of 06_55H) supports the MSRs listed in Table 2-50.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)
		18		SGX Global Functions Enable (R/WL)
		20		LMCE_ENABLED (R/WL)
		63:21		Reserved
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/W0) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) See Table 2-26.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		22:16		Reserved.
		23	Package	PPIN_CAP (R/O) See Table 2-26.
		27:24		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.
		30	Package	Programmable TJ OFFSET (R/O) See Table 2-26.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) See Table 2-26.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/WO)
		16		Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).
		24:17		Reserved

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		MCG_EM_P
		26		MCG_ELOG_P
		63:27		Reserved
17DH	381	MSR_SMM_MCA_CAP	THREAD	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and a host-space interface is available to SMM handler.
		59		Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface is available to SMM handler.
		63:60		Reserved
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal Threshold #1 Status (R/O) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.
		10		Power Limitation Status (R/O) See Table 2-2.
		11		Power Limitation Log (R/WCO) See Table 2-2.
		12		Current Limit Status (R/O) See Table 2-2.
		13		Current Limit Log (R/WCO) See Table 2-2.
		14		Cross Domain Limit Status (R/O) See Table 2-2.
		15		Cross Domain Limit Log (R/WCO) See Table 2-2.
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.
		31		Reading Valid (R/O) See Table 2-2.
63:32		Reserved		
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		23:16		Temperature Target (R/O) See Table 2-26.
		27:24		TCC Activation Offset (R/W) See Table 2-26.
		63:28		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	This register defines the ratio limits. RATIO[0:7] must be populated in ascending order. RATIO[i+1] must be less than or equal to RATIO[i]. Entries with RATIO[i] will be ignored. If any of the rules above are broken, the configuration is silently rejected. If the programmed ratio is: <ul style="list-style-type: none"> Above the fused ratio for that core count, it will be clipped to the fuse limits (assuming !OC). Below the min supported ratio, it will be clipped.
		7:0		RATIO_0 Defines ratio limits.
		15:8		RATIO_1 Defines ratio limits.
		23:16		RATIO_2 Defines ratio limits.
		31:24		RATIO_3 Defines ratio limits.
		39:32		RATIO_4 Defines ratio limits.
		47:40		RATIO_5 Defines ratio limits.
		55:48		RATIO_6 Defines ratio limits.
		63:56		RATIO_7 Defines ratio limits.
1AEH	430	MSR_TURBO_RATIO_LIMIT_CORES	Package	This register defines the active core ranges for each frequency point. NUMCORE[0:7] must be populated in ascending order. NUMCORE[i+1] must be greater than NUMCORE[i]. Entries with NUMCORE[i] == 0 will be ignored. The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, the configuration is silently rejected.
		7:0		NUMCORE_0 Defines the active core ranges for each frequency point.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		15:8		NUMCORE_1 Defines the active core ranges for each frequency point.
		23:16		NUMCORE_2 Defines the active core ranges for each frequency point.
		31:24		NUMCORE_3 Defines the active core ranges for each frequency point.
		39:32		NUMCORE_4 Defines the active core ranges for each frequency point.
		47:40		NUMCORE_5 Defines the active core ranges for each frequency point.
		55:48		NUMCORE_6 Defines the active core ranges for each frequency point.
		63:56		NUMCORE_7 Defines the active core ranges for each frequency point.
280H	640	IA32_MC0_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
400H	1024	IA32_MC0_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC0 reports MC errors from the IFU module.
401H	1025	IA32_MC0_STATUS	Core	
402H	1026	IA32_MC0_ADDR	Core	
403H	1027	IA32_MC0_MISC	Core	
404H	1028	IA32_MC1_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.
405H	1029	IA32_MC1_STATUS	Core	
406H	1030	IA32_MC1_ADDR	Core	
407H	1031	IA32_MC1_MISC	Core	
408H	1032	IA32_MC2_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.
409H	1033	IA32_MC2_STATUS	Core	
40AH	1034	IA32_MC2_ADDR	Core	
40BH	1035	IA32_MC2_MISC	Core	
40CH	1036	IA32_MC3_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.
40DH	1037	IA32_MC3_STATUS	Core	
40EH	1038	IA32_MC3_ADDR	Core	
40FH	1039	IA32_MC3_MISC	Core	
410H	1040	IA32_MC4_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.
411H	1041	IA32_MC4_STATUS	Package	
412H	1042	IA32_MC4_ADDR	Package	
413H	1043	IA32_MC4_MISC	Package	
414H	1044	IA32_MC5_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
420H	1056	IA32_MC8_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
440H	1088	IA32_MC16_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 15.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 15.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
C8DH	3213	IA32_QM_EVTSEL	THREAD	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.
		31:8		Reserved
		41:32		RMID (R/W)
		63:42		Reserved
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		31:10		Reserved
		51:32		COS (R/W)
		63: 52		Reserved
C90H	3216	IA32_L3_QOS_MASK_0	Package	L3 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:19		CBM: Bit vector of available L3 ways for COS 0 enforcement.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:20		Reserved
C91H	3217	IA32_L3_QOS_MASK_1	Package	L3 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:19		CBM: Bit vector of available L3 ways for COS 1 enforcement.
		63:20		Reserved
C92H	3218	IA32_L3_QOS_MASK_2	Package	L3 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:19		CBM: Bit vector of available L3 ways for COS 2 enforcement.
		63:20		Reserved
C93H	3219	IA32_L3_QOS_MASK_3	Package	L3 Class Of Service Mask - COS 3 (R/W). If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L3 ways for COS 3 enforcement.
		63:20		Reserved
C94H	3220	IA32_L3_QOS_MASK_4	Package	L3 Class Of Service Mask - COS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.
		0:19		CBM: Bit vector of available L3 ways for COS 4 enforcement.
		63:20		Reserved
C95H	3221	IA32_L3_QOS_MASK_5	Package	L3 Class Of Service Mask - COS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.
		0:19		CBM: Bit vector of available L3 ways for COS 5 enforcement.
		63:20		Reserved
C96H	3222	IA32_L3_QOS_MASK_6	Package	L3 Class Of Service Mask - COS 6 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6.
		0:19		CBM: Bit vector of available L3 ways for COS 6 enforcement.
		63:20		Reserved
C97H	3223	IA32_L3_QOS_MASK_7	Package	L3 Class Of Service Mask - COS 7 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7.
		0:19		CBM: Bit vector of available L3 ways for COS 7 enforcement.
		63:20		Reserved
C98H	3224	IA32_L3_QOS_MASK_8	Package	L3 Class Of Service Mask - COS 8 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8.
		0:19		CBM: Bit vector of available L3 ways for COS 8 enforcement.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:20		Reserved
C99H	3225	IA32_L3_QOS_MASK_9	Package	L3 Class Of Service Mask - COS 9 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9.
		0:19		CBM: Bit vector of available L3 ways for COS 9 enforcement.
		63:20		Reserved
C9AH	3226	IA32_L3_QOS_MASK_10	Package	L3 Class Of Service Mask - COS 10 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10.
		0:19		CBM: Bit vector of available L3 ways for COS 10 enforcement.
		63:20		Reserved
C9BH	3227	IA32_L3_QOS_MASK_11	Package	L3 Class Of Service Mask - COS 11 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11.
		0:19		CBM: Bit vector of available L3 ways for COS 11 enforcement.
		63:20		Reserved
C9CH	3228	IA32_L3_QOS_MASK_12	Package	L3 Class Of Service Mask - COS 12 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12.
		0:19		CBM: Bit vector of available L3 ways for COS 12 enforcement.
		63:20		Reserved
C9DH	3229	IA32_L3_QOS_MASK_13	Package	L3 Class Of Service Mask - COS 13 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13.
		0:19		CBM: Bit vector of available L3 ways for COS 13 enforcement.
		63:20		Reserved
C9EH	3230	IA32_L3_QOS_MASK_14	Package	L3 Class Of Service Mask - COS 14 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14.
		0:19		CBM: Bit vector of available L3 ways for COS 14 enforcement.
		63:20		Reserved
C9FH	3231	IA32_L3_QOS_MASK_15	Package	L3 Class Of Service Mask - COS 15 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15.
		0:19		CBM: Bit vector of available L3 ways for COS 15 enforcement.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:20		Reserved

2.17.7 MSRs Specific to 3rd Generation Intel® Xeon® Scalable Processor Family Based on Ice Lake Microarchitecture

The 3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH) support the MSRs listed in Table 2-51.

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
612H	1554	MSR_PACKAGE_ENERGY_TIME_STATUS	Package	Package energy consumed by the entire CPU (R/W)
		31:0		Total amount of energy consumed since last reset.
		63:32		Total time elapsed when the energy was last updated. This is a monotonic increment counter with auto wrap back to zero after overflow. Unit is 10ns.
618H	1560	MSR_DRAM_POWER_LIMIT	Package	Allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.
		14:0		DRAM_PP_PWR_LIM: Power Limit[0] for DDR domain. Units = Watts, Format = 11.3, Resolution = 0.125W, Range = 0-2047.875W.
		15		PWR_LIM_CTRL_EN: Power Limit[0] enable bit for DDR domain.
		16		Reserved
		23:17		CTRL_TIME_WIN: Power Limit[0] time window Y value, for DDR domain. Actual time_window for RAPL is: $(1/1024 \text{ seconds}) * (1+(x/4)) * (2^y)$
		62:24		Reserved
		63		PP_PWR_LIM_LOCK: When set, this entire register becomes read-only. This bit will typically be set by BIOS during boot.
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM Power Parameters (R/W)
		14:0		Spec DRAM Power (DRAM_TDP): The Spec power allowed for DRAM. The TDP setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].
		15		Reserved
		30:16		Minimal DRAM Power (DRAM_MIN_PWR): The minimal power setting allowed for DRAM. Lower values will be clamped to this value. The minimum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].
		31		Reserved
		46:32		Maximal Package Power (DRAM_MAX_PWR): The maximal power setting allowed for DRAM. Higher values will be clamped to this value. The maximum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].
		47		Reserved
		54:48		Maximal Time Window (DRAM_MAX_WIN): The maximal time window allowed for the DRAM. Higher values will be clamped to this value. x = PKG_MAX_WIN[54:53] y = PKG_MAX_WIN[52:48] The timing interval window is a floating-point number given by 1.x *power(2,y). The unit of measurement is defined in MSR_DRAM_POWER_INFO_UNIT[TIME_UNIT].
		62:55		Reserved
		63		LOCK: Lock bit to lock the register.
981H	2433	IA32_TME_CAPABILITY		See Table 2-2.
982H	2434	IA32_TME_ACTIVATE		See Table 2-2.
983H	2435	IA32_TME_EXCLUDE_MASK		See Table 2-2.
984H	2436	IA32_TME_EXCLUDE_BASE		See Table 2-2.

2.17.8 MSRs Specific to the 4th Generation Intel® Xeon® Scalable Processor Family Based on Sapphire Rapids Microarchitecture

The 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_8FH) supports the MSRs listed in Section 2.17, “MSRs In the 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, 12th Generation, and 13th Generation Intel® Core™ Processors, Intel® Xeon® Scalable Processor Family, 2nd, 3rd, and 4th Generation Intel® Xeon® Scalable Processor Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E processors,” including Table 2-52. For an MSR listed in Table 2-52 that also appears in the model-specific tables of prior generations, Table 2-52 supersedes prior generation tables.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register (R/W)
		27:0		Reserved.
		28		UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 9.1.2.3, “Features to Disable Bus Locks.”
		29		SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 9.1.2.3, “Features to Disable Bus Locks.”
		31:30		Reserved.
A7H	167	MSR_BIOS_DEBUG	Thread	BIOS DEBUG (R/O) See Table 2-45.
BCH	188	IA32_MISC_PACKAGE_CTL	Package	Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.
CFH	207	IA32_CORE_CAPABILITIES	Core	IA32 Core Capabilities Register (R/W) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		0		Reserved: returns zero.
		1		Reserved: returns zero.
		2		INTEGRITY_CAPABILITIES When set to 1, the processor supports MSR_INTEGRITY_CAPABILITIES.
		3		RSM_IN_CPLD_ONLY Indicates that RSM will only be allowed in CPLD and will #GP for all non-CPLD privilege levels.
		4		UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5		SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.
		6		Reserved: returns zero.
		7		UC_STORE_THROTTLING_SUPPORTED Indicates that the snoop filter quality of service MSRs are supported on this core. This is based on the existence of a non-inclusive cache and the L2/MLC QoS feature supported.
		63:8		Reserved: returns zero.
E1H	225	IA32_UMWAIT_CONTROL		UMWAIT Control (R/W) See Table 2-2.
EDH	237	MSR_RAR_CONTROL	Thread	RAR Control (R/W)
		63:32		Reserved.
		31		ENABLE RAR events are recognized. When RAR is not enabled, RARs are dropped.
		30		IGNORE_IF Allow RAR servicing at the RLP regardless of the value of RFLAGS.IF.
		29:0		Reserved.
EEH	238	MSR_RAR_ACTION_VECTOR_BASE	Thread	Pointer to RAR Action Vector (R/W)
		63:MAXPHYADDR		Reserved.
		MAXPHYADDR-1:6		VECTOR_PHYSICAL_ADDRESS Pointer to the physical address of the 64B aligned RAR action vector.
		5:0		Reserved.
EFH	239	MSR_RAR_PAYLOAD_TABLE_BASE	Thread	Pointer to Base of RAR Payload Table (R/W)
		63:MAXPHYADDR		Reserved.
		MAXPHYADDR-1:12		TABLE_PHYSICAL_ADDRESS Pointer to the base physical address of the 4K aligned RAR payload table.
		11:0		Reserved.
FOH	240	MSR_RAR_INFO	Thread	Read Only RAR Information (RO)
		63:38		Always zero.
		37:32		Table Max Index Maximum supported payload table index.
		31:0		Supported payload type bitmap. A value of 1 in bit position [i] indicates that payload type [i] is supported.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
105H	261	MSR_CORE_BIST	Core	Core BIST (R/W) Controls Array BIST activation and status checking as part of FUSA.
		31:0		BIST_ARRAY Bitmap indicating which arrays to run BIST on (WRITE). Bitmap indicating which arrays were not processed, i.e., completion mask (READ).
		39:32		BANK Array bank of the [least significant set bit] array indicated in EAX to start BIST(WRITE). Array bank interrupted or failed (READ).
		47:40		DWORD Array dword of the [least significant set bit] array indicated in EAX to start BIST (WRITE). Array dword interrupted or failed (READ).
		62:48		Reserved
		63		CTRL_RESULT Indicates whether WRMSR should signal Machine-Check upon BIST-error (WRITE). BIST result PASS(0)/FAIL(1) of the (least significant set bit) array indicated in EAX (READ).
10AH	266	IA32_ARCH_CAPABILITIES		Enumeration of Architectural Features (R/O) See Table 2-2.
1A4H	420	MSR_PREFETCH_CONTROL		Prefetch Disable Bits (R/W)
		0		L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2		DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3		DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.
		4		Reserved.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5		AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.
		63:6		Reserved.
1ADH	429	MSR_PRIMARY_TURBO_RATIO_LIMIT	Package	Primary Maximum Turbo Ratio Limit (R/W) See Table 2-46.
1AEH	430	MSR_TURBO_RATIO_LIMIT_CORES	Package	See Table 2-50.
1C4H	452	IA32_XFD		Extended Feature Detect (R/W) See Table 2-2.
1C5H	453	IA32_XFD_ERR		XFD Error Code (R/W) See Table 2-2.
2C2H	706	MSR_COPY_SCAN_HASHES	Die	COPY_SCAN_HASHES (w)
		63:0		SCAN_HASH_ADDR Contains the linear address of the SCAN Test HASH Binary loaded into memory.
2C3H	707	MSR_SCAN_HASHES_STATUS		SCAN_HASHES_STATUS (R/O)
		15:0	Die	CHUNK_SIZE Chunk size of the test in KB.
		23:16	Die	NUM_CHUNKS Total number of chunks.
		31:24		Reserved: all zeros.
		39:32	Thread	ERROR_CODE The error-code refers to the LP that runs WRMSR(2C2H). 0x0: No error reported. 0x1: Attempt to copy scan-hashes when copy already in progress. 0x2: Secure Memory not set up correctly. 0x3: Scan-image header Image_info.ProgramID doesn't match RDMSR(2D9H)[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4: Reserved 0x5: Integrity check failed. 0x6: Re-install of scan test image attempted when current scan test image is in use by other LPs.
		50:40		Reserved: set to all zeros.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		62:51	Die	MAX_CORE_LIMIT Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.
		63	Die	Valid Valid bit is set when COPY_SCAN_HASHES has completed successfully.
2C4H	708	MSR_AUTHENTICATE_AND_COPY_CHUNK	Die	AUTHENTICATE_AND_COPY_CHUNK (W)
		7:0		CHUNK_INDEX Chunk Index, should be less than the total number of chunks defined by NUM_CHUNKS (MSR_SCAN_HASHES_STATUS[23:16]).
		63:8		CHUNK_ADDR Bits 63:8 of 256B aligned Linear address of scan chunk in memory.
2C5H	709	MSR_CHUNKS_AUTHENTICATION_STATUS		CHUNKS_AUTHENTICATION_STATUS (R/O)
		7:0	Die	VALID_CHUNKS Total number of Valid (authenticated) chunks.
		15:8	Die	TOTAL_CHUNKS Total number of chunks.
		31:16		Reserved: all zeros.
		39:32	Thread	ERROR_CODE The error code refers to the LP that runs WRMSR(2C4H). 0x0: No error reported. 0x1: Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. 0x2: CHUNK authentication error. HASH of chunk did not match expected value.
		63:40		Reserved: set to all zeros.
2C6H	710	MSR_ACTIVATE_SCAN	Thread	ACTIVATE_SCAN (W)
		7:0		CHUNK_START_INDEX Indicates chunk index to start from.
		15:8		CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive).
		31:16		Reserved: all zeros.
		62:32		THREAD_WAIT_DELAY TSC based delay to allow threads to rendezvous.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63		SIGNAL_MCE If 1, then on scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. If 0, then no logging/no signaling.
2C7H	711	MSR_SCAN_STATUS		SCAN_STATUS (R/O)
		7:0	Core	CHUNK_NUM SCAN Chunk that was reached.
		15:8	Core	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive). Maps to same field in WRMSR(ACTIVATE_SCAN).
		31:16		Reserved: return all zeros.
		39:32	Thread	ERROR_CODE 0x0: No error. 0x1: SCAN operation did not start. Other thread did not join in time. 0x2: SCAN operation did not start. Interrupt occurred prior to threads rendezvous. 0x3: SCAN operation did not start. Power Management conditions are inadequate to run Intel In-field Scan. 0x4: SCAN operation did not start. Non-valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5: SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6: SCAN operation did not start. Core not capable of performing SCAN currently. 0x8: SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Intel In-field Scan concurrently. MAX_CORE_LIMIT exceeded. 0x9: Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed.
		61:40		Reserved: return all zeros.
		62	Core	SCAN_CONTROL_ERROR Scan-System-Controller malfunction.
		63	Core	SCAN_SIGNATURE_ERROR Core failed SCAN-SIGNATURE checking for this chunk.
2C8H	712	MSR_SCAN_MODULE_ID	Module	SCAN_MODULE_ID (R/O)
		31:0		RevID of the currently installed scan test image. Maps to Revision field in external header (offset 4).
		63:32		Reserved: return all zeros.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
2C9H	713	MSR_LAST_SAF_WP	Core	LAST_SAF_WP (R/O)
		31:0		LAST_WP Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. Available only if enumerated in MSR_INTEGRITY_CAPABILITIES[10:9].
		63:32		Reserved: return all zeros.
2D9H	729	MSR_INTEGRITY_CAPABILITIES	Module	INTEGRITY_CAPABILITIES (R/O)
		0		STARTUP_SCAN_BIST When set, supports Intel In-field Scan.
		3:1		Reserved: return all zeros.
		4		PERIODIC_SCAN_BIST When set, supports Intel In-field Scan.
		23:5		Reserved: return all zeros.
		31:24		ID of the scan programs supported for this part. WRMSR(2C2H) verifies this value against the corresponding value in the scan-image header, i.e., Image_info.
410H	1040	IA32_MC4_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs," through Section 16.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.
411H	1041	IA32_MC4_STATUS	Package	
412H	1042	IA32_MC4_ADDR	Package	
413H	1043	IA32_MC4_MISC	Package	
492H	1170	IA32_VMX_PROCBASED_CTL3		Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Table 2-2.
493H	1171	IA32_VMX_EXIT_CTL2		Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Table 2-2.
540H	1344	MSR_THREAD_UARCH_CTL	Thread	Thread Microarchitectural Control (R/W) See Table 2-47.
65CH	1628	MSR_PLATFORM_POWER_LIMIT	Platform	Platform Power Limit Control (R/W-L) See Table 2-39.
6A0H	1696	IA32_U_CET		Configure User Mode CET (R/W) See Table 2-2.
6A2H	1698	IA32_S_CET		Configure Supervisor Mode CET (R/W) See Table 2-2.
6A4H	1700	IA32_PLO_SSP		Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6A5H	1701	IA32_PL1_SSP		Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.
6A6H	1702	IA32_PL2_SSP		Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.
6A7H	1703	IA32_PL3_SSP		Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.
6A8H	1704	IA32_INTERRUPT_SSP_TABLE_ADDR		Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.
6E1H	1761	IA32_PKRS		Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.
776H	1910	IA32_HWP_CTL		See Table 2-2.
981H	2433	IA32_TME_CAPABILITY		Memory Encryption Capability MSR See Table 2-2.
985H	2437	IA32_UINTR_RR		User Interrupt Request Register (R/W) See Table 2-2.
986H	2438	IA32_UINTR_HANDLER		User Interrupt Handler Address (R/W) See Table 2-2.
987H	2439	IA32_UINTR_STACKADJUST		User Interrupt Stack Adjustment (R/W) See Table 2-2.
988H	2440	IA32_UINTR_MISC		User-Interrupt Target-Table Size and Notification Vector (R/W) See Table 2-2.
989H	2441	IA32_UINTR_PD		User Interrupt PID Address (R/W) See Table 2-2.
98AH	2442	IA32_UINTR_TT		User-Interrupt Target Table (R/W) See Table 2-2.
C70H	3184	MSR_B1_PMON_EVNT_SELO	Package	Uncore B-box 1 perfmon event select MSR.
C71H	3185	MSR_B1_PMON_CTR0	Package	Uncore B-box 1 perfmon counter MSR.
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR.
C73H	3187	MSR_B1_PMON_CTR1	Package	Uncore B-box 1 perfmon counter MSR.

Table 2-52. Additional MSRs Supported by the 4th Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C74H	3188	MSR_B1_PMON_EVNT_SEL2	Package	Uncore B-box 1 perfmon event select MSR.
C75H	3189	MSR_B1_PMON_CTR2	Package	Uncore B-box 1 perfmon counter MSR.
C76H	3190	MSR_B1_PMON_EVNT_SEL3	Package	Uncore B-box 1 vperfmon event select MSR.
C77H	3191	MSR_B1_PMON_CTR3	Package	Uncore B-box 1 perfmon counter MSR.
C82H	3122	MSR_W_PMON_BOX_OVF_CTRL	Package	Uncore W-box perfmon local box overflow control MSR.
C8FH	3215	IA32_PQR_ASSOC		See Table 2-2.
C90H - C9EH	3216 - 3230	IA32_L3_QOS_MASK_0 through IA32_L3_QOS_MASK_14	Package	See Table 2-50.
D10H - D17H	3344 - 3351	IA32_L2_QOS_MASK_[0-7]	Core	IA32_CR_L2_QOS_MASK_[0-7] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. See Table 2-2.
D93H	3475	IA32_PASID		See Table 2-2.
1200H - 121FH	4608 - 4639	IA32_LBR_x_INFO		Last Branch Record Entry X Info Register (R/W) See Table 2-2.
1406H	5126	IA32_MCU_CONTROL		See Table 2-2.
14CEH	5326	IA32_LBR_CTL		Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.
14CFH	5327	IA32_LBR_DEPTH		Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.
1500H - 151FH	5376 - 5407	IA32_LBR_x_FROM_IP		Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.
1600H - 161FH	5632 - 5663	IA32_LBR_x_TO_IP		Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.

2.18 MSRS IN THE INTEL® XEON PHI™ PROCESSOR 3200/5200/7200 SERIES AND THE INTEL® XEON PHI™ PROCESSOR 7215/7285/7295 SERIES

The Intel® Xeon Phi™ processor 3200, 5200, 7200 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_57H, supports the MSR interfaces listed in Table 2-53. These processors are based on the Knights Landing microarchitecture. The Intel® Xeon Phi™ processor 7215, 7285, 7295 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_85H, supports the MSR interfaces listed in Table 2-53 and Table 2-54.

These processors are based on the Knights Mill microarchitecture. Some MSRs are shared between a pair of processor cores, and the scope is marked as module.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 9.10.5, "Monitor/Mwait Address Range Determination." See Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 18.17, "Time-Stamp Counter," and Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O)
		63:32		Reserved
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Reserved
		2		Enable VMX outside SMX operation (R/WL)
3BH	59	IA32_TSC_ADJUST	THREAD	Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	THREAD	BIOS Update Signature ID (R/W) See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
C1H	193	IA32_PMC0	THREAD	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	THREAD	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Package	C-State Configuration Control (R/W)

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		2:0		<p>Package C-State Limit (R/W)</p> <p>Specifies the lowest C-state for the package. This feature does not limit the processor core C-state. The power-on default value from bit[2:0] of this register reports the deepest package C-state the processor is capable to support when manufactured. It is recommended that BIOS always read the power-on default value reported from this bit field to determine the supported deepest C-state on the processor and leave it as default without changing it.</p> <p>000b - C0/C1 (No package C-state support) 001b - C2 010b - C6 (non retention)* 011b - C6 (Retention)* 100b - Reserved 101b - Reserved 110b - Reserved 111b - No package C-state limit. All C-States supported by the processor are available.</p> <p>Note: C6 retention mode provides more power saving than C6 non-retention mode. Limiting the package to C6 non retention mode does prevent the MSR_PKG_C6_RESIDENCY counter (MSR 3F9h) from being incremented.</p>
		9:3		Reserved
		10		<p>I/O MWAIT Redirection Enable (R/W)</p> <p>When set, will map IO_read instructions sent to IO registers at MSR_PMG_IO_CAPTURE_BASE[15:0] to MWAIT instructions.</p>
		14:11		Reserved
		15		<p>CFG Lock (R/O)</p> <p>When set, locks bits [15:0] of this register for further writes until the next reset occurs.</p>
		25		Reserved
		26		<p>C1 State Auto Demotion Enable (R/W)</p> <p>When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.</p>
		27		Reserved
		28		<p>C1 State Auto Undemotion Enable (R/W)</p> <p>When set, enables Undemotion from Demoted C1.</p>
		29		<p>PKG C-State Auto Demotion Enable (R/W)</p> <p>When set, enables Package C state demotion.</p>
		63:30		Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Tile	Power Management IO Capture Base (R/W)
		15:0		LVL_2 Base Address (R/W) Microcode will compare IO-read zone to this base address to determine if an MWAIT(C2/3/4) needs to be issued instead of the IO-read. Should be programmed to the chipset Plevel_2 IO address.
		22:16		C-State Range (R/W) The IO-port block size in which IO-redirection will be executed (0-127). Should be programmed based on the number of LVLx registers existing in the chipset.
		63:23		Reserved
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Core	Memory Type Range Register (R) See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, the AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
140H	320	MISC_FEATURE_ENABLES	Thread	MISC_FEATURE_ENABLES
		0		Reserved
		1		User Mode MONITOR and MWAIT (R/W) If set to 1, the MONITOR and MWAIT instructions do not cause invalid-opcode exceptions when executed with CPL > 0 or in virtual-8086 mode. If MWAIT is executed when CPL > 0 or in virtual-8086 mode, and if EAX indicates a C-state other than C0 or C1, the instruction operates as if EAX indicated the C-state C1.
		63:2		Reserved
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	See Table 2-2.
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		31:0		Bank Support (SMM-RO) One bit per MCA bank. If the bit is set, that bank supports Enhanced MCA (Default all 0; does not support EMCA).
		55:32		Reserved
		56		Targeted SMI (SMM-RO) Set if targeted SMI is supported.
		57		SMM_CPU_SVRSTR (SMM-RO) Set if SMM SRAM save/restore feature is supported.
		58		SMM_CODE_ACCESS_CHK (SMM-RO) Set if SMM code access check feature is supported.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
186H	390	IA32_PERFEVTSELO	Thread	Performance Monitoring Event Select Register (R/W) See Table 2-2.
		7:0		Event Select
		15:8		UMask
		16		USR
		17		OS
		18		Edge
		19		PC
		20		INT
		21		AnyThread
		22		EN
		23		INV
		31:24		CMASK
		63:32		Reserved
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Module	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Module	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O)
		1		Thermal Status Log (R/WCO)
		2		PROTCHOT # or FORCEPR# Status (R/O)
		3		PROTCHOT # or FORCEPR# Log (R/WCO)
		4		Critical Temperature Status (R/O)
		5		Critical Temperature Status Log (R/WCO)
		6		Thermal Threshold #1 Status (R/O)
		7		Thermal Threshold #1 Log (R/WCO)
		8		Thermal Threshold #2 Status (R/O)
		9		Thermal Threshold #2 Log (R/WCO)
		10		Power Limitation Status (R/O)
		11		Power Limitation Log (R/WCO)
		15:12		Reserved
		22:16		Digital Readout (R/O)
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O)
31		Reading Valid (R/O)		
63:32		Reserved		
1A0H	416	IA32_MISC_ENABLE	Thread	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0		Fast-Strings Enable
		2:1		Reserved
		3		Automatic Thermal Control Circuit Enable (R/W)
		6:4		Reserved
		7		Performance Monitoring Available (R)
		10:8		Reserved
		11		Branch Trace Storage Unavailable (R/O)
		12		Processor Event Based Sampling Unavailable (R/O)
15:13		Reserved		

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		16		Enhanced Intel SpeedStep Technology Enable (R/W)
		18		ENABLE MONITOR FSM (R/W)
		21:19		Reserved
		22		Limit CPUID Maxval (R/W)
		23		xTPR Message Disable (R/W)
		33:24		Reserved
		34		XD Bit Disable (R/W)
		37:35		Reserved
		38		Turbo Mode Disable (R/W)
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R)
		29:24		Target Offset (R/W)
		63:30		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher.
		1	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher.
		63:2		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Shared	Offcore Response Event Select Register (R/W)
1A7H	423	MSR_OFFCORE_RSP_1	Shared	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode for Groups of Cores (R/W)
		0		Reserved
		7:1	Package	Maximum Number of Cores in Group 0 Number active processor cores which operates under the maximum ratio limit for group 0.
		15:8	Package	Maximum Ratio Limit for Group 0 Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.
		20:16	Package	Number of Incremental Cores Added to Group 1 Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		23:21	Package	Group Ratio Delta for Group 1 An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.
		28:24	Package	Number of Incremental Cores Added to Group 2 Group 2, which includes the specified number of additional cores plus all the cores in group 1, operates under the group 2 turbo max ratio limit = "group 1 Max ratio limit" - "group ratio delta for group 2".
		31:29	Package	Group Ratio Delta for Group 2 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.
		36:32	Package	Number of Incremental Cores Added to Group 3 Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".
		39:37	Package	Group Ratio Delta for Group 3 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.
		44:40	Package	Number of Incremental Cores Added to Group 4 Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".
		47:45	Package	Group Ratio Delta for Group 4 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.
		52:48	Package	Number of Incremental Cores Added to Group 5 Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".
		55:53	Package	Group Ratio Delta for Group 5 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.
		60:56	Package	Number of Incremental Cores Added to Group 6 Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".
		63:61	Package	Group Ratio Delta for Group 6 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.
1B0H	432	IA32_ENERGY_PERF_BIAS	Thread	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
63:9		Reserved		
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W)
		0		LBR Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.
		1		BTF Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.
		5:2		Reserved
		6		TR Setting this bit to 1 enables branch trace messages to be sent.
		7		BTS Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.
		8		BTINT When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.
9		BTS_OFF_OS When set, BTS or BTM is skipped if CPL = 0.		

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		10		BTS_OFF_USR When set, BTS or BTM is skipped if CPL > 0.
		11		FREEZE_LBRS_ON_PMI When set, the LBR stack is frozen on a PMI request.
		12		FREEZE_PERFMON_ON_PMI When set, each ENABLE bit of the global counter control MSR are frozen (address 3BFH) on a PMI request.
		13		Reserved
		14		FREEZE_WHILE_SMM When set, freezes perfmon and trace messages while in SMM.
		31:15		Reserved
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record from Linear IP (R)
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record to Linear IP (R)
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Package	See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2.
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Table 2-2.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C3 Residency Counter (R/O)
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	
		63:0		Package C6 Residency Counter (R/O)
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	
		63:0		Package C7 Residency Counter (R/O)
3FCH	1020	MSR_MCO_RESIDENCY	Module	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Module C0 Residency Counter (R/O)
3FDH	1021	MSR_MC6_RESIDENCY	Module	
		63:0		Module C6 Residency Counter (R/O)

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3FFH	1023	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		CORE C6 Residency Counter (R/O)
400H	1024	IA32_MCO_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR	Core	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
414H	1044	IA32_MC5_CTL	Package	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	IA32_MC5_ADDR	Package	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 15.10.1, "RAPL Interfaces."
		7:4	Package	Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 15.10.1, "RAPL Interfaces."
		63:20		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C2 Residency Counter (R/O)
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 15.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W) See Section 15.10.3, "Package RAPL Domain."
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."
638H	1592	MSR_PPO_POWER_LIMIT	Package	PPO RAPL Power Limit Control (R/W) See Section 15.10.4, "PPO/PP1 RAPL Domains."
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 15.10.4, "PPO/PP1 RAPL Domains."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O) See Table 2-25.
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O) See Table 2-25.
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O) See Table 2-25.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W) See Table 2-25.
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W) See Table 2-25.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (RO)
		1		Thermal Status (RO)
		5:2		Reserved
		6		VR Therm Alert Status (RO)
		7		Reserved
		8		Electrical Design Point Status (RO)
		63:9		Reserved
6E0H	1760	IA32_TSC_DEADLINE	Core	TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID Register (R/O)
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version Register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority Register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority Register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI Register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination Register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector Register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service Register Bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service Register Bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service Register Bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service Register Bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service Register Bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service Register Bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service Register Bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service Register Bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode Register Bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode Register Bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode Register Bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode Register Bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode Register Bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode Register Bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode Register Bits [223:192] (R/O)

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode Register Bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request Register Bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request Register Bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request Register Bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request Register Bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request Register Bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request Register Bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request Register Bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request Register Bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status Register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command Register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt Register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt Register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor Register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 Register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 Register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error Register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count Register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count Register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration Register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI Register (W/O)
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2

Table 2-54 lists model-specific registers that are supported by the Intel® Xeon Phi™ processor 7215, 7285, 7295 series based on the Knights Mill microarchitecture.

Table 2-54. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
9BH	155	IA32_SMM_MONITOR_CTL	Core	SMM Monitor Configuration (R/W) This MSR is readable only if VMX is enabled, and writeable only if VMX is enabled and in SMM mode, and is used to configure the VMX MSEG base address. See Table 2-2.
480H	1152	IA32_VMX_BASIC	Core	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.
481H	1153	IA32_VMX_PINBASED_CTL	Core	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.
482H	1154	IA32_VMX_PROCBASED_CTL	Core	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)
483H	1155	IA32_VMX_EXIT_CTL	Core	Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.
484H	1156	IA32_VMX_ENTRY_CTL	Core	Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.
485H	1157	IA32_VMX_MISC	Core	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.
486H	1158	IA32_VMX_CR0_FIXED0	Core	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2.
487H	1159	IA32_VMX_CR0_FIXED1	Core	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2.
488H	1160	IA32_VMX_CR4_FIXED0	Core	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	Core	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.

Table 2-54. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
48AH	1162	IA32_VMX_VMCS_ENUM	Core	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.
48BH	1163	IA32_VMX_PROCBASED_CTLSS2	Core	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Table 2-2.
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLSS	Core	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLSS	Core	Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2.
48FH	1167	IA32_VMX_TRUE_EXIT_CTLSS	Core	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.
490H	1168	IA32_VMX_TRUE_ENTRY_CTLSS	Core	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.
491H	1169	IA32_VMX_FMFUNC	Core	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.

2.19 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-55 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table 2-1.

- MSRs with an "IA32_" prefix are designated as "architectural." This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an "MSR_" prefix are model specific with respect to address functionalities. The column "Model Availability" lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 2.23, "MSRs in Pentium Processors."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_LINE_SIZE	3, 4, 6	Shared	See Section 9.10.5, "Monitor/Mwait Address Range Determination."
10H	16	IA32_TIME_STAMP_COUNTER	0, 1, 2, 3, 4, 6	Unique	Time Stamp Counter See Table 2-2.
					On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3, 4, 6	Shared	Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	0, 1, 2, 3, 4, 6	Unique	APIC Location and Status (R/W) See Table 2-2. See Section 11.4.4, "Local APIC Status and Location."
2AH	42	MSR_EBC_HARD_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.
		0			Output Tri-state Enabled (R) Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		1			Execute BIST (R) Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		2			In Order Queue Depth (R) Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
		3			MCERR# Observation Disabled (R) Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		4			BINIT# Observation Enabled (R) Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		6:5			APIC Cluster ID (R) Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		7			Bus Park Disable (R) Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		11:8			Reserved
		13:12			Agent ID (R) Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		63:14			Reserved
2BH	43	MSR_EBC_SOFT_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Soft Power-On Configuration (R/W) Enables and disables processor features.
		0			RCNT/SCNT On Request Encoding Enable (R/W) Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).
		1			Data Error Checking Disable (R/W) Set to disable system data bus parity checking; clear to enable parity checking.
		2			Response Error Checking Disable (R/W) Set to disable (default); clear to enable.
		3			Address/Request Error Checking Disable (R/W) Set to disable (default); clear to enable.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		4			Initiator MCERR# Disable (R/W) Set to disable MCERR# driving for initiator bus requests (default); clear to enable.
		5			Internal MCERR# Disable (R/W) Set to disable MCERR# driving for initiator internal errors (default); clear to enable.
		6			BINIT# Driver Disable (R/W) Set to disable BINIT# driver (default); clear to enable driver.
		63:7			Reserved
2CH	44	MSR_EBC_FREQUENCY_ID	2,3, 4, 6	Shared	Processor Frequency Configuration The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2. (R) The field Indicates the current processor frequency configuration.
		15:0			Reserved
		18:16			Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6)
					133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
					266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.
		23:19			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		31:24			Core Clock Frequency to System Bus Frequency Ratio (R) The processor core clock frequency to system bus frequency ratio observed at the deassertion of the reset pin.
		63:25			Reserved
2CH	44	MSR_EBC_FREQUENCY_ID	0, 1	Shared	Processor Frequency Configuration (R) The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.
		20:0			Reserved
		23:21			Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz All others values reserved.
		63:24			Reserved
3AH	58	IA32_FEATURE_CONTROL	3, 4, 6	Unique	Control Features in IA-32 Processor (R/W) See Table 2-2. (If CPUID.01H:ECX.[bit 5])
79H	121	IA32_BIOS_UPDT_TRIG	0, 1, 2, 3, 4, 6	Shared	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
9BH	155	IA32_SMM_MONITOR_CTL	3, 4, 6	Unique	SMM Monitor Configuration (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	0, 1, 2, 3, 4, 6	Unique	MTRR Information See Section 12.11.1, "MTRR Feature Identification."
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3, 4, 6	Unique	CS Register Target for CPL 0 Code (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures With the SYSENTER and SYSEXIT Instructions."
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3, 4, 6	Unique	Stack Pointer for CPL 0 Stack (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures With the SYSENTER and SYSEXIT Instructions."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	CPL 0 Code Entry Point (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures With the SYSENTER and SYSEXIT Instructions."
179H	377	IA32_MCG_CAP	0, 1, 2, 3, 4, 6	Unique	Machine Check Capabilities (R) See Table 2-2. See Section 16.3.1.1, "IA32_MCG_CAP MSR."
17AH	378	IA32_MCG_STATUS	0, 1, 2, 3, 4, 6	Unique	Machine Check Status (R) See Table 2-2. See Section 16.3.1.2, "IA32_MCG_STATUS MSR."
17BH	379	IA32_MCG_CTL			Machine Check Feature Enable (R/W) See Table 2-2. See Section 16.3.1.3, "IA32_MCG_CTL MSR."
180H	384	MSR_MCG_RAX	0, 1, 2, 3, 4, 6	Unique	Machine Check EAX/RAX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
181H	385	MSR_MCG_RBX	0, 1, 2, 3, 4, 6	Unique	Machine Check EBX/RBX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
182H	386	MSR_MCG_RCX	0, 1, 2, 3, 4, 6	Unique	Machine Check ECX/RCX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
183H	387	MSR_MCG_RDX	0, 1, 2, 3, 4, 6	Unique	Machine Check EDX/RDX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
184H	388	MSR_MCG_RSI	0, 1, 2, 3, 4, 6	Unique	Machine Check ESI/RSI Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
185H	389	MSR_MCG_RDI	0, 1, 2, 3, 4, 6	Unique	Machine Check EDI/RDI Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
186H	390	MSR_MCG_RBP	0, 1, 2, 3, 4, 6	Unique	Machine Check EBP/RBP Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
187H	391	MSR_MCG_RSP	0, 1, 2, 3, 4, 6	Unique	Machine Check ESP/RSP Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
188H	392	MSR_MCG_RFLAGS	0, 1, 2, 3, 4, 6	Unique	Machine Check EFLAGS/RFLAG Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
189H	393	MSR_MCG_RIP	0, 1, 2, 3, 4, 6	Unique	Machine Check EIP/RIP Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
18AH	394	MSR_MCG_MISC	0, 1, 2, 3, 4, 6	Unique	Machine Check Miscellaneous See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		0			DS When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.
		63:1			Reserved
18BH-18FH	395-399	MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
190H	400	MSR_MCG_R8	0, 1, 2, 3, 4, 6	Unique	Machine Check R8 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
191H	401	MSR_MCG_R9	0, 1, 2, 3, 4, 6	Unique	Machine Check R9D/R9 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
192H	402	MSR_MCG_R10	0, 1, 2, 3, 4, 6	Unique	Machine Check R10 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
193H	403	MSR_MCG_R11	0, 1, 2, 3, 4, 6	Unique	Machine Check R11 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
194H	404	MSR_MCG_R12	0, 1, 2, 3, 4, 6	Unique	Machine Check R12 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
195H	405	MSR_MCG_R13	0, 1, 2, 3, 4, 6	Unique	Machine Check R13 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
196H	406	MSR_MCG_R14	0, 1, 2, 3, 4, 6	Unique	Machine Check R14 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
197H	407	MSR_MCG_R15	0, 1, 2, 3, 4, 6	Unique	Machine Check R15 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	See Table 2-2. See Section 15.1, "Enhanced Intel Speedstep® Technology."
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	See Table 2-2. See Section 15.1, "Enhanced Intel Speedstep® Technology."
19AH	410	IA32_CLOCK_MODULATION	0, 1, 2, 3, 4, 6	Unique	Thermal Monitor Control (R/W) See Table 2-2. See Section 15.8.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3, 4, 6	Unique	Thermal Interrupt Control (R/W) See Section 15.8.2, "Thermal Monitor," and Table 2-2.
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	Thermal Monitor Status (R/W) See Section 15.8.2, "Thermal Monitor," and Table 2-2.
19DH	413	MSR_THERM2_CTL			Thermal Monitor 2 Control
			3,	Shared	For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.
			4, 6	Shared	For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3, 4, 6	Shared	Enable Miscellaneous Processor Features (R/W)

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		0			Fast-Strings Enable. See Table 2-2.
		1			Reserved
		2			x87 FPU Fopcode Compatibility Mode Enable
		3			Thermal Monitor 1 Enable See Section 15.8.2, "Thermal Monitor," and Table 2-2.
		4			Split-Lock Disable When set, the bit causes an #AC exception to be issued instead of a split-lock cycle. Operating systems that set this bit must align system structures to avoid split-lock scenarios. When the bit is clear (default), normal split-locks are issued to the bus.
					This debug feature is specific to the Pentium 4 processor.
		5			Reserved
		6			Third-Level Cache Disable (R/W) When set, the third-level cache is disabled; when clear (default) the third-level cache is enabled. This flag is reserved for processors that do not have a third-level cache. Note that the bit controls only the third-level cache; and only if overall caching is enabled through the CD flag of control register CR0, the page-level cache controls, and/or the MTRRs. See Section 12.5.4, "Disabling and Enabling the L3 Cache."
		7			Performance Monitoring Available (R) See Table 2-2.
		8			Suppress Lock Enable When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.
		9			Prefetch Queue Disable When set, disables the prefetch queue. When clear (default), enables the prefetch queue.
		10			FERR# Interrupt Reporting Enable (R/W) When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
					When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt.
					This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted.
		11			Branch Trace Storage Unavailable (BTS_UNAVILABLE) (R) See Table 2-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.
		12			PEBS_UNAVILABLE: Processor Event Based Sampling Unavailable (R) See Table 2-2. When set, the processor does not support processor event-based sampling (PEBS); when clear, PEBS is supported.
		13	3		TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		17:14			Reserved
		18	3, 4, 6		ENABLE MONITOR FSM (R/W) See Table 2-2.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
		19			Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line of the 128-byte sector containing currently required data. When set to 0, the processor fetches both cache lines in the sector.
					Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.
		21:20			Reserved
		22	3, 4, 6		Limit CPUID MAXVAL (R/W) See Table 2-2. Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.
		23		Shared	xTPR Message Disable (R/W) See Table 2-2.
		24			L1 Data Cache Context Mode (R/W) When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 12.5.6, "L1 Data Cache Context Mode." When L1 is running in adaptive mode and CR3s are identical, data in L1 is shared across logical processors. Otherwise, L1 is not shared and cache use is competitive. If the Context ID feature flag (ECX[10]) is set to 0 after executing CPUID with EAX = 1, the ability to switch modes is not supported. BIOS must not alter the contents of IA32_MISC_ENABLE[24].
		33:25			Reserved
		34		Unique	XD Bit Disable (R/W) See Table 2-2.
		63:35			Reserved
		1A1H	417	MSR_PLATFORM_BRV	3, 4, 6
		17:0			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
		18			PLATFORM Requirements When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.
		63:19			Reserved
1D7H	471	MSR_LER_FROM_LIP	0, 1, 2, 3, 4, 6	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.13.3, "Last Exception Records."
		31:0			From Linear IP Linear address of the last branch instruction.
		63:32			Reserved
1D7H	471	63:0		Unique	From Linear IP Linear address of the last branch instruction (If IA-32e mode is active).
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3, 4, 6	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.13.3, "Last Exception Records."
		31:0			From Linear IP Linear address of the target of the last branch instruction.
		63:32			Reserved
1D8H	472	63:0		Unique	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).
1D9H	473	MSR_DEBUGCTLA	0, 1, 2, 3, 4, 6	Unique	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 18.13.1, "MSR_DEBUGCTLA MSR."
1DAH	474	MSR_LASTBRANCH_TOS	0, 1, 2, 3, 4, 6	Unique	Last Branch Record Stack TOS (R/O) Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 18.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture," and addresses 1DBH-1DEH and 680H-68FH.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2	Unique	Last Branch Record 0 (R/O) One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH.
					See Section 18.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
1DCH	477	MSR_LASTBRANCH_1	0, 1, 2	Unique	Last Branch Record 1 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2	Unique	Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2	Unique	Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
200H	512	IA32_MTRR_PHYSBASE0	0, 1, 2, 3, 4, 6	Shared	Variable Range Base MTRR See Section 12.11.2.3, "Variable Range MTRRs."
201H	513	IA32_MTRR_PHYSMASK0	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
202H	514	IA32_MTRR_PHYSBASE1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
203H	515	IA32_MTRR_PHYSMASK1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
204H	516	IA32_MTRR_PHYSBASE2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
205H	517	IA32_MTRR_PHYSMASK2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
206H	518	IA32_MTRR_PHYSBASE3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
207H	519	IA32_MTRR_PHYSMASK3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
208H	520	IA32_MTRR_PHYSBASE4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
209H	521	IA32_MTRR_PHYSMASK4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
20AH	522	IA32_MTRR_PHYSBASE5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
20BH	523	IA32_MTRR_PHYSMASK5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
20CH	524	IA32_MTRR_PHYSBASE6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
20DH	525	IA32_MTRR_PHYSMASK6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
20EH	526	IA32_MTRR_PHYSBASE7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
20FH	527	IA32_MTRR_PHYSMASK7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."
250H	592	IA32_MTRR_FIX64K_00000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
258H	600	IA32_MTRR_FIX16K_80000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
259H	601	IA32_MTRR_FIX16K_A0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
268H	616	IA32_MTRR_FIX4K_C0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
269H	617	IA32_MTRR_FIX4K_C8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
26AH	618	IA32_MTRR_FIX4K_D0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
26BH	619	IA32_MTRR_FIX4K_D8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
26CH	620	IA32_MTRR_FIX4K_E0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
26DH	621	IA32_MTRR_FIX4K_E8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
26EH	622	IA32_MTRR_FIX4K_F0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
26FH	623	IA32_MTRR_FIX4K_F8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."
277H	631	IA32_PAT	0, 1, 2, 3, 4, 6	Unique	Page Attribute Table See Section 12.11.2.2, "Fixed Range MTRRs."
2FFH	767	IA32_MTRR_DEF_TYPE	0, 1, 2, 3, 4, 6	Shared	Default Memory Types (R/W) See Table 2-2. See Section 12.11.2.1, "IA32_MTRR_DEF_TYPE MSR."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
308H	776	MSR_FLAME_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
309H	777	MSR_FLAME_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
30AH	778	MSR_FLAME_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
30BH	779	MSR_FLAME_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.2, "Performance Counters."
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
368H	872	MSR_FLAME_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.3, "CCCR MSRs."
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3AEH	942	MSR_SAA_T_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3AFH	943	MSR_SAA_T_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 20.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	Shared	See Section 20.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3F0H	1008	MSR_TC_PRECISE_EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 20.6.3.1, "ESCR MSRs."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	0, 1, 2, 3, 4, 6	Shared	Processor Event Based Sampling (PEBS) (R/W) Controls the enabling of processor event sampling and replay tagging.
		12:0			See https://perfmon-events.intel.com/ .
		23:13			Reserved
		24			UOP Tag Enables replay tagging when set.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
		25			ENABLE_PEBS_MY_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 20.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is called ENABLE_PEBS in IA-32 processors that do not support Intel Hyper-Threading Technology.
		26			ENABLE_PEBS_OTH_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 20.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Intel Hyper-Threading Technology.
		63:27			Reserved
3F2H	1010	MSR_PEBS_MATRIX_VERT	0, 1, 2, 3, 4, 6	Shared	See https://perfmon-events.intel.com/ .
400H	1024	IA32_MCO_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MCO_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MCO_MISC MSR is either not implemented or does not contain additional information if the MISC_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
406H	1030	IA32_MC1_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	IA32_MC1_MISC		Shared	See Section 16.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISC_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR			See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40BH	1035	IA32_MC2_MISC			See Section 16.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISC_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
40FH	1039	IA32_MC3_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR			See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRIV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC4_MISC			See Section 16.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	3, 4, 6	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	3, 4, 6	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	3, 4, 6	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and see Table 2-2.
483H	1155	IA32_VMX_EXIT_CTL	3, 4, 6	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and Table 2-2.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
484H	1156	IA32_VMX_ENTRY_CTL5	3, 4, 6	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and Table 2-2.
485H	1157	IA32_VMX_MISC	3, 4, 6	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and Table 2-2.
486H	1158	IA32_VMX_CR0_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.
487H	1159	IA32_VMX_CR0_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.
488H	1160	IA32_VMX_CR4_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.
48AH	1162	IA32_VMX_VMCS_ENUM	3, 4, 6	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration," and Table 2-2.
48BH	1163	IA32_VMX_PROCBASED_CTL52	3, 4, 6	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	3, 4, 6	Unique	Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
					The MSRs at 680H-68FH, 6C0H-6CfH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH, which performed the same function for early releases. See Section 18.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
681H	1665	MSR_LASTBRANCH_1_FROM_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	3, 4, 6	Unique	Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 18.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6C0H.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6C0H.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6C0H.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6C0H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6COH.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6COH.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6COH.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6COH.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6COH.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6COH.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6COH.
C000_ 0080H		IA32_EFER	3, 4, 6	Unique	Extended Feature Enables See Table 2-2.
C000_ 0081H		IA32_STAR	3, 4, 6	Unique	System Call Target Address (R/W) See Table 2-2.
C000_ 0082H		IA32_LSTAR	3, 4, 6	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_ 0084H		IA32_FMASK	3, 4, 6	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_ 0100H		IA32_FS_BASE	3, 4, 6	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_ 0101H		IA32_GS_BASE	3, 4, 6	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_ 0102H		IA32_KERNEL_GS_BASE	3, 4, 6	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

NOTES

1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.

2.19.1 MSRs Unique to Intel® Xeon® Processor MP with L3 Cache

The MSRs listed in Table 2-56 apply to Intel® Xeon® Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (see CPUID instruction for more details).

Table 2-56. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache

Register Address	Register Name Fields and Flags	Model Availability	Shared/Unique	Bit Description
107CCH	MSR_IFSB_BUSQ0	3, 4	Shared	IFSB BUSQ Event Control and Counter Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
107CDH	MSR_IFSB_BUSQ1	3, 4	Shared	IFSB BUSQ Event Control and Counter Register (R/W)
107CEH	MSR_IFSB_SNPQ0	3, 4	Shared	IFSB SNPQ Event Control and Counter Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
107CFH	MSR_IFSB_SNPQ1	3, 4	Shared	IFSB SNPQ Event Control and Counter Register (R/W)
107D0H	MSR_EFSB_DRDY0	3, 4	Shared	EFSB DRDY Event Control and Counter Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
107D1H	MSR_EFSB_DRDY1	3, 4	Shared	EFSB DRDY Event Control and Counter Register (R/W)
107D2H	MSR_IFSB_CTL6	3, 4	Shared	IFSB Latency Event Control Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
107D3H	MSR_IFSB_CNTR7	3, 4	Shared	IFSB Latency Event Counter Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."

The MSRs listed in Table 2-57 apply to Intel® Xeon® Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table 2-57 are shared between logical processors in the same core, but are replicated for each core.

Table 2-57. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address	Register Name Fields and Flags	Model Availability	Shared/Unique	Bit Description
107CCH	MSR_EMON_L3_CTR_CTL0	6	Shared	GBUSQ Event Control and Counter Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."

Table 2-57. MSRs Unique to Intel® Xeon® Processor 7100 Series (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique	Bit Description
107CDH		MSR_EMON_L3_CTR_CTL1	6	Shared	GBUSQ Event Control and Counter Register (R/W)
107CEH		MSR_EMON_L3_CTR_CTL2	6	Shared	GSNPQ Event Control and Counter Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
107CFH		MSR_EMON_L3_CTR_CTL3	6	Shared	GSNPQ Event Control and Counter Register (R/W)
107D0H		MSR_EMON_L3_CTR_CTL4	6	Shared	FSB Event Control and Counter Register (R/W) See Section 20.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
107D1H		MSR_EMON_L3_CTR_CTL5	6	Shared	FSB Event Control and Counter Register (R/W)
107D2H		MSR_EMON_L3_CTR_CTL6	6	Shared	FSB Event Control and Counter Register (R/W)
107D3H		MSR_EMON_L3_CTR_CTL7	6	Shared	FSB Event Control and Counter Register (R/W)

2.20 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table 2-58. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	P5_MC_ADDR	Unique	See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.
1H	1	P5_MC_TYPE	Unique	See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 18.17, "Time-Stamp Counter," and Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	Unique	See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		6: 5		Reserved
		7		BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled
		9		Execute BIST (R/O) 1 = Enabled; 0 = Disabled
		10		MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled
		11		Reserved
		12		BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled
		13		Reserved
		14		1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		APIC Cluster ID (R/O)
		18		System Bus Frequency (R/O) 0 = 100 MHz 1 = Reserved
		19		Reserved
		21: 20		Symmetric Arbitration ID (R/O)
26:22	Clock Frequency Ratio (R/O)			

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in IA-32 Processor (R/W) See Table 2-2.
40H	64	MSR_LASTBRANCH_0	Unique	Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_LASTBRANCH_1	Unique	Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	Unique	Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	Unique	Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	Unique	Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	Unique	Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	Unique	Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	Unique	Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Unique	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (R/O) This field indicates the scalable bus clock speed:
		2:0		<ul style="list-style-type: none"> ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) <p>133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.</p>
		63:3		Reserved

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved
		8		L2 Enabled (R/W) 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved
		23		L2 Not Present (R/O) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1		EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2		MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2. See Section 15.8.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2. See Section 15.8.2, "Thermal Monitor".
19DH	413	MSR_THERM2_CTL	Unique	Thermal Monitor 2 Control
		15:0		Reserved
		16		TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		2:0		Reserved
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		9:8		Reserved
		10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12		Reserved
		13	Shared	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19		Reserved
		22	Shared	Limit CPUID Maxval (R/W) See Table 2-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 2.
		33:23		Reserved
		34	Shared	XD Bit Disable (R/W) See Table 2-2.
		63:35		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	MTRRphysBase0	Unique	Memory Type Range Registers
201H	513	MTRRphysMask0	Unique	Memory Type Range Registers
202H	514	MTRRphysBase1	Unique	Memory Type Range Registers
203H	515	MTRRphysMask1	Unique	Memory Type Range Registers
204H	516	MTRRphysBase2	Unique	Memory Type Range Registers
205H	517	MTRRphysMask2	Unique	Memory Type Range Registers
206H	518	MTRRphysBase3	Unique	Memory Type Range Registers
207H	519	MTRRphysMask3	Unique	Memory Type Range Registers
208H	520	MTRRphysBase4	Unique	Memory Type Range Registers
209H	521	MTRRphysMask4	Unique	Memory Type Range Registers
20AH	522	MTRRphysBase5	Unique	Memory Type Range Registers
20BH	523	MTRRphysMask5	Unique	Memory Type Range Registers
20CH	524	MTRRphysBase6	Unique	Memory Type Range Registers
20DH	525	MTRRphysMask6	Unique	Memory Type Range Registers
20EH	526	MTRRphysBase7	Unique	Memory Type Range Registers
20FH	527	MTRRphysMask7	Unique	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Unique	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Unique	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Unique	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Unique	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Unique	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Unique	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Unique	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Unique	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Unique	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Unique	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Unique	Memory Type Range Registers

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	Default Memory Types (R/W) See Table 2-2. See Section 12.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MCO_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	Unique	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	Unique	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC4_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC3_CTL		See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC3_ADDR	Unique	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
413H	1043	MSR_MC3_MISC	Unique	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISCV flag in the IA32_MCI_STATUS register is set.
414H	1044	MSR_MC5_CTL	Unique	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
415H	1045	MSR_MC5_STATUS	Unique	Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
416H	1046	MSR_MC5_ADDR	Unique	Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDRV flag in the IA32_MCI_STATUS register is set.
417H	1047	MSR_MC5_MISC	Unique	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISCV flag in the IA32_MCI_STATUS register is set.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information." (If CPUID.01H:ECX.[bit 5])
481H	1153	IA32_VMX_PINBASED_CTL	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX.[bit 5])
482H	1154	IA32_VMX_PROCBASED_CTL	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX.[bit 5])
483H	1155	IA32_VMX_EXIT_CTL	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls." (If CPUID.01H:ECX.[bit 5])
484H	1156	IA32_VMX_ENTRY_CTL	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls." (If CPUID.01H:ECX.[bit 5])
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data." (If CPUID.01H:ECX.[bit 5])
486H	1158	IA32_VMX_CRO_FIXED0	Unique	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CRO." (If CPUID.01H:ECX.[bit 5])

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_FIXED1	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX.[bit 5])
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX.[bit 5])
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX.[bit 5])
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration." (If CPUID.01H:ECX.[bit 5])
48BH	1163	IA32_VMX_PROCBASED_CTL2	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX.[bit 5] and IA32_VMX_PROCBASED_CTL2[bit 63])
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."
		31:0		DS Buffer Management Area Linear address of the first byte of the DS buffer management area.
		63:32		Reserved
C000_0080H		IA32_EFER	Unique	See Table 2-2.
		10:0		Reserved
		11		Execute Disable Bit Enable
		63:12		Reserved

2.21 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 2.22 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table 2-59. MSRs in Pentium M Processors

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.23, "MSRs in Pentium Processors."

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
1H	1	P5_MC_TYPE	See Section 2.23, "MSRs in Pentium Processors."
10H	16	IA32_TIME_STAMP_COUNTER	See Section 18.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
2AH	42	MSR_EBL_CR_POWERON	Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.
		0	Reserved
		1	Data Error Checking Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		2	Response Error Checking Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		3	MCERR# Drive Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		4	Address Parity Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		6:5	Reserved
		7	BINIT# Driver Enable (R) 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled
		9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled
		10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		11	Reserved
		12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
13	Reserved		

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes Always 0 on the Pentium M processor.
		15	Reserved
		17:16	APIC Cluster ID (R/O) Always 00B on the Pentium M processor.
		18	System Bus Frequency (R/O) 0 = 100 MHz 1 = Reserved Always 0 on the Pentium M processor.
		19	Reserved
		21:20	Symmetric Arbitration ID (R/O) Always 00B on the Pentium M processor.
		26:22	Clock Frequency Ratio (R/O)
40H	64	MSR_LASTBRANCH_0	Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_LASTBRANCH_1	Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.
119H	281	MSR_BBL_CR_CTL	Control Register Used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.
		63:0	Reserved
11EH	281	MSR_BBL_CR_CTL3	Control register 3 Used to configure the L2 Cache.

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
		4:1	Reserved
		5	ECC Check Enable (R/O) This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default) 1 = Enabled For the Pentium M processor, ECC checking on the cache data bus is always enabled.
		7:6	Reserved
		8	L2 Enabled (R/W) 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9	Reserved
		23	L2 Not Present (R/O) 0 = L2 Present 1 = L2 Not Present
		63:24	Reserved
179H	377	IA32_MCG_CAP	Read-only register that provides information about the machine-check architecture of the processor.
		7:0	Count (R/O) Indicates the number of hardware unit error reporting banks available in the processor.
		8	IA32_MCG_CTL Present (R/O) 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
		63:9	Reserved
17AH	378	IA32_MCG_STATUS	Global Machine Check Status
		0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3	Reserved
198H	408	IA32_PERF_STATUS	See Table 2-2.
199H	409	IA32_PERF_CTL	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation (R/W). See Table 2-2. See Section 15.8.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) See Table 2-2. See Section 15.8.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	Thermal Monitor Status (R/W) See Table 2-2. See Section 15.8.2, "Thermal Monitor."
19DH	413	MSR_THERM2_CTL	Thermal Monitor 2 Control
		15:0	Reserved
		16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16	Reserved
1A0H	416	IA32_MISC_ENABLE	Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		2:0	Reserved

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
		6:4	Reserved
		7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.
		9:8	Reserved
		10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
			Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
		12	Processor Event Based Sampling Unavailable (R/O) 1 = Processor does not support processor event based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
		15:13	Reserved
		16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
		22:17	Reserved
		23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
		63:24	Reserved

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
1C9H	457	MSR_LASTBRANCH_TOS	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: <ul style="list-style-type: none"> MSR_LASTBRANCH_0_FROM_IP (at 40H). Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
1D9H	473	MSR_DEBUGCTLB	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
1DDH	477	MSR_LER_TO_LIP	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 18.16.2, "Last Branch and Last Exception MSRs."
1DEH	478	MSR_LER_FROM_LIP	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 18.16.2, "Last Branch and Last Exception MSRs."
2FFH	767	IA32_MTRR_DEF_TYPE	Default Memory Types (R/W) Sets the memory type for the regions of physical memory that are not mapped by the MTRRs. See Section 12.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MCO_CTL	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs". The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs". The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	See Chapter 16.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
40AH	1034	IA32_MC2_ADDR	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	MSR_MC4_ADDR	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_STATUS	See Section 16.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	MSR_MC3_ADDR	See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
600H	1536	IA32_DS_AREA	DS Save Area (R/W) See Table 2-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 20.6.3.4, "Debug Store (DS) Mechanism."
		31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.
		63:32	Reserved

2.22 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table 2-2 for a list of the architectural MSRs.

Table 2-60. MSRs in the P6 Family Processors

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.23, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Section 2.23, "MSRs in Pentium Processors."
10H	16	TSC	See Section 18.17, "Time-Stamp Counter."

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
17H	23	IA32_PLATFORM_ID	Platform ID (R) The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.
		49:0	Reserved
		52:50	Platform Id (R) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
		56:53	L2 Cache Latency Read.
		59:57	Reserved
		60	Clock Frequency Ratio Read.
		63:61	Reserved
		1BH	27
7:0	Reserved		
8	Boot Strap Processor Indicator Bit 1 = BSP		
10:9	Reserved		
11	APIC Global Enable Bit - Permanent till reset 1 = Enabled 0 = Disabled		
31:12	APIC Base Address.		
63:32	Reserved		
2AH	42	EBL_CR_POWERON	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0	Reserved ¹
		1	Data Error Checking Enable (R/W) 1 = Enabled 0 = Disabled
		2	Response Error Checking Enable FRCERR Observation Enable (R/W) 1 = Enabled 0 = Disabled
		3	AERR# Drive Enable (R/W) 1 = Enabled 0 = Disabled

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		4	BERR# Enable for Initiator Bus Requests (R/W) 1 = Enabled 0 = Disabled
		5	Reserved
		6	BERR# Driver Enable for Initiator Internal Errors (R/W) 1 = Enabled 0 = Disabled
		7	BINIT# Driver Enable (R/W) 1 = Enabled 0 = Disabled
		8	Output Tri-state Enabled (R) 1 = Enabled 0 = Disabled
		9	Execute BIST (R) 1 = Enabled 0 = Disabled
		10	AERR# Observation Enabled (R) 1 = Enabled 0 = Disabled
		11	Reserved
		12	BINIT# Observation Enabled (R) 1 = Enabled 0 = Disabled
		13	In Order Queue Depth (R) 1 = 1 0 = 8
		14	1-MByte Power on Reset Vector (R) 1 = 1MByte 0 = 4GBytes
		15	FRC Mode Enable (R) 1 = Enabled 0 = Disabled
		17:16	APIC Cluster ID (R)
		19:18	System Bus Frequency (R) 00 = 66MHz 10 = 100MHz 01 = 133MHz 11 = Reserved
		21:20	Symmetric Arbitration ID (R)
		25:22	Clock Frequency Ratio (R)
		26	Low Power Mode Enable (R/W)

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		27	Clock Frequency Ratio
		63:28	Reserved ¹
33H	51	MSR_TEST_CTRL	Test Control Register
		29:0	Reserved
		30	Streaming Buffer Disable
		31	Disable LOCK# Assertion for split locked access.
79H	121	BIOS_UPDT_TRIG	BIOS Update Trigger Register.
88H	136	BBL_CR_D0[63:0]	Chunk 0 data register D[63:0]: used to write to and read from the L2
89H	137	BBL_CR_D1[63:0]	Chunk 1 data register D[63:0]: used to write to and read from the L2
8AH	138	BBL_CR_D2[63:0]	Chunk 2 data register D[63:0]: used to write to and read from the L2
8BH	139	BIOS_SIGN/BBL_CR_D3[63:0]	BIOS Update Signature Register or Chunk 3 data register D[63:0] Used to write to and read from the L2 depending on the usage model.
C1H	193	PerfCtr0 (PERFCTR0)	Performance Counter Register See Table 2-2.
C2H	194	PerfCtr1 (PERFCTR1)	Performance Counter Register See Table 2-2.
FEH	254	MTRRcap	Memory Type Range Registers
116H	278	BBL_CR_ADDR [63:0]	Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.
		BBL_CR_ADDR [63:32]	Reserved,
		BBL_CR_ADDR [31:3]	Address bits [35:3]
		BBL_CR_ADDR [2:0]	Reserved Set to 0.
118H	280	BBL_CR_DECC[63:0]	Data ECC register D[7:0]: used to write ECC and read ECC to/from L2
119H	281	BBL_CR_CTL	Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response
		BL_CR_CTL[63:22]	Reserved
		BBL_CR_CTL[21]	Processor number ² Disable = 1 Enable = 0 Reserved

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		BBL_CR_CTL[20:19] BBL_CR_CTL[18] BBL_CR_CTL[17] BBL_CR_CTL[16] BBL_CR_CTL[15:14] BBL_CR_CTL[13:12] BBL_CR_CTL[11:10] BBL_CR_CTL[9:8] BBL_CR_CTL[7] BBL_CR_CTL[6:5]	User supplied ECC Reserved L2 Hit Reserved State from L2 Modified - 11, Exclusive - 10, Shared - 01, Invalid - 00 Way from L2 Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11 Way to L2 Reserved State to L2
		BBL_CR_CTL[4:0] 01100 01110 01111 00010 00011 010 + MESI encode 111 + MESI encode 100 + MESI encode	L2 Command Data Read w/ LRU update (RLU) Tag Read w/ Data Read (TRR) Tag Inquire (TI) L2 Control Register Read (CR) L2 Control Register Write (CW) Tag Write w/ Data Read (TWR) Tag Write w/ Data Write (TWW) Tag Write (TW)
11AH	282	BBL_CR_TRIG	Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.
11BH	283	BBL_CR_BUSY	Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY
11EH	286	BBL_CR_CTL3 BBL_CR_CTL3[63:26] BBL_CR_CTL3[25] BBL_CR_CTL3[24] BBL_CR_CTL3[23] BBL_CR_CTL3[22:20] 111 110 101 100 011 010 001 000 BBL_CR_CTL3[19] BBL_CR_CTL3[18]	Control register 3: used to configure the L2 Cache Reserved Cache bus fraction (read only) Reserved L2 Hardware Disable (read only) L2 Physical Address Range support 64GBytes 32GBytes 16GBytes 8GBytes 4GBytes 2GBytes 1GBytes 512MBytes Reserved Cache State error checking enable (read/write)

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		BBL_CR_CTL3[17:13] 00001 00010 00100 01000 10000 BBL_CR_CTL3[12:11] BBL_CR_CTL3[10:9] 00 01 10 11 BBL_CR_CTL3[8] BBL_CR_CTL3[7] BBL_CR_CTL3[6] BBL_CR_CTL3[5] BBL_CR_CTL3[4:1] BBL_CR_CTL3[0]	Cache size per bank (read/write) 256KBytes 512KBytes 1MByte 2MByte 4MBytes Number of L2 banks (read only) L2 Associativity (read only) Direct Mapped 2 Way 4 Way Reserved L2 Enabled (read/write) CRTN Parity Check Enable (read/write) Address Parity Check Enable (read/write) ECC Check Enable (read/write) L2 Cache Latency (read/write) L2 Configured (read/write)
174H	372	SYSENTER_CS_MSR	CS register target for CPL 0 code
175H	373	SYSENTER_ESP_MSR	Stack pointer for CPL 0 stack
176H	374	SYSENTER_EIP_MSR	CPL 0 code entry point
179H	377	MCG_CAP	Machine Check Global Control Register
17AH	378	MCG_STATUS	Machine Check Error Reporting Register - contains information related to machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
17BH	379	MCG_CTL	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
186H	390	PerfEvtSel0 (EVNTSEL0)	Performance Event Select Register 0 (R/W)
		7:0	Event Select Refer to Performance Counter section for a list of event encodings.
		15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
		16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	OS Controls the counting of events at Privilege level of 0.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		18	E Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	PC Enabled the signaling of performance counter overflow via BPO pin
		20	INT Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		22	ENABLE Enables the counting of performance events in both counters 1 = Enable 0 = Disable
		23	INV Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	CMASK (Counter Mask)
187H	391	PerfEvtSel1 (EVNTSEL1)	Performance Event Select for Counter 1 (R/W)
		7:0	Event Select Refer to Performance Counter section for a list of event encodings.
		15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
		16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	OS Controls the counting of events at Privilege level of 0.
		18	E Occurrence/Duration Mode Select. 1 = Occurrence 0 = Duration
		19	PC Enabled the signaling of performance counter overflow via BPO pin.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		20	INT Enables the signaling of counter overflow via input to APIC. 1 = Enable 0 = Disable
		23	INV Inverts the result of the CMASK condition. 1 = Inverted 0 = Non-Inverted
		31:24	CMASK (Counter Mask)
1D9H	473	DEBUGCTLMR	Enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.
		0	Enable/Disable Last Branch Records
		1	Branch Trap Flag
		2	Performance Monitoring/Break Point Pins
		3	Performance Monitoring/Break Point Pins
		4	Performance Monitoring/Break Point Pins
		5	Performance Monitoring/Break Point Pins
		6	Enable/Disable Execution Trace Messages
31:7	Reserved		
1DBH	475	LASTBRANCHFROMIP	32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.
1DCH	476	LASTBRANCHTOIP	32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.
1DDH	477	LASTINTFROMIP	Last INT from IP
1DEH	478	LASTINTTOIP	Last INT to IP
200H	512	MTRRphysBase0	Memory Type Range Registers
201H	513	MTRRphysMask0	Memory Type Range Registers
202H	514	MTRRphysBase1	Memory Type Range Registers
203H	515	MTRRphysMask1	Memory Type Range Registers
204H	516	MTRRphysBase2	Memory Type Range Registers
205H	517	MTRRphysMask2	Memory Type Range Registers
206H	518	MTRRphysBase3	Memory Type Range Registers
207H	519	MTRRphysMask3	Memory Type Range Registers
208H	520	MTRRphysBase4	Memory Type Range Registers
209H	521	MTRRphysMask4	Memory Type Range Registers
20AH	522	MTRRphysBase5	Memory Type Range Registers

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
20BH	523	MTRRphysMask5	Memory Type Range Registers
20CH	524	MTRRphysBase6	Memory Type Range Registers
20DH	525	MTRRphysMask6	Memory Type Range Registers
20EH	526	MTRRphysBase7	Memory Type Range Registers
20FH	527	MTRRphysMask7	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Memory Type Range Registers
2FFH	767	MTRRdefType	Memory Type Range Registers
		2:0	Default memory type
		10	Fixed MTRR enable
		11	MTRR Enable
400H	1024	MCO_CTL	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
401H	1025	MCO_STATUS	Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
		15:0	MC_STATUS_MCACOD
		31:16	MC_STATUS_MSCOD
		57	MC_STATUS_DAM
		58	MC_STATUS_ADDRV
		59	MC_STATUS_MISCV
		60	MC_STATUS_EN. (Note: For MCO_STATUS only, this bit is hardcoded to 1.)
		61	MC_STATUS_UC
		62	MC_STATUS_O
63	MC_STATUS_V		
402H	1026	MCO_ADDR	
403H	1027	MCO_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
404H	1028	MC1_CTL	
405H	1029	MC1_STATUS	Bit definitions same as MCO_STATUS.
406H	1030	MC1_ADDR	
407H	1031	MC1_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
408H	1032	MC2_CTL	
409H	1033	MC2_STATUS	Bit definitions same as MCO_STATUS.
40AH	1034	MC2_ADDR	
40BH	1035	MC2_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
40CH	1036	MC4_CTL	
40DH	1037	MC4_STATUS	Bit definitions same as MCO_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.
40EH	1038	MC4_ADDR	Defined in MCA architecture but not implemented in P6 Family processors.
40FH	1039	MC4_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
410H	1040	MC3_CTL	
411H	1041	MC3_STATUS	Bit definitions same as MCO_STATUS.
412H	1042	MC3_ADDR	
413H	1043	MC3_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

NOTES

- 1.Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors.
- 2.The processor number feature may be disabled by setting bit 21 of the BBL_CR_CTL MSR (model-specific register address 119h) to "1". Once set, bit 21 of the BBL_CR_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset.
- 3.The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.

2.23 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5_MC_ADDR, P5_MC_TYPE, and TSC MSRs (named IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, and IA32_TIME_STAMP_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section 2.1, "Architectural MSRs"). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table 2-61. MSRs in the Pentium Processor

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 16.10.2, "Pentium Processor Machine-Check Exception Handling."
1H	1	P5_MC_TYPE	See Section 16.10.2, "Pentium Processor Machine-Check Exception Handling."
10H	16	TSC	See Section 18.17, "Time-Stamp Counter."
11H	17	CESR	See Section 20.6.9.1, "Control and Event Select Register (CESR)."
12H	18	CTRO	Section 20.6.9.3, "Events Counted."
13H	19	CTR1	Section 20.6.9.3, "Events Counted."

2.24 MSR INDEX

MSRs of recent processors are indexed here for convenience. IA32 MSRs are excluded from this index.

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_ALF_ESCR0	
0FH	See Table 2-55
MSR_ALF_ESCR1	
0FH	See Table 2-55
MSR_ANY_CORE_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_ANY_GFXE_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_BO_PMON_BOX_CTRL	
06_2EH	See Table 2-17
MSR_BO_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_BO_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_BO_PMON_CTRO	
06_2EH	See Table 2-17
MSR_BO_PMON_CTR1	
06_2EH	See Table 2-17
MSR_BO_PMON_CTR2	
06_2EH	See Table 2-17
MSR_BO_PMON_CTR3	
06_2EH	See Table 2-17
MSR_BO_PMON_EVTN_SELO	
06_2EH	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_BO_PMON_EVNT_SEL1 06_2EH	See Table 2-17
MSR_BO_PMON_EVNT_SEL2 06_2EH	See Table 2-17
MSR_BO_PMON_EVNT_SEL3 06_2EH	See Table 2-17
MSR_BO_PMON_MASK 06_2EH	See Table 2-17
MSR_BO_PMON_MATCH 06_2EH	See Table 2-17
MSR_B1_PMON_BOX_CTRL 06_2EH	See Table 2-17
MSR_B1_PMON_BOX_OVF_CTRL 06_2EH	See Table 2-17
MSR_B1_PMON_BOX_STATUS 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL0 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL1 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL2 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL3 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SELO 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL1 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL2 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL3 06_2EH	See Table 2-17
MSR_B1_PMON_MASK 06_2EH	See Table 2-17
MSR_B1_PMON_MATCH 06_2EH	See Table 2-17
MSR_BBL_CR_CTL 06_09H	See Table 2-59
MSR_BBL_CR_CTL3 06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_BIOS_DEBUG	
06_8CH, 06_8DH	See Table 2-45
MSR_BIOS_DONE	
06_7DH, 06_7EH	See Table 2-44
MSR_BIOS_MCU_ERRORCODE	
06_7DH, 06_7EH	See Table 2-44
06_8CH, 06_8DH	See Table 2-45
MSR_BPU_CCCR0	
0FH	See Table 2-55
MSR_BPU_CCCR1	
0FH	See Table 2-55
MSR_BPU_CCCR2	
0FH	See Table 2-55
MSR_BPU_CCCR3	
0FH	See Table 2-55
MSR_BPU_COUNTER0	
0FH	See Table 2-55
MSR_BPU_COUNTER1	
0FH	See Table 2-55
MSR_BPU_COUNTER2	
0FH	See Table 2-55
MSR_BPU_COUNTER3	
0FH	See Table 2-55
MSR_BPU_ESCR0	
0FH	See Table 2-55
MSR_BPU_ESCR1	
0FH	See Table 2-55
MSR_BR_DETECT_COUNTER_CONFIG_i	
06_66H	See Table 2-42
MSR_BR_DETECT_CTRL	
06_66H	See Table 2-42
MSR_BR_DETECT_STATUS	
06_66H	See Table 2-42
MSR_BSU_ESCR0	
0FH	See Table 2-55
MSR_BSU_ESCR1	
0FH	See Table 2-55
MSR_CO_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH	See Table 2-33
MSR_CO_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_CO_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_CO_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_CO_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_CO_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_CO_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_CO_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL1	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_CO_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C1_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C1_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C1_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C1_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH	See Table 2-33
MSR_C1_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C1_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C1_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C1_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C10_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C10_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C10_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C11_PMON_BOX_FILTER	
06_3EH	See Table 2-28

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C11_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C11_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C12_PMON_BOX_FILTER 06_3EH	See Table 2-28
MSR_C12_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C12_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C13_PMON_BOX_FILTER 06_3EH	See Table 2-28
MSR_C13_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C13_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C14_PMON_BOX_FILTER 06_3EH	See Table 2-28
MSR_C14_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C14_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C15_PMON_BOX_CTL 06_3FH	See Table 2-33
MSR_C15_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C15_PMON_BOX_FILTER1 06_3FH	See Table 2-33
MSR_C15_PMON_BOX_STATUS 06_3FH	See Table 2-33
MSR_C15_PMON_CTR0 06_3FH	See Table 2-33
MSR_C15_PMON_CTR1 06_3FH	See Table 2-33
MSR_C15_PMON_CTR2 06_3FH	See Table 2-33
MSR_C15_PMON_CTR3 06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C15_PMON_EVNTSELO 06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL1 06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL2 06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL3 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_CTL 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_FILTER1 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_STATUS 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL0 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL3 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL2 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL3 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSELO 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL1 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL2 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL3 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_CTL 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_FILTER1 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_STATUS 06_3FH	See Table 2-33
MSR_C17_PMON_CTRL0 06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C17_PMON_CTR1 06_3FH	See Table 2-33
MSR_C17_PMON_CTR2 06_3FH	See Table 2-33
MSR_C17_PMON_CTR3 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSELO 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL1 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL2 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL3 06_3FH	See Table 2-33
MSR_C2_PMON_BOX_CTRL 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_FILTER 06_2DH	See Table 2-24
MSR_C2_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C2_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_OVF_CTRL 06_2EH	See Table 2-17
MSR_C2_PMON_BOX_STATUS 06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C2_PMON_CTR0 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR1 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR2 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C2_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C2_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C2_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C2_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C3_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C3_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C3_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C3_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C3_PMON_EVTN_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL4	
06_2EH	See Table 2-17
MSR_C3_PMON_EVTN_SEL5	
06_2EH	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C4_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C4_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C4_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C4_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C4_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL1	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C4_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C5_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C5_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C5_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C5_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C5_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C5_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C5_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C5_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C6_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C6_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C6_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C6_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C6_PMON_EVTN_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL4	
06_2EH	See Table 2-17
MSR_C6_PMON_EVTN_SEL5	
06_2EH	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C7_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C7_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C7_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C7_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C7_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL1	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C7_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C8_PMON_BOX_CTRL	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C8_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_OVF_CTRL	
06_2FH	See Table 2-19
MSR_C8_PMON_BOX_STATUS	
06_2FH	See Table 2-19
06_3FH	See Table 2-33
MSR_C8_PMON_CTR0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTR1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTR2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C8_PMON_CTR3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTR4	
06_2FH	See Table 2-19
MSR_C8_PMON_CTR5	
06_2FH	See Table 2-19
MSR_C8_PMON_EVNT_SELO	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL4	
06_2FH	See Table 2-19
MSR_C8_PMON_EVNT_SEL5	
06_2FH	See Table 2-19
MSR_C9_PMON_BOX_CTRL	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C9_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_OVF_CTRL	
06_2FH	See Table 2-19
MSR_C9_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2FH	See Table 2-19
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL4	
06_2FH	See Table 2-19
MSR_C9_PMON_CTRL5	
06_2FH	See Table 2-19
MSR_C9_PMON_EVTN_SEL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL4	
06_2FH	See Table 2-19
MSR_C9_PMON_EVTN_SEL5	
06_2FH	See Table 2-19

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_CC6_DEMOTION_POLICY_CONFIG	
06_37H	See Table 2-9
MSR_CONFIG_TDP_CONTROL	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_LEVEL1	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_LEVEL2	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_NOMINAL	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CORE_C1_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_66H	See Table 2-42
MSR_CORE_C3_RESIDENCY	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
MSR_CORE_C6_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_CORE_C7_RESIDENCY	
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
MSR_CORE_GFXE_OVERLAP_CO	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_CORE_HDC_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_CORE_PERF_LIMIT_REASONS	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H	See Table 2-30
06_3F	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_56H, 06_4FH	See Table 2-36
06_57H, 06_85H.....	See Table 2-53
MSR_CORE_THREAD_COUNT	
06_3FH.....	See Table 2-32
MSR_CRASHLOG_CONTROL	
06_7DH, 06_7EH	See Table 2-44
MSR_CRU_ESCR0	
0FH	See Table 2-55
MSR_CRU_ESCR1	
0FH	See Table 2-55
MSR_CRU_ESCR2	
0FH	See Table 2-55
MSR_CRU_ESCR3	
0FH	See Table 2-55
MSR_CRU_ESCR4	
0FH	See Table 2-55
MSR_CRU_ESCR5	
0FH	See Table 2-55
MSR_DAC_ESCR0	
0FH	See Table 2-55
MSR_DAC_ESCR1	
0FH	See Table 2-55
MSR_DRAM_ENERGY_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H.....	See Table 2-53
MSR_DRAM_PERF_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H.....	See Table 2-53
MSR_DRAM_POWER_INFO	
06_5CH, 06_7AH	See Table 2-12

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H	See Table 2-53
MSR_DRAM_POWER_LIMIT	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H	See Table 2-53
MSR_EBC_FREQUENCY_ID	
0FH	See Table 2-55
MSR_EBC_HARD_POWERON	
0FH	See Table 2-55
MSR_EBC_SOFT_POWERON	
0FH	See Table 2-55
MSR_EBL_CR_POWERON	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_EFSB_DRDY0	
0F_03H, 0F_04H	See Table 2-56
MSR_EFSB_DRDY1	
0F_03H, 0F_04H	See Table 2-56
MSR_EMON_L3_CTR_CTL0	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL1	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL2	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL3	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_EMON_L3_CTR_CTL4	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL5	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL6	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL7	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_GL_CTL	
06_0FH, 06_17H	See Table 2-3
MSR_ERROR_CONTROL	
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
MSR_FAST_UNCORE_MSRS_CAPABILITY	
06_7DH, 06_7EH	See Table 2-44
MSR_FAST_UNCORE_MSRS_CTL	
06_7DH, 06_7EH	See Table 2-44
MSR_FAST_UNCORE_MSRS_STATUS	
06_7DH, 06_7EH	See Table 2-44
MSR_FEATURE_CONFIG	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_25H, 06_2CH	See Table 2-18
06_2FH	See Table 2-19
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_FIRM_ESCR0	
0FH	See Table 2-55
MSR_FIRM_ESCR1	
0FH	See Table 2-55
MSR_FLAME_CCCR0	
0FH	See Table 2-55
MSR_FLAME_CCCR1	
0FH	See Table 2-55
MSR_FLAME_CCCR2	
0FH	See Table 2-55
MSR_FLAME_CCCR3	
0FH	See Table 2-55

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_FLAME_COUNTER0 OFH	See Table 2-55
MSR_FLAME_COUNTER1 OFH	See Table 2-55
MSR_FLAME_COUNTER2 OFH	See Table 2-55
MSR_FLAME_COUNTER3 OFH	See Table 2-55
MSR_FLAME_ESCRO OFH	See Table 2-55
MSR_FLAME_ESCR1 OFH	See Table 2-55
MSR_FSB_ESCRO OFH	See Table 2-55
MSR_FSB_ESCR1 OFH	See Table 2-55
MSR_FSB_FREQ 06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_4CH	See Table 2-11
06_0EH	See Table 2-58
MSR_GQ_SNOOP_MESF 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_GRAPHICS_PERF_LIMIT_REASONS 06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_IFSB_BUSQ0 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_BUSQ1 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_CNTR7 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_CTL6 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_SNPQ0 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_SNPQ1 OF_03H, OF_04H	See Table 2-56
MSR_IQ_CCCR0 OFH	See Table 2-55
MSR_IQ_CCCR1 OFH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_IQ_CCCR2	
OFH	See Table 2-55
MSR_IQ_CCCR3	
OFH	See Table 2-55
MSR_IQ_CCCR4	
OFH	See Table 2-55
MSR_IQ_CCCR5	
OFH	See Table 2-55
MSR_IQ_COUNTER0	
OFH	See Table 2-55
MSR_IQ_COUNTER1	
OFH	See Table 2-55
MSR_IQ_COUNTER2	
OFH	See Table 2-55
MSR_IQ_COUNTER3	
OFH	See Table 2-55
MSR_IQ_COUNTER4	
OFH	See Table 2-55
MSR_IQ_COUNTER5	
OFH	See Table 2-55
MSR_IQ_ESCR0	
OFH	See Table 2-55
MSR_IQ_ESCR1	
OFH	See Table 2-55
MSR_IS_ESCR0	
OFH	See Table 2-55
MSR_IS_ESCR1	
OFH	See Table 2-55
MSR_ITLB_ESCR0	
OFH	See Table 2-55
MSR_ITLB_ESCR1	
OFH	See Table 2-55
MSR_IX_ESCR0	
OFH	See Table 2-55
MSR_IX_ESCR1	
OFH	See Table 2-55
MSR_LASTBRANCH_0	
OFH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_0_FROM_IP	
06_OFH, 06_17H	See Table 2-3

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH.....	See Table 2-12
06_7AH.....	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_0_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH.....	See Table 2-12
06_7AH.....	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_1_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_1_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_10_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_10_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_11_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_11_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_12_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_12_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_13_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_13_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_14_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_14_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_15_FROM_IP	
06_5CH, 06_7AH	See Table 2-12

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_15_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_16_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_16_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_17_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_17_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_18_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_18_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_19_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_19_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_2	
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_2_FROM_IP	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_2_TO_IP	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_20_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_20_TO_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_21_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_21_TO_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_22_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_22_TO_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_23_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_23_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_24_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_24_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_25_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_25_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_26_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_26_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_27_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_27_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_28_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_28_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_29_FROM_IP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_29_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_3	
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_3_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_3_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_30_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_30_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_31_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_31_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_4	
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LASTBRANCH_4_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_4_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_5	
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LASTBRANCH_5_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_5_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_6	
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LASTBRANCH_6_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_6_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_7	
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_7_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_7_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_8_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_8_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_9_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_9_TO_IP	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_TOS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_INFO_0	
06_7AH	See Table 2-13
MSR_LBR_INFO_1	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_10	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_11	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_12	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_13	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_14	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_15	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_16	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_17	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_18	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_19	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_2	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_20	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_21	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_22	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_23	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_24	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_25	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_26	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_27	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_28	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_29	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_3	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_30	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_31	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_4	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_5	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_6	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_7	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_8	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_9	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_SELECT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H.....	See Table 2-53
MSR_LER_FROM_LIP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
0FH	See Table 2-55
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LER_TO_LIP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
0FH	See Table 2-55
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_MO_PMON_ADDR_MASK	
06_2EH.....	See Table 2-17
MSR_MO_PMON_ADDR_MATCH	
06_2EH.....	See Table 2-17
MSR_MO_PMON_BOX_CTRL	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_MO_PMON_BOX_OVF_CTRL 06_2EH.....	See Table 2-17
MSR_MO_PMON_BOX_STATUS 06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL0 06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL1 06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL2 06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL3 06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL4 06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL5 06_2EH.....	See Table 2-17
MSR_MO_PMON_DSP 06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SELO 06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL1 06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL2 06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL3 06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL4 06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL5 06_2EH.....	See Table 2-17
MSR_MO_PMON_ISS 06_2EH.....	See Table 2-17
MSR_MO_PMON_MAP 06_2EH.....	See Table 2-17
MSR_MO_PMON_MM_CONFIG 06_2EH.....	See Table 2-17
MSR_MO_PMON_MSC_THR 06_2EH.....	See Table 2-17
MSR_MO_PMON_PGT 06_2EH.....	See Table 2-17
MSR_MO_PMON_PLD	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH.....	See Table 2-17
MSR_MO_PMON_TIMESTAMP	
06_2EH.....	See Table 2-17
MSR_MO_PMON_ZDP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ADDR_MASK	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ADDR_MATCH	
06_2EH.....	See Table 2-17
MSR_M1_PMON_BOX_CTRL	
06_2EH.....	See Table 2-17
MSR_M1_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-17
MSR_M1_PMON_BOX_STATUS	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL0	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL1	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL2	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL3	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL4	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL5	
06_2EH.....	See Table 2-17
MSR_M1_PMON_DSP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SELO	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL4	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL5	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ISS	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH.....	See Table 2-17
MSR_M1_PMON_MAP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_MM_CONFIG	
06_2EH.....	See Table 2-17
MSR_M1_PMON_MSC_THR	
06_2EH.....	See Table 2-17
MSR_M1_PMON_PGT	
06_2EH.....	See Table 2-17
MSR_M1_PMON_PLD	
06_2EH.....	See Table 2-17
MSR_M1_PMON_TIMESTAMP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ZDP	
06_2EH.....	See Table 2-17
IA32_MCO_MISC / MSR_MCO_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_MCO_RESIDENCY	
06_57H, 06_85H.....	See Table 2-53
IA32_MC1_MISC / MSR_MC1_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
IA32_MC10_ADDR / MSR_MC10_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC10_CTL / MSR_MC10_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC10_MISC / MSR_MC10_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC10_STATUS / MSR_MC10_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC11_ADDR / MSR_MC11_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC11_CTL / MSR_MC11_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC11_MISC / MSR_MC11_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC11_STATUS / MSR_MC11_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC12_ADDR / MSR_MC12_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC12_CTL / MSR_MC12_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
IA32_MC12_MISC / MSR_MC12_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC12_STATUS / MSR_MC12_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_ADDR / MSR_MC13_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_CTL / MSR_MC13_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_MISC / MSR_MC13_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_STATUS / MSR_MC13_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC14_ADDR / MSR_MC14_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
IA32_MC14_CTL / MSR_MC14_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC14_MISC / MSR_MC14_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC14_STATUS / MSR_MC14_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_ADDR / MSR_MC15_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_CTL / MSR_MC15_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_MISC / MSR_MC15_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_STATUS / MSR_MC15_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
IA32_MC16_ADDR / MSR_MC16_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC16_CTL / MSR_MC16_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC16_MISC / MSR_MC16_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC16_STATUS / MSR_MC16_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC17_ADDR / MSR_MC17_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC17_CTL / MSR_MC17_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC17_MISC / MSR_MC17_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC17_STATUS / MSR_MC17_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_ADDR / MSR_MC18_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_CTL / MSR_MC18_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_MISC / MSR_MC18_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_STATUS / MSR_MC18_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_ADDR / MSR_MC19_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_CTL / MSR_MC19_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_MISC / MSR_MC19_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_STATUS / MSR_MC19_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC2_MISC / MSR_MC2_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
IA32_MC20_ADDR / MSR_MC20_ADDR	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC20_CTL / MSR_MC20_CTL	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC20_MISC / MSR_MC20_MISC	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC20_STATUS / MSR_MC20_STATUS	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC21_ADDR / MSR_MC21_ADDR	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC21_CTL / MSR_MC21_CTL	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC21_MISC / MSR_MC21_MISC	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC21_STATUS / MSR_MC21_STATUS	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC22_ADDR / MSR_MC22_ADDR	
06_3EH.....	See Table 2-26
IA32_MC22_CTL / MSR_MC22_CTL	
06_3EH.....	See Table 2-26
IA32_MC22_MISC / MSR_MC22_MISC	
06_3EH.....	See Table 2-26
IA32_MC22_STATUS / MSR_MC22_STATUS	
06_3EH.....	See Table 2-26
IA32_MC23_ADDR / MSR_MC23_ADDR	
06_3EH.....	See Table 2-26
IA32_MC23_CTL / MSR_MC23_CTL	
06_3EH.....	See Table 2-26
IA32_MC23_MISC / MSR_MC23_MISC	
06_3EH.....	See Table 2-26
IA32_MC23_STATUS / MSR_MC23_STATUS	
06_3EH.....	See Table 2-26
IA32_MC24_ADDR / MSR_MC24_ADDR	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
IA32_MC24_CTL / MSR_MC24_CTL	
06_3EH.....	See Table 2-26
IA32_MC24_MISC / MSR_MC24_MISC	
06_3EH.....	See Table 2-26
IA32_MC24_STATUS / MSR_MC24_STATUS	
06_3EH.....	See Table 2-26
IA32_MC25_ADDR / MSR_MC25_ADDR	
06_3EH.....	See Table 2-26
IA32_MC25_CTL / MSR_MC25_CTL	
06_3EH.....	See Table 2-26
IA32_MC25_MISC / MSR_MC25_MISC	
06_3EH.....	See Table 2-26
IA32_MC25_STATUS / MSR_MC25_STATUS	
06_3EH.....	See Table 2-26
IA32_MC26_ADDR / MSR_MC26_ADDR	
06_3EH.....	See Table 2-26
IA32_MC26_CTL / MSR_MC26_CTL	
06_3EH.....	See Table 2-26
IA32_MC26_MISC / MSR_MC26_MISC	
06_3EH.....	See Table 2-26
IA32_MC26_STATUS / MSR_MC26_STATUS	
06_3EH.....	See Table 2-26
IA32_MC27_ADDR / MSR_MC27_ADDR	
06_3EH.....	See Table 2-26
IA32_MC27_CTL / MSR_MC27_CTL	
06_3EH.....	See Table 2-26
IA32_MC27_MISC / MSR_MC27_MISC	
06_3EH.....	See Table 2-26
IA32_MC27_STATUS / MSR_MC27_STATUS	
06_3EH.....	See Table 2-26
IA32_MC28_ADDR / MSR_MC28_ADDR	
06_3EH.....	See Table 2-26
IA32_MC28_CTL / MSR_MC28_CTL	
06_3EH.....	See Table 2-26
IA32_MC28_MISC / MSR_MC28_MISC	
06_3EH.....	See Table 2-26
IA32_MC28_STATUS / MSR_MC28_STATUS	
06_3EH.....	See Table 2-26
IA32_MC29_ADDR / MSR_MC29_ADDR	
06_3EH.....	See Table 2-27
IA32_MC29_CTL / MSR_MC29_CTL	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-27
IA32_MC29_MISC / MSR_MC29_MISC	
06_3EH.....	See Table 2-27
IA32_MC29_STATUS / MSR_MC29_STATUS	
06_3EH.....	See Table 2-27
IA32_MC3_ADDR / MSR_MC3_ADDR	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC3_CTL / MSR_MC3_CTL	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC3_MISC / MSR_MC3_MISC	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_0EH.....	See Table 2-58
IA32_MC3_STATUS / MSR_MC3_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC30_ADDR / MSR_MC30_ADDR	
06_3EH.....	See Table 2-27
IA32_MC30_CTL / MSR_MC30_CTL	
06_3EH.....	See Table 2-27
IA32_MC30_MISC / MSR_MC30_MISC	
06_3EH.....	See Table 2-27
IA32_MC30_STATUS / MSR_MC30_STATUS	
06_3EH.....	See Table 2-27
IA32_MC31_ADDR / MSR_MC31_ADDR	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-27
IA32_MC31_CTL / MSR_MC31_CTL	
06_3EH.....	See Table 2-27
IA32_MC31_MISC / MSR_MC31_MISC	
06_3EH.....	See Table 2-27
IA32_MC31_STATUS / MSR_MC31_STATUS	
06_3EH.....	See Table 2-27
IA32_MC4_ADDR / MSR_MC4_ADDR	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC4_CTL / MSR_MC4_CTL	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC4_CTL2 / MSR_MC4_CTL2	
06_2AH, 06_2DH	See Table 2-20
IA32_MC4_STATUS / MSR_MC4_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_MC5_ADDR / MSR_MC5_ADDR	
06_0FH, 06_17H	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_57H, 06_85H.....	See Table 2-53

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH.....	See Table 2-58
IA32_MC5_CTL / MSR_MC5_CTL	
06_0FH, 06_17H.....	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
IA32_MC5_MISC / MSR_MC5_MISC	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_0EH.....	See Table 2-58
IA32_MC5_STATUS / MSR_MC5_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
IA32_MC6_ADDR / MSR_MC6_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC6_CTL / MSR_MC6_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
MSR_MC6_DEMOTION_POLICY_CONFIG	
06_37H.....	See Table 2-9
IA32_MC6_MISC / MSR_MC6_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
MSR_MC6_RESIDENCY_COUNTER	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_37H.....	See Table 2-9
06_57H, 06_85H.....	See Table 2-53
IA32_CORE_CAPABILITIES (Note there are no architecturally defined bits; all bits are model-specific)	
06_86H.....	See Table 2-14
06_8CH, 06_8DH.....	See Table 2-45
IA32_MC6_STATUS / MSR_MC6_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_ADDR / MSR_MC7_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_CTL / MSR_MC7_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_MISC / MSR_MC7_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_STATUS / MSR_MC7_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC8_ADDR / MSR_MC8_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC8_CTL / MSR_MC8_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC8_MISC / MSR_MC8_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC8_STATUS / MSR_MC8_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC9_ADDR / MSR_MC9_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC9_CTL / MSR_MC9_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC9_MISC / MSR_MC9_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC9_STATUS / MSR_MC9_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
MSR_MCG_MISC	
0FH.....	See Table 2-55
MSR_MCG_R10	
0FH.....	See Table 2-55
MSR_MCG_R11	
0FH.....	See Table 2-55
MSR_MCG_R12	
0FH.....	See Table 2-55
MSR_MCG_R13	
0FH.....	See Table 2-55
MSR_MCG_R14	
0FH.....	See Table 2-55
MSR_MCG_R15	
0FH.....	See Table 2-55
MSR_MCG_R8	
0FH.....	See Table 2-55
MSR_MCG_R9	
0FH.....	See Table 2-55
MSR_MCG_RAX	
0FH.....	See Table 2-55
MSR_MCG_RBP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
OFH.....	See Table 2-55
MSR_MCG_RBX	
OFH.....	See Table 2-55
MSR_MCG_RCX	
OFH.....	See Table 2-55
MSR_MCG_RDI	
OFH.....	See Table 2-55
MSR_MCG_RDX	
OFH.....	See Table 2-55
MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5	
OFH.....	See Table 2-55
MSR_MCG_RFLAGS	
OFH.....	See Table 2-55
MSR_MCG_RIP	
OFH.....	See Table 2-55
MSR_MCG_RSI	
OFH.....	See Table 2-55
MSR_MCG_RSP	
OFH.....	See Table 2-55
MSR_MEMORY_CTRL	
06_86H.....	See Table 2-14
06_7DH, 06_7EH.....	See Table 2-44
06_97H, 06_9AH.....	See Table 2-46
MSR_MISC_FEATURE_CONTROL	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
MSR_MISC_PWR_MGMT	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
MSR_MOB_ESCRO	
OFH.....	See Table 2-55
MSR_MOB_ESCR1	
OFH.....	See Table 2-55
MSR_MS_CCCRO	
OFH.....	See Table 2-55
MSR_MS_CCCR1	
OFH.....	See Table 2-55
MSR_MS_CCCR2	
OFH.....	See Table 2-55
MSR_MS_CCCR3	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
0FH.....	See Table 2-55
MSR_MS_COUNTER0	
0FH.....	See Table 2-55
MSR_MS_COUNTER1	
0FH.....	See Table 2-55
MSR_MS_COUNTER2	
0FH.....	See Table 2-55
MSR_MS_COUNTER3	
0FH.....	See Table 2-55
MSR_MS_ESCRO	
0FH.....	See Table 2-55
MSR_MS_ESCR1	
0FH.....	See Table 2-55
MSR_MTRRCAP	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH....	See Table 2-39
MSR_OFFCORE_RSP_0	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_OFFCORE_RSP_1	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_25H, 06_2CH.....	See Table 2-18
06_2FH.....	See Table 2-19
06_2AH, 06_2DH.....	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_PACKAGE_ENERGY_TIME_STATUS	
06_6AH, 06_6CH.....	See Table 2-51
MSR_PCIE_PLL_RATIO	
06_3FH.....	See Table 2-32
MSR_PCU_PMON_BOX_CTL	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_BOX_FILTER	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_BOX_STATUS	
06_3EH.....	See Table 2-28
06_3FH.....	See Table 2-33
MSR_PCU_PMON_CTRL0	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_PCU_PMON_CTR1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_CTR2	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_CTR3	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSELO	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSEL1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSEL2	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSEL3	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PEBS_DATA_CFG	
06_7DH, 06_7EH.....	See Table 2-44
MSR_PEBS_ENABLE	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH.....	See Table 2-12
06_7AH.....	See Table 2-13
06_86H.....	See Table 2-14
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_3EH.....	See Table 2-27
06_57H, 06_85H.....	See Table 2-53
0FH.....	See Table 2-55
MSR_PEBS_FRONTEND	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH....	See Table 2-39
MSR_PEBS_LD_LAT	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
MSR_PEBS_MATRIX_VERT	
0FH.....	See Table 2-55

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_PEBS_NUM_ALT	
06_2DH.....	See Table 2-23
MSR_PERF_CAPABILITIES	
06_0FH, 06_17H.....	See Table 2-3
MSR_PERF_GLOBAL_CTRL	
06_0FH, 06_17H.....	See Table 2-3
MSR_PERF_GLOBAL_OVF_CTRL	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_PERF_GLOBAL_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_PERF_METRICS	
06_7DH, 06_7EH.....	See Table 2-44
MSR_PERF_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_2AH, 06_2DH.....	See Table 2-20
MSR_PKG_C10_RESIDENCY	
06_5CH, 06_7AH.....	See Table 2-12
06_45H.....	See Table 2-30 and Table 2-31
06_4FH.....	See Table 2-38
MSR_PKG_C2_RESIDENCY	
06_27H.....	See Table 2-5
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_PKG_C3_RESIDENCY	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH.....	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20
06_66H.....	See Table 2-42
06_57H, 06_85H.....	See Table 2-53
MSR_PKG_C4_RESIDENCY	
06_27H.....	See Table 2-5
MSR_PKG_C6_RESIDENCY	
06_27H.....	See Table 2-5
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH.....	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_57H, 06_85H	See Table 2-53
MSR_PKG_C7_RESIDENCY	
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_C8_RESIDENCY	
06_45H	See Table 2-31
06_4FH	See Table 2-38
MSR_PKG_C9_RESIDENCY	
06_45H	See Table 2-31
06_4FH	See Table 2-38
MSR_PKG_CST_CONFIG_CONTROL	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_4CH	See Table 2-11
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-30
06_45H	See Table 2-31
06_3F	See Table 2-32
06_3DH	See Table 2-35
06_56H, 06_4FH	See Table 2-36
06_57H, 06_85H	See Table 2-53
MSR_PKG_ENERGY_STATUS	
06_37H, 06_4AH, 06_4CH, 06_5AH, 06_5DH	See Table 2-8
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3DH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_HDC_CONFIG	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_HDC_DEEP_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_HDC_SHALLOW_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_PERF_STATUS	
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2DH	See Table 2-23

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3AH, 06_3EH	See Table 2-26
06_3CH, 06_3DH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-29
06_57H, 06_85H.....	See Table 2-53
MSR_PKG_POWER_INFO	
06_4DH.....	See Table 2-10
06_5CH, 06_7AH, 06_86H.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3DH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_POWER_LIMIT	
06_37H, 06_4AH, 06_4CH, 06_5AH, 06_5DH	See Table 2-8
06_4DH.....	See Table 2-10
06_5CH, 06_7AH, 06_86H.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3DH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_PKGC_IRTL1	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
MSR_PKGC_IRTL2	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
MSR_PKGC3_IRTL	
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH	See Table 2-20
MSR_PKGC6_IRTL	
06_2AH, 06_2DH	See Table 2-20
MSR_PKGC7_IRTL	
06_2AH.....	See Table 2-21
MSR_PLATFORM_BRV	
0FH.....	See Table 2-55
MSR_PLATFORM_ENERGY_COUNTER	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PLATFORM_ID	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
MSR_PLATFORM_INFO	
06_5CH, 06_7AH	See Table 2-12

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29 and Table 2-30
06_56H, 06_4FH	See Table 2-36
06_57H	See Table 2-53
MSR_PLATFORM_POWER_LIMIT	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PMG_IO_CAPTURE_BASE	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_4CH	See Table 2-11
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_57H	See Table 2-53
MSR_PMH_ESCRO	
0FH	See Table 2-55
MSR_PMH_ESCR1	
0FH	See Table 2-55
MSR_PMON_GLOBAL_CONFIG	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_PMON_GLOBAL_CTL	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_PMON_GLOBAL_STATUS	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_POWER_CTL	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
MSR_PPO_ENERGY_STATUS	
06_37H, 06_4AH, 06_5AH, 06_5DH	See Table 2-8
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H	See Table 2-53
MSR_PPO_POLICY	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_45H	See Table 2-21
MSR_PP0_POWER_LIMIT	
06_4CH.....	See Table 2-11
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H.....	See Table 2-53
MSR_PP1_ENERGY_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PP1_POLICY	
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PP1_POWER_LIMIT	
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PPERF	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
IA32_PPIN / MSR_PPIN	
06_3EH.....	See Table 2-26
06_56H, 06_4FH	See Table 2-36
06_55H.....	See Table 2-50
06_57H, 06_85H	See Table 2-53
IA32_PPIN_CTL / MSR_PPIN_CTL	
06_3EH.....	See Table 2-26
06_56H, 06_4FH	See Table 2-36
06_55H.....	See Table 2-50
06_57H, 06_85H	See Table 2-53
MSR_PRMRR_BASE_0	
06_7DH, 06_7EH	See Table 2-44
MSR_PRMRR_PHYS_BASE	
06_8EH, 06_9EH	See Table 2-41
MSR_PRMRR_PHYS_MASK	
06_8EH, 06_9EH	See Table 2-41
MSR_PRMRR_VALID_CONFIG	
06_8EH, 06_9EH	See Table 2-41
MSR_RELOAD_FIXED_CTRx	
06_86H	See Table 2-14
MSR_RELOAD_PMCx	
06_86H	See Table 2-14
MSR_RING_RATIO_LIMIT	
06_8EH, 06_9EH	See Table 2-41

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_RO_PMON_BOX_CTRL 06_2EH.....	See Table 2-17
MSR_RO_PMON_BOX_OVF_CTRL 06_2EH.....	See Table 2-17
MSR_RO_PMON_BOX_STATUS 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL0 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL1 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL2 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL3 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL4 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL5 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL6 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL7 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SELO 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL1 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL2 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL3 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL4 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL5 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL6 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL7 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P0 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P1 06_2EH.....	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_RO_PMON_IPERFO_P2 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P3 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P4 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P5 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P6 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P7 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P0 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P1 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P2 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P3 06_2EH.....	See Table 2-17
MSR_R1_PMON_BOX_CTRL 06_2EH.....	See Table 2-17
MSR_R1_PMON_BOX_OVF_CTRL 06_2EH.....	See Table 2-17
MSR_R1_PMON_BOX_STATUS 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR10 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR11 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR12 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR13 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR14 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR15 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR8 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR9 06_2EH.....	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_R1_PMON_EVNT_SEL10 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVNT_SEL11 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVNT_SEL12 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVNT_SEL13 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVNT_SEL14 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVNT_SEL15 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVNT_SEL8 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVNT_SEL9 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P10 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P11 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P12 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P13 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P14 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P15 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P8 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P9 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P4 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P5 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P6 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P7 06_2EH.....	See Table 2-17
MSR_RAPL_POWER_UNIT 06_37H, 06_4AH, 06_5AH, 06_5DH.....	See Table 2-8

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-36
06_57H.....	See Table 2-53
MSR_RAT_ESCRO	
0FH.....	See Table 2-55
MSR_RAT_ESCR1	
0FH.....	See Table 2-55
MSR_RING_PERF_LIMIT_REASONS	
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SO_PMON_BOX_CTRL	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_SO_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-17
MSR_SO_PMON_BOX_STATUS	
06_2EH.....	See Table 2-17
MSR_SO_PMON_CTRL0	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_CTRL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_CTRL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_CTRL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_EVNT_SELO	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_S0_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S0_PMON_MASK	
06_2EH.....	See Table 2-17
MSR_S0_PMON_MATCH	
06_2EH.....	See Table 2-17
MSR_S1_PMON_BOX_CTRL	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_S1_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-17
MSR_S1_PMON_BOX_STATUS	
06_2EH.....	See Table 2-17
MSR_S1_PMON_CTRL0	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_CTRL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_CTRL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_CTRL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SELO	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_MASK	
06_2EH.....	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_S1_PMON_MATCH	
06_2EH.....	See Table 2-17
MSR_S2_PMON_BOX_CTL	
06_3FH.....	See Table 2-33
MSR_S2_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL0	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL1	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL2	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL3	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSELO	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSEL1	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSEL2	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSEL3	
06_3FH.....	See Table 2-33
MSR_S3_PMON_BOX_CTL	
06_3FH.....	See Table 2-33
MSR_S3_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL0	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL1	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL2	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL3	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSELO	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSEL1	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSEL2	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSEL3	
06_3FH.....	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_SAAT_ESCR0	
0FH.....	See Table 2-55
MSR_SAAT_ESCR1	
0FH.....	See Table 2-55
MSR_SGXOWNEREPOCH0	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_SGXOWNEREPOCH1	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_SMI_COUNT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_57H.....	See Table 2-53
MSR_SMM_BLOCKED	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SMM_DELAYED	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SMM_FEATURE_CONTROL	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SMM_MCA_CAP	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-36
06_57H.....	See Table 2-53
MSR_SMRR_PHYSBASE	
06_0FH, 06_17H.....	See Table 2-3
MSR_SMRR_PHYSMASK	
06_0FH, 06_17H.....	See Table 2-3
MSR_SSU_ESCR0	
0FH.....	See Table 2-55
MSR_TBPU_ESCR0	
0FH.....	See Table 2-55
MSR_TBPU_ESCR1	
0FH.....	See Table 2-55

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_TC_ESCR0	
0FH.....	See Table 2-55
MSR_TC_ESCR1	
0FH.....	See Table 2-55
MSR_TC_PRECISE_EVENT	
0FH.....	See Table 2-55
MSR_TEMPERATURE_TARGET	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_3EH.....	See Table 2-26
06_56H, 06_4FH.....	See Table 2-36
06_57H.....	See Table 2-53
MSR_TEST_CTRL	
P6 Family.....	See Table 2-60
MSR_THERM2_CTL	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
0FH.....	See Table 2-55
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_THREAD_ID_INFO	
06_3FH.....	See Table 2-32
MSR_TRACE_HUB_STH ACPIBAR_BASE	
06_8EH, 06_9EH.....	See Table 2-41
MSR_TURBO_ACTIVATION_RATIO	
06_5CH, 06_7AH.....	See Table 2-12
06_3AH.....	See Table 2-25
06_3CH, 06_45H, 06_46H.....	See Table 2-29
06_57H.....	See Table 2-53
MSR_TURBO_GROUP_CORECNT	
06_5CH, 06_7AH.....	See Table 2-12
MSR_TURBO_POWER_CURRENT_LIMIT	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_TURBO_RATIO_LIMIT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH.....	See Table 2-15
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH.....	See Table 2-16
06_2EH.....	See Table 2-17
06_25H, 06_2CH.....	See Table 2-18

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2FH.....	See Table 2-19
06_2AH, 06_45H.....	See Table 2-21
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26 and Table 2-27
06_3CH, 06_45H, 06_46H.....	See Table 2-30
06_3FH.....	See Table 2-32
06_3DH.....	See Table 2-35
06_56H, 06_4FH.....	See Table 2-36
06_55H.....	See Table 2-50
06_57H.....	See Table 2-53
MSR_TURBO_RATIO_LIMIT1	
06_3EH.....	See Table 2-26 and Table 2-27
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-36
MSR_TURBO_RATIO_LIMIT2	
06_3FH.....	See Table 2-32
MSR_TURBO_RATIO_LIMIT3	
06_56H.....	See Table 2-37
06_4FH.....	See Table 2-38
MSR_TURBO_RATIO_LIMIT_CORES	
06_55H.....	See Table 2-50
MSR_U_PMON_BOX_STATUS	
06_3EH.....	See Table 2-28
06_3FH.....	See Table 2-33
MSR_U_PMON_CTR	
06_2EH.....	See Table 2-17
MSR_U_PMON_CTR0	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_U_PMON_CTR1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_U_PMON_EVNT_SEL	
06_2EH.....	See Table 2-17
MSR_U_PMON_EVNTSELO	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_U_PMON_EVNTSEL1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_U_PMON_GLOBAL_CTRL	
06_2EH	See Table 2-17
MSR_U_PMON_GLOBAL_OVF_CTRL	
06_2EH	See Table 2-17
MSR_U_PMON_GLOBAL_STATUS	
06_2EH	See Table 2-17
MSR_U_PMON_UCLK_FIXED_CTL	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U_PMON_UCLK_FIXED_CTR	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U2L_ESCR0	
0FH	See Table 2-55
MSR_U2L_ESCR1	
0FH	See Table 2-55
MSR_UNC_ARB_PERFCTRO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTRO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTR2	
06_2AH	See Table 2-22

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_0_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_0_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_0_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_0_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_UNIT_STATUS	
06_2AH	See Table 2-22

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_2_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_3_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR0	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR1	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSELO	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL1	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_CONFIG	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_FIXED_CTR	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_FIXED_CTRL	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_GLOBAL_CTRL	
06_2AH	See Table 2-22

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_GLOBAL_STATUS	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNCORE_ADDR_OPCODE_MATCH	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_FIXED_CTR_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_FIXED_CTR0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSELO	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL1	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL2	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL3	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL4	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL5	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL6	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL7	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC1	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC2	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC3	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNCORE_PMC4 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC5 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
06_2EH	See Table 2-17
MSR_UNCORE_PMC6 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC7 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PRMRR_BASE 06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_UNCORE_PRMRR_MASK 06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_UNCORE_PRMRR_PHYS_BASE 06_8EH, 06_9EH	See Table 2-41
MSR_UNCORE_PRMRR_PHYS_MASK 06_8EH, 06_9EH	See Table 2-41
MSR_VR_CURRENT_CONFIG 06_8CH, 06_8DH	See Table 2-45
MSR_W_PMON_BOX_CTRL 06_2EH	See Table 2-17
MSR_W_PMON_BOX_OVF_CTRL 06_2EH	See Table 2-17
MSR_W_PMON_BOX_STATUS 06_2EH	See Table 2-17
MSR_W_PMON_CTRL0 06_2EH	See Table 2-17
MSR_W_PMON_CTRL1 06_2EH	See Table 2-17
MSR_W_PMON_CTRL2 06_2EH	See Table 2-17
MSR_W_PMON_CTRL3 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SELO 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL1 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL2 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL3	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
MSR_W_PMON_FIXED_CTR	
06_2EH	See Table 2-17
MSR_W_PMON_FIXED_CTR_CTL	
06_2EH	See Table 2-17
MSR_WEIGHTED_CORE_CO	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MTRRfix16K_80000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix16K_A0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_C0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_C8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_D0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_D8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_E0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_E8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_F0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_F8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix64K_00000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase0	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase1	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase2	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase3	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase4	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase5	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase6	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase7	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask0	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask1	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask2	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask3	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask4	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask5	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask6	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask7	
06_0EH	See Table 2-58
P6 Family	See Table 2-60