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

Documentation Changes

March 2024

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: 252046-075

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.

Contents

Revision History

Revision	Description	Date
-001	• Initial release	November 2002
-002	• 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	• 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	• Removed Documentation changes 1-17. • Added Documentation changes 1-24.	June 2003
-005	• Removed Documentation Changes 1-24. • Added Documentation Changes 1-15.	September 2003
-006	• Added Documentation Changes 16- 34.	November 2003
-007	• Updated Documentation changes 14, 16, 17, and 28. • Added Documentation Changes 35-45.	January 2004
-008	• Removed Documentation Changes 1-45. • Added Documentation Changes 1-5.	March 2004
-009	• Added Documentation Changes 7-27.	May 2004
-010	• Removed Documentation Changes 1-27. • Added Documentation Changes 1.	August 2004
-011	• Added Documentation Changes 2-28.	November 2004
-012	• Removed Documentation Changes 1-28. • Added Documentation Changes 1-16.	March 2005
-013	• Updated title. • There are no Documentation Changes for this revision of the document.	July 2005
-014	• Added Documentation Changes 1-21.	September 2005
-015	• Removed Documentation Changes 1-21. • Added Documentation Changes 1-20.	March 9, 2006
-016	• Added Documentation changes 21-23.	March 27, 2006
-017	• Removed Documentation Changes 1-23. • Added Documentation Changes 1-36.	September 2006
-018	• Added Documentation Changes 37-42.	October 2006
-019	• Removed Documentation Changes 1-42. • Added Documentation Changes 1-19.	March 2007
-020	• Added Documentation Changes 20-27.	May 2007
-021	• Removed Documentation Changes 1-27. • Added Documentation Changes 1-6	November 2007
-022	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-6	August 2008
-023	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-21	March 2009

Revision	Description	Date
-024	• Removed Documentation Changes 1-21 • Added Documentation Changes 1-16	June 2009
-025	• Removed Documentation Changes 1-16 • Added Documentation Changes 1-18	September 2009
-026	• Removed Documentation Changes 1-18 • Added Documentation Changes 1-15	December 2009
-027	• Removed Documentation Changes 1-15 • Added Documentation Changes 1-24	March 2010
-028	• Removed Documentation Changes 1-24 • Added Documentation Changes 1-29	June 2010
-029	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	September 2010
-030	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	January 2011
-031	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	April 2011
-032	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-14	May 2011
-033	• Removed Documentation Changes 1-14 • Added Documentation Changes 1-38	October 2011
-034	• Removed Documentation Changes 1-38 • Added Documentation Changes 1-16	December 2011
-035	• Removed Documentation Changes 1-16 • Added Documentation Changes 1-18	March 2012
-036	• Removed Documentation Changes 1-18 • Added Documentation Changes 1-17	May 2012
-037	• Removed Documentation Changes 1-17 • Added Documentation Changes 1-28	August 2012
-038	• Removed Documentation Changes 1-28 • Add Documentation Changes 1-22	January 2013
-039	• Removed Documentation Changes 1-22 • Add Documentation Changes 1-17	June 2013
-040	• Removed Documentation Changes 1-17 • Add Documentation Changes 1-24	September 2013
-041	• Removed Documentation Changes 1-24 • Add Documentation Changes 1-20	February 2014
-042	• Removed Documentation Changes 1-20 • Add Documentation Changes 1-8	February 2014
-043	• Removed Documentation Changes 1-8 • Add Documentation Changes 1-43	June 2014
-044	• Removed Documentation Changes 1-43 • Add Documentation Changes 1-12	September 2014
-045	• Removed Documentation Changes 1-12 • Add Documentation Changes 1-22	January 2015
-046	• Removed Documentation Changes 1-22 • Add Documentation Changes 1-25	April 2015
-047	• Removed Documentation Changes 1-25 • Add Documentation Changes 1-19	June 2015

Revision	Description	Date
-048	• Removed Documentation Changes 1-19 • Add Documentation Changes 1-33	September 2015
-049	• Removed Documentation Changes 1-33 • Add Documentation Changes 1-33	December 2015
-050	• Removed Documentation Changes 1-33 • Add Documentation Changes 1-9	April 2016
-051	• Removed Documentation Changes 1-9 • Add Documentation Changes 1-20	June 2016
-052	• Removed Documentation Changes 1-20 • Add Documentation Changes 1-22	September 2016
-053	• Removed Documentation Changes 1-22 • Add Documentation Changes 1-26	December 2016
-054	• Removed Documentation Changes 1-26 • Add Documentation Changes 1-20	March 2017
-055	• Removed Documentation Changes 1-20 • Add Documentation Changes 1-28	July 2017
-056	• Removed Documentation Changes 1-28 • Add Documentation Changes 1-18	October 2017
-057	• Removed Documentation Changes 1-18 • Add Documentation Changes 1-29	December 2017
-058	• Removed Documentation Changes 1-29 • Add Documentation Changes 1-17	March 2018
-059	• Removed Documentation Changes 1-17 • Add Documentation Changes 1-24	May 2018
-060	• Removed Documentation Changes 1-24 • Add Documentation Changes 1-23	November 2018
-061	• Removed Documentation Changes 1-23 • Add Documentation Changes 1-21	January 2019
-062	• Removed Documentation Changes 1-21 • Add Documentation Changes 1-28	May 2019
-063	• Removed Documentation Changes 1-28 • Add Documentation Changes 1-34	October 2019
-064	• Removed Documentation Changes 1-34 • Add Documentation Changes 1-36	May 2020
-065	• Removed Documentation Changes 1-36 • Add Documentation Changes 1-31	November 2020
-066	• Removed Documentation Changes 1-31 • Add Documentation Changes 1-24	April 2021
-067	• Removed Documentation Changes 1-24 • Add Documentation Changes 1-30	June 2021
-068	• Removed Documentation Changes 1-30 • Add Documentation Changes 1-29	December 2021
-069	• Removed Documentation Changes 1-29 • Add Documentation Changes 1-18	April 2022
-070	• Removed Documentation Changes 1-18 • Add Documentation Changes 1-41	December 2022
-071	• Removed Documentation Changes 1-41 • Add Documentation Changes 1-23	March 2023

Revision	Description	Date
-072	• Removed Documentation Changes 1-23 • Add Documentation Changes 1-19	June 2023
-073	• Removed Documentation Changes 1-19 • Add Documentation Changes 1-19	September 2023
-074	• Removed Documentation Changes 1-19 • Add Documentation Changes 1-20	December 2023
-075	• Removed Documentation Changes 1-20 • Add Documentation Changes 1-20	March 2024

§

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

A violet 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 9, Volume 1
3	Updates to Chapter 11, Volume 1
4	Updates to Chapter 18, Volume 1
5	Updates to Chapter 2, Volume 2A
6	Updates to Chapter 3, Volume 2A
7	Updates to Chapter 4, Volume 2B
8	Updates to Chapter 5, Volume 2C
9	Updates to Chapter 11, Volume 3A
10	Updates to Chapter 16, Volume 3B
11	Updates to Chapter 18, 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 33, Volume 3C
17	Updates to Chapter 35, Volume 3D
18	Updates to Chapter 37, Volume 3D
19	Updates to Chapter 38, Volume 3D
20	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 violet 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:

- Added the "Uncore PMI" feature with a planned removal date of 2026 onwards in Section 2.4, "Planned Removal of Intel® Instruction Set Architecture and Features from Upcoming Products."
- Added the VP2INTERSECT instruction with a removal date of 2023 onwards in Section 2.5, "Intel® Instruction Set Architecture and Features Removed."

CHAPTER 2
INTEL® 64 AND IA-32 ARCHITECTURES

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

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 l data and instruction caches were enlarged to 16 KBytes each, and Level 2 cache sizes of 256 KBytes, 512 KBytes, and 1 MBytes 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 MBytes 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 support 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 channels 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.

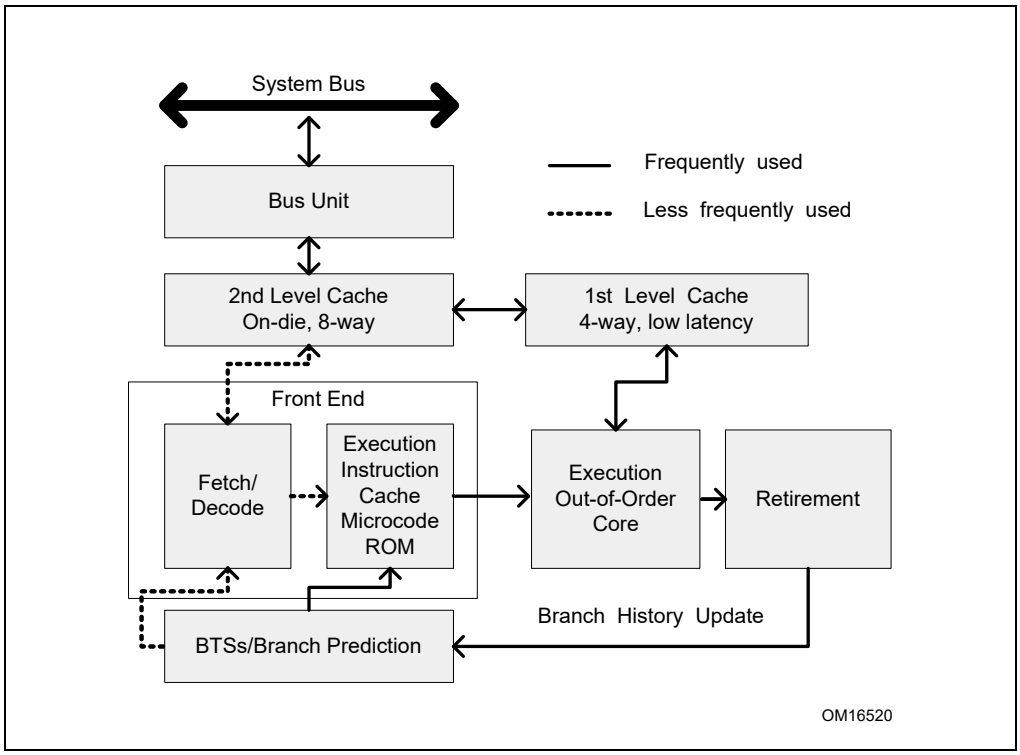


Figure 2-1. The P6 Processor Microarchitecture with Advanced Transfer Cache Enhancement

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 pipeline[1].
  - — 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.

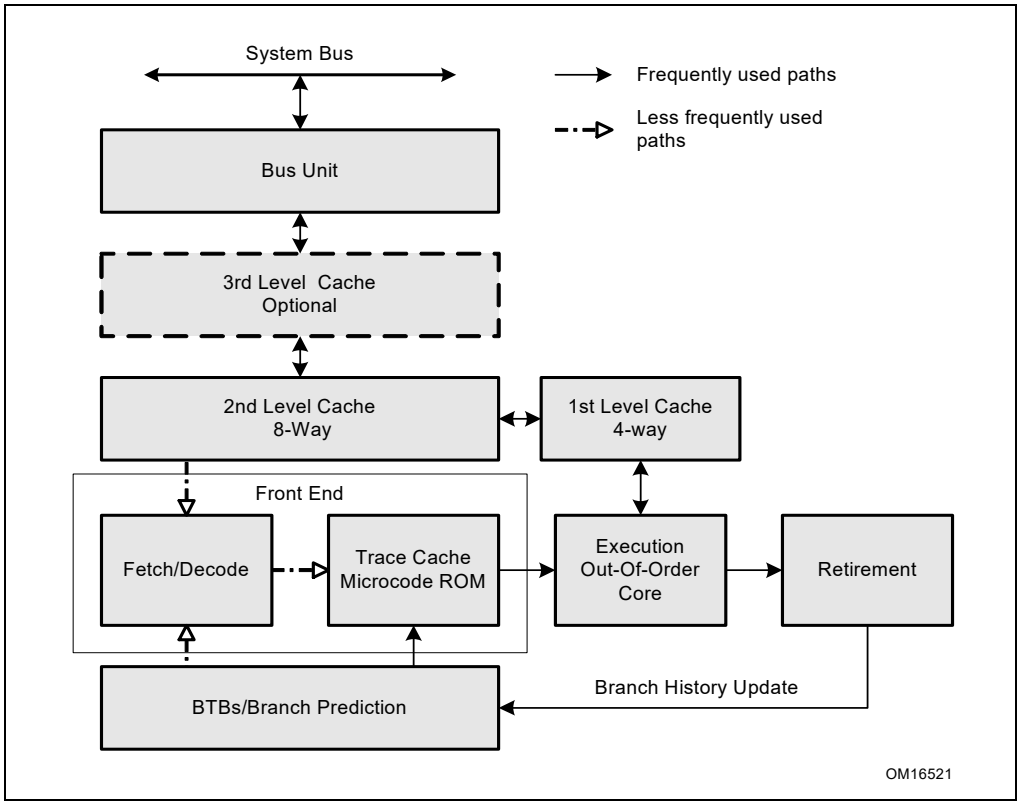


Figure 2-2. The Intel NetBurst® Microarchitecture

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 instructions with single-cycle throughput and floating-point operations.
  - — Single-cycle throughput of most 128-bit SIMD instructions.
  - — Up to eight floating-point operations 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.

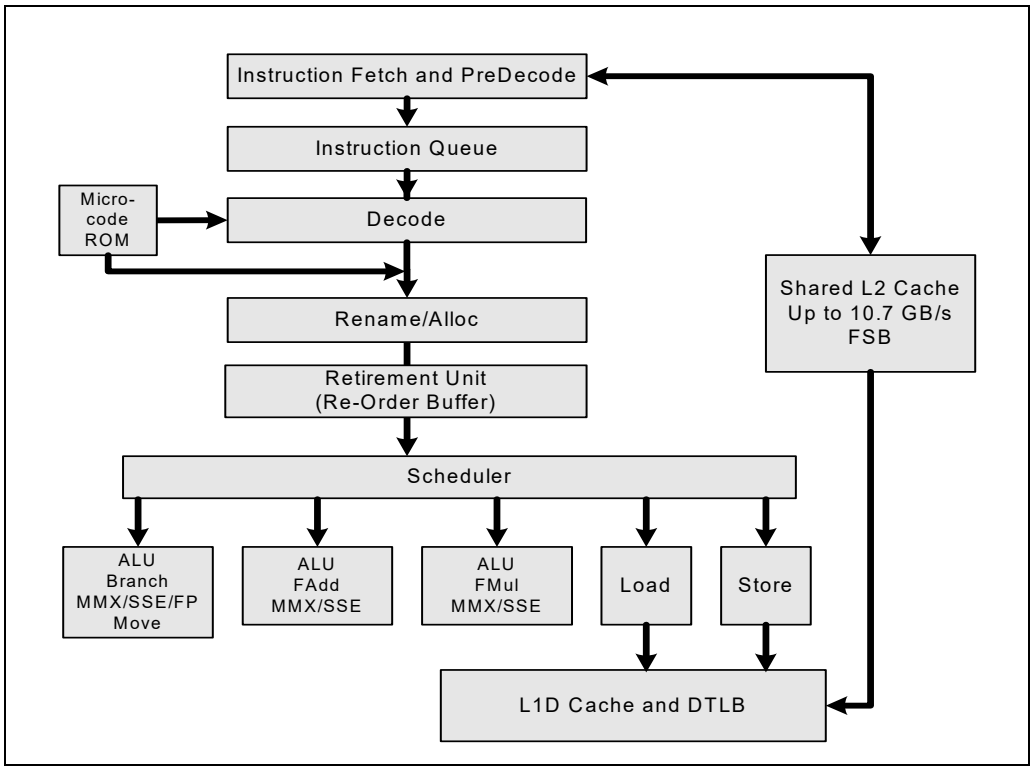


Figure 2-3. The Intel® Core™ Microarchitecture Pipeline Functionality

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 caches 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."

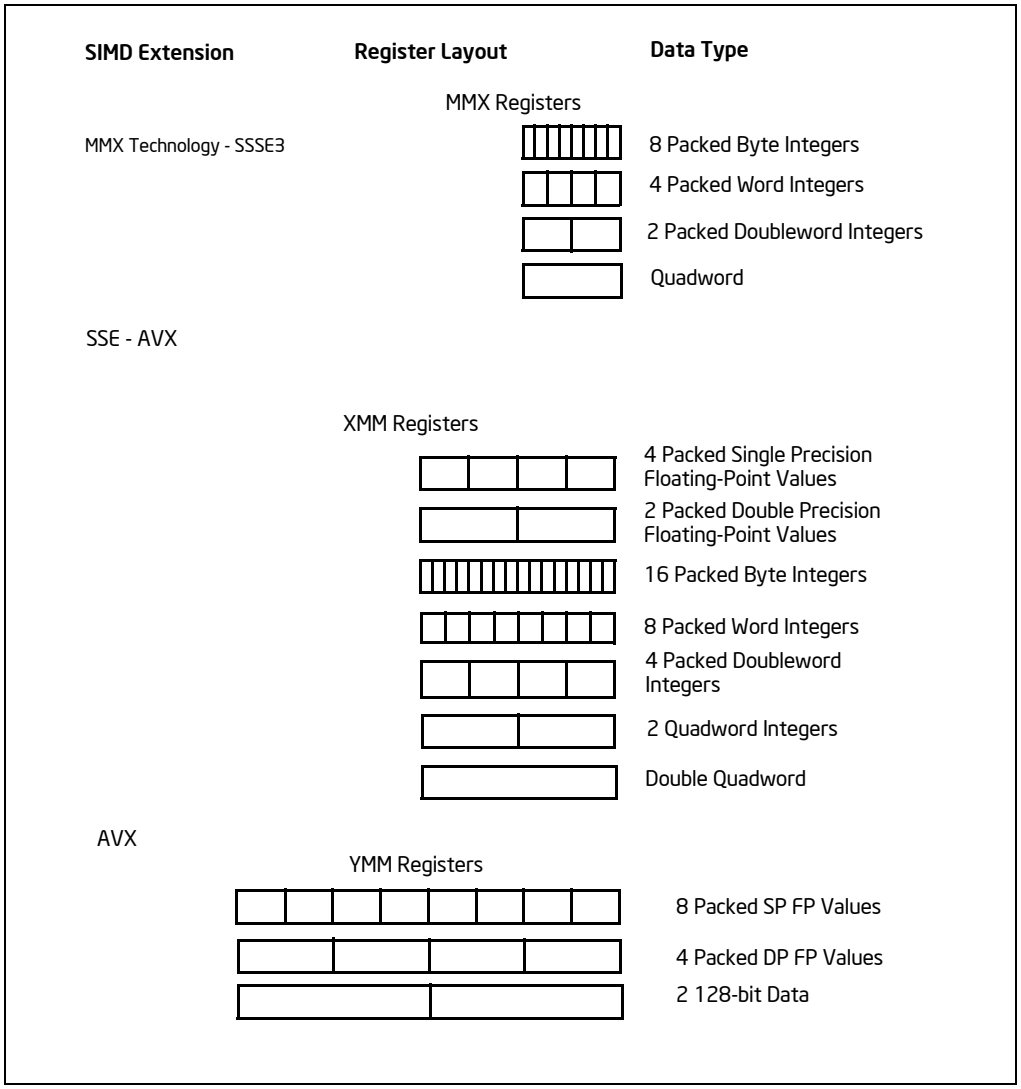


Figure 2-4. SIMD Extensions, Register Layouts, and Data Types

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.

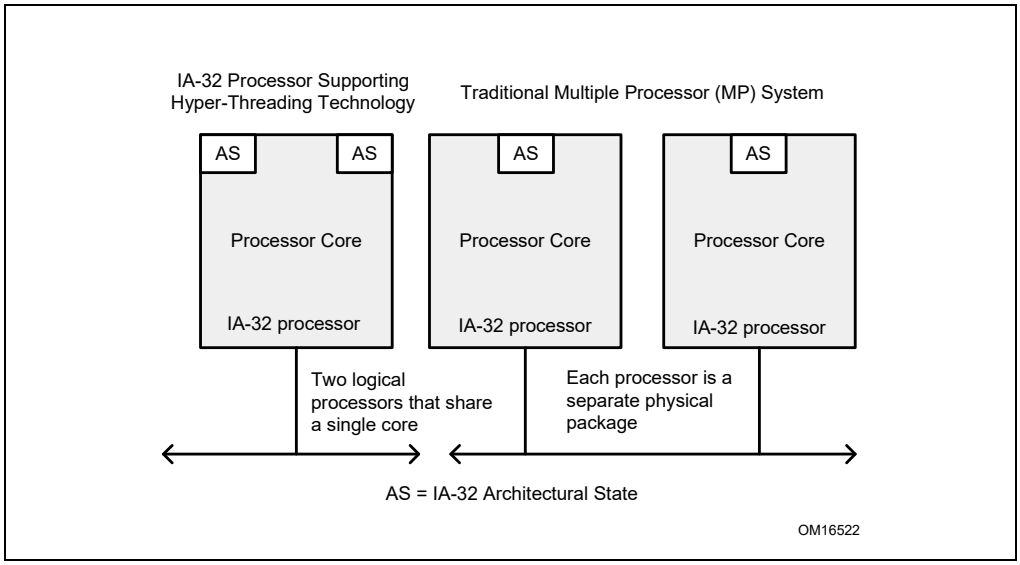


Figure 2-5. Comparison of an IA-32 Processor Supporting Intel® Hyper-Threading Technology and a Traditional Dual Processor System

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 threads[1] 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.

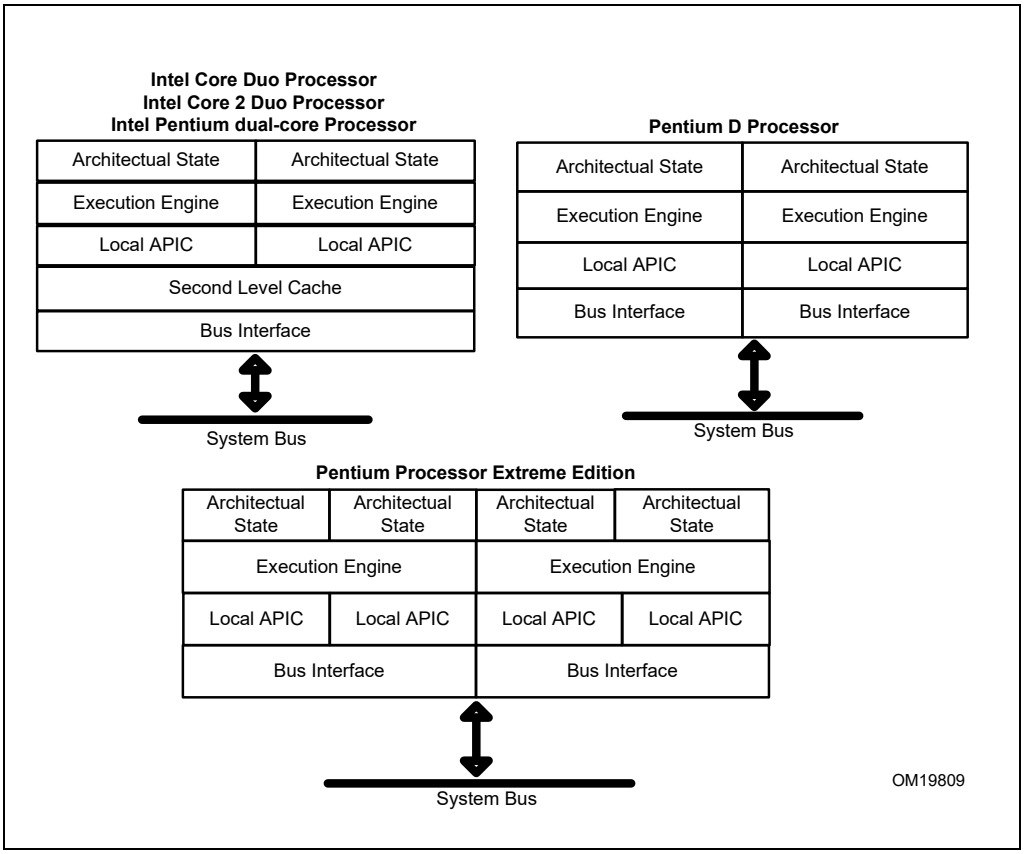


Figure 2-6. Intel 64 and IA-32 Processors that Support Dual-Core

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.

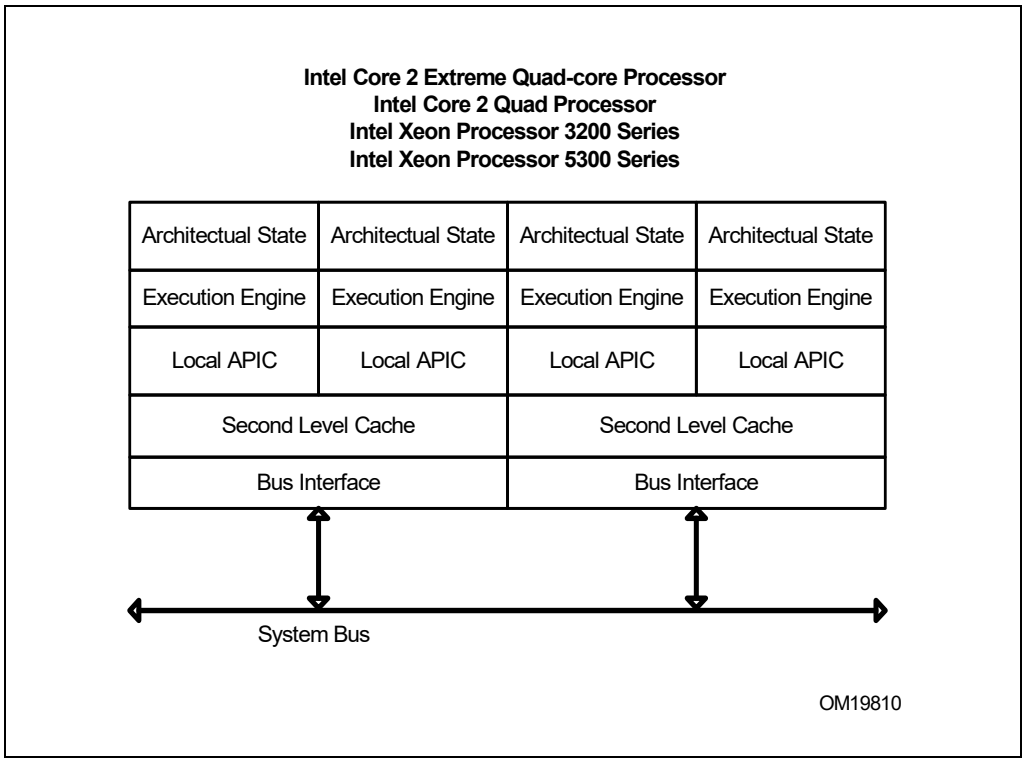


Figure 2-7. Intel® 64 Processors that Support Quad-Core

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 channels to DDR3 memory.

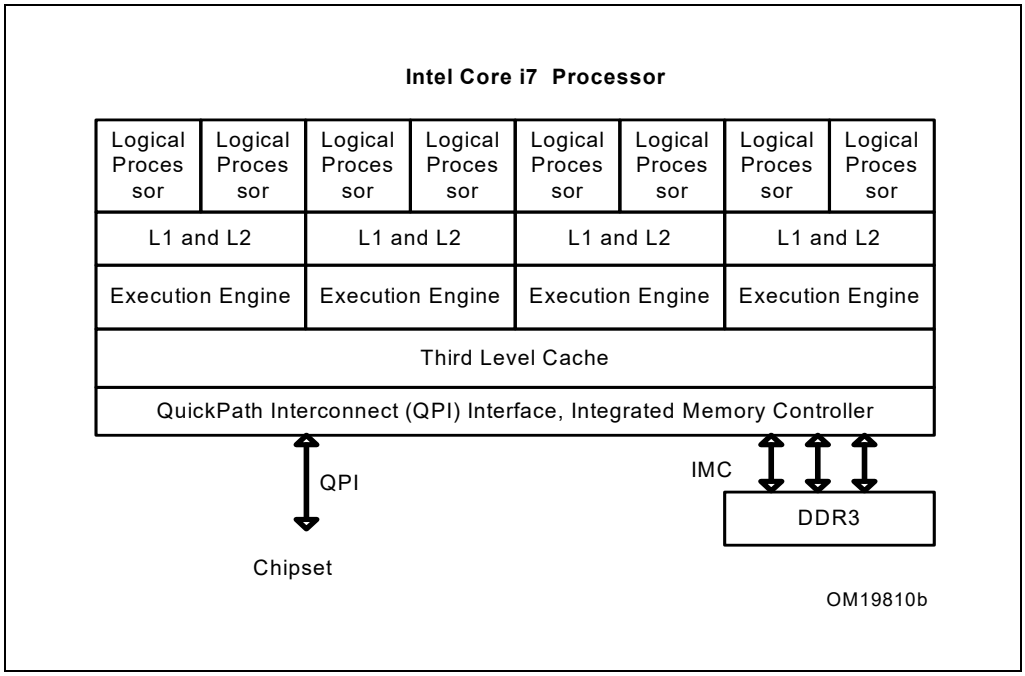


Figure 2-8. Intel® Core™ i7 Processor

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 the 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 head-room 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 Intro-duced	Microarchitecture	Top-Bin Clock Frequency at Introduction	Tran-sistors	Register Sizes[1]	System Bus Band-width	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	152 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	5.3 GB/s	4 GB	L1: 64 KB L2: 2 MB (2 MB Total)
Intel Atom Processor Z5xx series	2008	Intel Atom Microarchitecture; Intel Virtualization Technology.	1.86 GHz - 800 MHz	47 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	Up to 4.2 GB/s	4 GB	L1: 56 KB[4] L2: 512 KB

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 Intro-duced	Micro-architec-ture	Highest Processor Base Fre-quency at Intro-duction	Tran-sistors	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 µop 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	675 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	5.3 GB/s [1]	1024 GB (1 TB)	12K µop 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 Intro-duced	Micro-architec-ture	Highest Processor Base Fre-quency at Intro-duction	Tran-sistors	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 µop 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 µop Execution Trace Cache; 16 KB L1; 1 MB L2 (2 MB Total)
Dual-Core Intel Xeon Processor 7041	2005	Intel NetBurst Microarchitecture; Intel Hyper-Threading Technology; Intel 64 Architecture; Dual-core [3]	3.00 GHz	321 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K µop Execution Trace Cache; 16 KB L1; 2 MB L2 (4 MB 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 µop Execution Trace Cache; 16 KB L1; 2 MB L2
Intel Pentium Processor Extreme Edition 955	2006	Intel NetBurst Microarchitecture; Intel 64 Architecture; Dual Core; Intel Virtualization Technology.	3.46 GHz	376 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	64 GB	12K µop Execution Trace Cache; 16 KB L1; 2 MB L2 (4 MB Total)
Intel Core 2 Extreme Processor X6800	2006	Intel Core Microarchitecture; Dual Core; Intel 64 Architecture; Intel Virtualization Technology.	2.93 GHz	291 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	64 GB	L1: 64 KB L2: 4 MB (4 MB Total)

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

Intel Processor	Date Intro-duced	Micro-architec-ture	Highest Processor Base Fre-quency at Intro-duction	Tran-sistors	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	291 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	10.6 GB/s	64 GB	L1: 64 KB L2: 4 MB (4 MB 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: 1 MB (2 MB Total) L3: 16 MB (16 MB Total)
Intel Core 2 Extreme Processor QX6700	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	8.5 GB/s	64 GB	L1: 64 KB L2: 4 MB (4 MB 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: 4 MB (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: 4 MB (4 MB 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: 4 MB (8 MB Total)

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

Intel Processor	Date Intro-duced	Micro-architec-ture	Highest Processor Base Fre-quency at Intro-duction	Tran-sistors	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: 6 MB (12 MB 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: 512 KB
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: 3 MB (9 MB Total) L3: 16 MB
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: 512 KB (1 MB 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: 256 KB L3: 8 MB

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

Intel Processor	Date Intro-duced	Micro-architec-ture	Highest Processor Base Fre-quency at Intro-duction	Tran-sistors	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: 256 KB L3: 4 MB
Intel Xeon-Processor 5680	2010	Intel Turbo Boost Technology, Westmere microarchitecture; Six core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology.	3.33 GHz	1.1 B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; 32 GB/s	1 TB	L1: 64 KB L2: 256 KB L3: 12 MB
Intel Xeon-Processor 7560	2010	Intel Turbo Boost Technology, Nehalem microarchitecture; Eight core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology.	2.26 GHz	2.3 B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; Memory: 76 GB/s	16 TB	L1: 64 KB L2: 256 KB L3: 24 MB
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	995 M	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: 256 KB L3: 8 MB

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

Intel Processor	Date Intro-duced	Micro-architec-ture	Highest Processor Base Fre-quency at Intro-duction	Tran-sistors	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: 256 KB L3: 8 MB
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	2.2 B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; Memory: 102 GB/s	16 TB	L1: 64 KB L2: 256 KB L3: 30 MB

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 in 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 in 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 Intro-duced	Max. Clock Frequency/ Technology at Introduction	Tran-sistors	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 µop Execution Trace Cache; L1: 8 KB 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 µop Execution Trace Cache; L1: 8 KB L2: 512 KB
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 µop Execution Trace Cache; L1: 8 KB L2: 512 KB
Pentium M Processor	2003	1.60 GHz, Intel NetBurst Microarchitecture	77 M	32 GP 80 FPU 64 MMX 128 XMM	64	4 GB	L1: 64 KB 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 µop Execution Trace Cache; L1: 16 KB L2: 1 MB

NOTES:

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
Uncore PMI. IA32_DEBUGCTL MSR, bit 13 (MSR address 1D9H)	2026 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
VP2INTERSECT	2023 onwards

2. Updates to Chapter 9, Volume 1

Change bars and violet text show changes to Chapter 9 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

Changes to this chapter:

- Updated Table 9-3 to remove the "Branch Hint Prefixes" entry in Section 9.6.9, "Effect of Instruction Prefixes on MMX Instructions." Branch hint prefixes do not apply to MMX instructions.

CHAPTER 9
PROGRAMMING WITH INTEL® MMX™ TECHNOLOGY

The Intel MMX technology was introduced into the IA-32 architecture in the Pentium II processor family and Pentium processor with MMX technology. The extensions introduced in MMX technology support a single-instruction, multiple-data (SIMD) execution model that is designed to accelerate the performance of advanced media and communications applications.

This chapter describes MMX technology.

9.1 OVERVIEW OF MMX TECHNOLOGY

MMX technology defines a simple and flexible SIMD execution model to handle 64-bit packed integer data. This model adds the following features to the IA-32 architecture, while maintaining backwards compatibility with all IA-32 applications and operating-system code:

- Eight new 64-bit data registers, called MMX registers.
- Three new packed data types:
  - 64-bit packed byte integers (signed and unsigned).
  - 64-bit packed word integers (signed and unsigned).
  - 64-bit packed doubleword integers (signed and unsigned).
- Instructions that support the new data types and to handle MMX state management.
- Extensions to the CPUID instruction.

MMX technology is accessible from all the IA32-architecture execution modes (protected mode, real address mode, and virtual 8086 mode). It does not add any new modes to the architecture.

The following sections of this chapter describe MMX technology's programming environment, including MMX register set, data types, and instruction set. Additional instructions that operate on MMX registers have been added to the IA-32 architecture by the SSE/SSE2 extensions.

For more information, see:

- Section 10.4.4, "Intel® SSE 64-Bit SIMD Integer Instructions," describes MMX instructions added to the IA-32 architecture with the SSE extensions.
- Section 11.4.2, "Intel® SSE2 64-Bit and 128-Bit SIMD Integer Instructions," describes MMX instructions added to the IA-32 architecture with SSE2 extensions.
- The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D, gives detailed descriptions of MMX instructions.
- Chapter 13, "Intel® MMX™ Technology System Programming," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, describes the manner in which MMX technology is integrated into the IA-32 system programming model.

9.2 THE MMX TECHNOLOGY PROGRAMMING ENVIRONMENT

Figure 9-1 shows the execution environment for MMX technology. All MMX instructions operate on MMX registers, the general-purpose registers, and/or memory as follows:

- **MMX registers —** These eight registers (see Figure 9-1) are used to perform operations on 64-bit packed integer data. They are named MM0 through MM7.

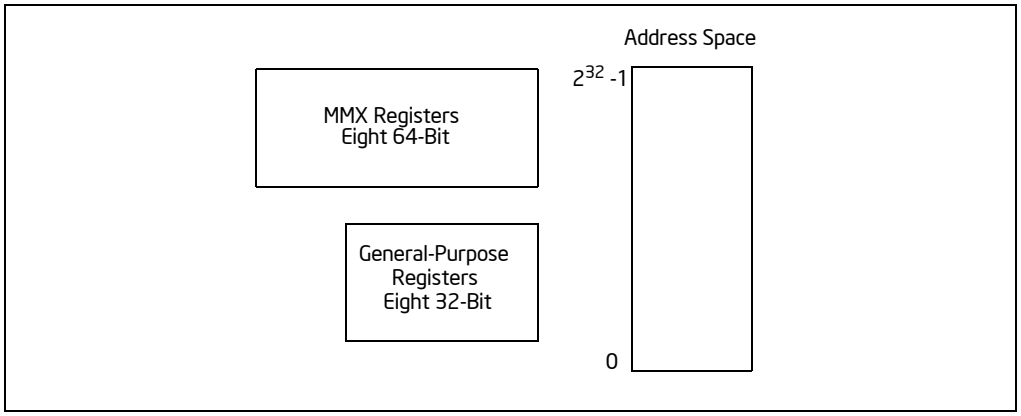


Figure 9-1. MMX Technology Execution Environment

- **General-purpose registers —** The eight general-purpose registers (see Figure 3-5) are used with existing IA-32 addressing modes to address operands in memory. (MMX registers cannot be used to address memory). General-purpose registers are also used to hold operands for some MMX technology operations. They are EAX, EBX, ECX, EDX, EBP, ESI, EDI, and ESP.

9.2.1 MMX Technology in 64-Bit Mode and Compatibility Mode

In compatibility mode and 64-bit mode, MMX instructions function like they do in protected mode. Memory operands are specified using the ModR/M, SIB encoding described in Section 3.7.5.

9.2.2 MMX Registers

The MMX register set consists of eight 64-bit registers (see Figure 9-2), that are used to perform calculations on the MMX packed integer data types. Values in MMX registers have the same format as a 64-bit quantity in memory.

The MMX registers have two data access modes: 64-bit access mode and 32-bit access mode. The 64-bit access mode is used for:

- 64-bit memory accesses.
- 64-bit transfers between MMX registers.
- All pack, logical, and arithmetic instructions.
- Some unpack instructions.

The 32-bit access mode is used for:

- 32-bit memory accesses.
- 32-bit transfer between general-purpose registers and MMX registers.
- Some unpack instructions.

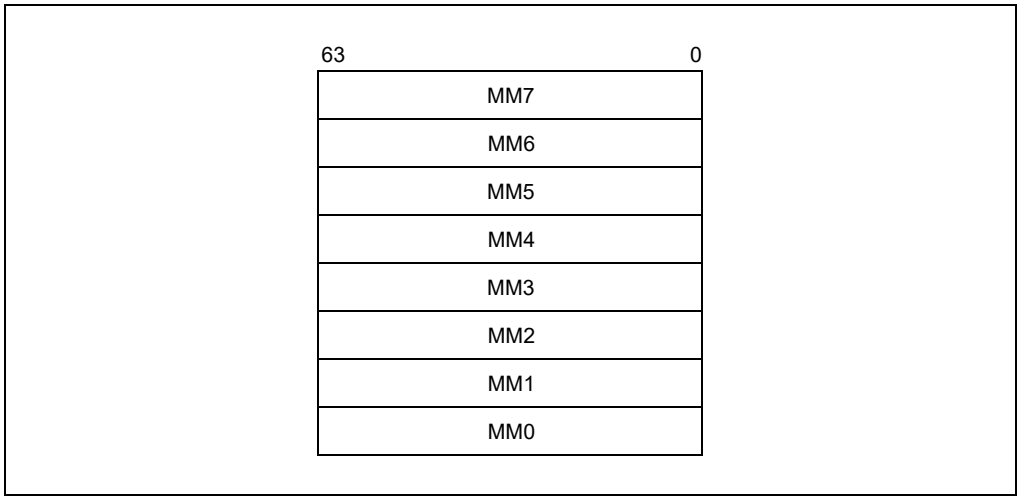


Figure 9-2. MMX Register Set

Although MMX registers are defined in the IA-32 architecture as separate registers, they are aliased to the registers in the FPU data register stack (R0 through R7).

See also Section 9.5, "Compatibility with x87 FPU Architecture."

9.2.3 MMX Data Types

MMX technology introduced the following 64-bit data types to the IA-32 architecture (see Figure 9-3):

- 64-bit packed byte integers — eight packed bytes.
- 64-bit packed word integers — four packed words.
- 64-bit packed doubleword integers — two packed doublewords.

MMX instructions move 64-bit packed data types (packed bytes, packed words, or packed doublewords) and the quadword data type between MMX registers and memory or between MMX registers in 64-bit blocks. However, when performing arithmetic or logical operations on the packed data types, MMX instructions operate in parallel on the individual bytes, words, or doublewords contained in MMX registers; see Section 9.2.5, "Single Instruction, Multiple Data (SIMD) Execution Model."

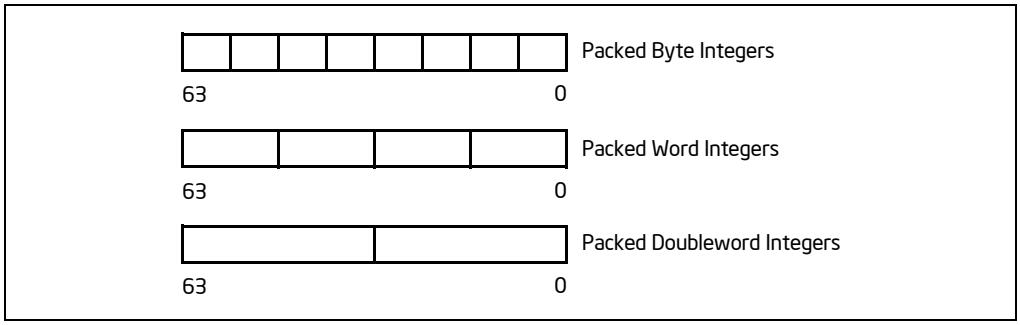


Figure 9-3. Data Types Introduced with the MMX Technology

9.2.4 Memory Data Formats

When stored in memory: bytes, words, and doublewords in the packed data types are stored in consecutive addresses. The least significant byte, word, or doubleword is stored at the lowest address and the most significant byte, word, or doubleword is stored at the high address. The ordering of bytes, words, or doublewords in memory is always little endian. That is, the bytes with the low addresses are less significant than the bytes with high addresses.

9.2.5 Single Instruction, Multiple Data (SIMD) Execution Model

MMX technology uses the single instruction, multiple data (SIMD) technique for performing arithmetic and logical operations on bytes, words, or doublewords packed into MMX registers (see Figure 9-4). For example, the PADDSW instruction adds 4 signed word integers from one source operand to 4 signed word integers in a second source operand and stores 4 word integer results in a destination operand. This SIMD technique speeds up software performance by allowing the same operation to be carried out on multiple data elements in parallel. MMX technology supports parallel operations on byte, word, and doubleword data elements when contained in MMX registers.

The SIMD execution model supported in the MMX technology directly addresses the needs of modern media, communications, and graphics applications, which often use sophisticated algorithms that perform the same operations on a large number of small data types (bytes, words, and doublewords). For example, most audio data is represented in 16-bit (word) quantities. The MMX instructions can operate on 4 words simultaneously with one instruction. Video and graphics information is commonly represented as palletized 8-bit (byte) quantities. In Figure 9-4, one MMX instruction operates on 8 bytes simultaneously.

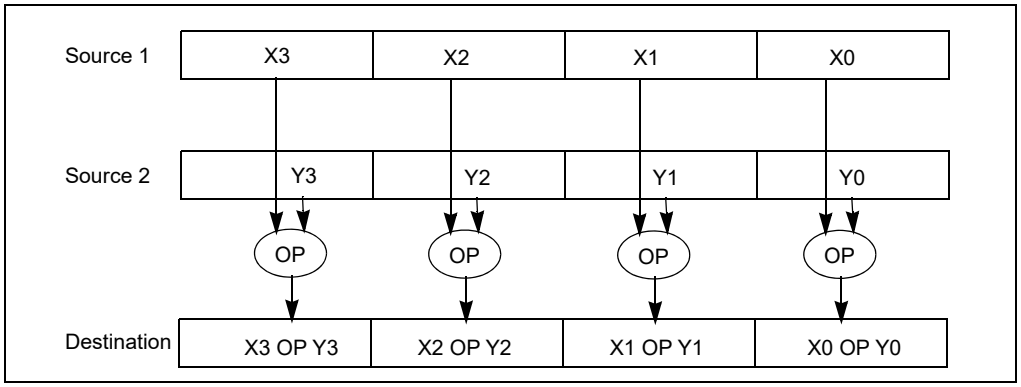


Figure 9-4. SIMD Execution Model

9.3 SATURATION AND WRAPAROUND MODES

When performing integer arithmetic, an operation may result in an out-of-range condition, where the true result cannot be represented in the destination format. For example, when performing arithmetic on signed word integers, positive overflow can occur when the true signed result is larger than 16 bits.

The MMX technology provides three ways of handling out-of-range conditions:

- **Wraparound arithmetic —** With wraparound arithmetic, a true out-of-range result is truncated (that is, the carry or overflow bit is ignored and only the least significant bits of the result are returned to the destination). Wraparound arithmetic is suitable for applications that control the range of operands to prevent out-of-range results. If the range of operands is not controlled, however, wraparound arithmetic can lead to large errors. For example, adding two large signed numbers can cause positive overflow and produce a negative result.
- **Signed saturation arithmetic —** With signed saturation arithmetic, out-of-range results are limited to the representable range of signed integers for the integer size being operated on (see Table 9-1). For example, if positive overflow occurs when operating on signed word integers, the result is "saturated" to 7FFFH, which is the largest positive integer that can be represented in 16 bits; if negative overflow occurs, the result is saturated to 8000H.
- **Unsigned saturation arithmetic —** With unsigned saturation arithmetic, out-of-range results are limited to the representable range of unsigned integers for the integer size. So, positive overflow when operating on unsigned byte integers results in FFH being returned and negative overflow results in 00H being returned.

Table 9-1. Data Range Limits for Saturation

Data Type	Lower Limit		Upper Limit	
	Hexadecimal	Decimal	Hexadecimal	Decimal
Signed Byte	80H	-128	7FH	127
Signed Word	8000H	-32,768	7FFFH	32,767
Unsigned Byte	00H	0	FFH	255
Unsigned Word	0000H	0	FFFFH	65,535

Saturation arithmetic provides an answer for many overflow situations. For example, in color calculations, saturation causes a color to remain pure black or pure white without allowing inversion. It also prevents wraparound artifacts from entering into computations when range checking of source operands it not used.

MMX instructions do not indicate overflow or underflow occurrence by generating exceptions or setting flags in the EFLAGS register.

9.4 MMX INSTRUCTIONS

The MMX instruction set consists of 47 instructions, grouped into the following categories:

- Data transfer
- Arithmetic
- Comparison
- Conversion
- Unpacking
- Logical
- Shift
- Empty MMX state instruction (EMMS)

Table 9-2 gives a summary of the instructions in the MMX instruction set. The following sections give a brief overview of the instructions within each group.

NOTES

The MMX instructions described in this chapter are those instructions that are available in an IA-32 processor when CPUID.01H:EDX.MMX[bit 23] = 1.

Section 10.4.4, "Intel® SSE 64-Bit SIMD Integer Instructions," and Section 11.4.2, "Intel® SSE2 64-Bit and 128-Bit SIMD Integer Instructions," list additional instructions included with the Intel SSE/SSE2 extensions that operate on MMX registers but are not considered part of the MMX instruction set.

Table 9-2. MMX Instruction Set Summary

Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	Multiplication	PMULL, PMULH		
	Multiply and Add	PMADD		
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB
Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		
		Packed		**Full Quadword**
Logical	And			PAND
	And Not			PANDN
	Or			POR
	Exclusive OR			PXOR
Shift	Shift Left Logical	PSLLW, PSLLD		PSLLQ
	Shift Right Logical	PSRLW, PSRLD		PSRLQ
	Shift Right Arithmetic	PSRAW, PSRAD		
		Doubleword Transfers		**Quadword Transfers**
Data Transfer	Register to Register	MOVD		MOVQ
	Load from Memory	MOVD		MOVQ
	Store to Memory	MOVD		MOVQ
Empty MMX State		**EMMS**		

9.4.1 Data Transfer Instructions

The MOVD (Move 32 Bits) instruction transfers 32 bits of packed data from memory to an MMX register and vice versa; or from a general-purpose register to an MMX register and vice versa.

The MOVQ (Move 64 Bits) instruction transfers 64 bits of packed data from memory to an MMX register and vice versa; or transfers data between MMX registers.

9.4.2 Arithmetic Instructions

The arithmetic instructions perform addition, subtraction, multiplication, and multiply/add operations on packed data types.

The PADDB/PADDW/PADDD (add packed integers) instructions and the PSUBB/PSUBW/ PSUBD (subtract packed integers) instructions add or subtract the corresponding signed or unsigned data elements of the source and desti-

nation operands in wraparound mode. These instructions operate on packed byte, word, and doubleword data types.

The PADDSB/PADDSW (add packed signed integers with signed saturation) instructions and the PSUBSB/PSUBSW (subtract packed signed integers with signed saturation) instructions add or subtract the corresponding signed data elements of the source and destination operands and saturate the result to the limits of the signed data-type range. These instructions operate on packed byte and word data types.

The PADDUSB/PADDUSW (add packed unsigned integers with unsigned saturation) instructions and the PSUBUSB/PSUBUSW (subtract packed unsigned integers with unsigned saturation) instructions add or subtract the corresponding unsigned data elements of the source and destination operands and saturate the result to the limits of the unsigned data-type range. These instructions operate on packed byte and word data types.

The PMULHW (multiply packed signed integers and store high result) and PMULLW (multiply packed signed integers and store low result) instructions perform a signed multiply of the corresponding words of the source and destination operands and write the high-order or low-order 16 bits of each of the results, respectively, to the destination operand.

The PMADDWD (multiply and add packed integers) instruction computes the products of the corresponding signed words of the source and destination operands. The four intermediate 32-bit doubleword products are summed in pairs (high-order pair and low-order pair) to produce two 32-bit doubleword results.

9.4.3 Comparison Instructions

The PCMPEQB/PCMPEQW/PCMPEQD (compare packed data for equal) instructions and the PCMPGTB/PCMPGTW/PCMPGTD (compare packed signed integers for greater than) instructions compare the corresponding signed data elements (bytes, words, or doublewords) in the source and destination operands for equal to or greater than, respectively.

These instructions generate a mask of ones or zeros which are written to the destination operand. Logical operations can use the mask to select packed elements. This can be used to implement a packed conditional move operation without a branch or a set of branch instructions. No flags in the EFLAGS register are affected.

9.4.4 Conversion Instructions

The PACKSSWB (pack words into bytes with signed saturation) and PACKSSDW (pack doublewords into words with signed saturation) instructions convert signed words into signed bytes and signed doublewords into signed words, respectively, using signed saturation.

PACKUSWB (pack words into bytes with unsigned saturation) converts signed words into unsigned bytes, using unsigned saturation.

9.4.5 Unpack Instructions

The PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ (unpack high-order data elements) instructions and the PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ (unpack low-order data elements) instructions unpack bytes, words, or doublewords from the high- or low-order data elements of the source and destination operands and interleave them in the destination operand. By placing all 0s in the source operand, these instructions can be used to convert byte integers to word integers, word integers to doubleword integers, or doubleword integers to quadword integers.

9.4.6 Logical Instructions

PAND (bitwise logical AND), PANDN (bitwise logical AND NOT), POR (bitwise logical OR), and PXOR (bitwise logical exclusive OR) perform bitwise logical operations on the quadword source and destination operands.

9.4.7 Shift Instructions

The logical shift left, logical shift right and arithmetic shift right instructions shift each element by a specified number of bit positions.

The PSLLW/PSLLD/PSLLQ (shift packed data left logical) instructions and the PSRLW/PSRLD/PSRLQ (shift packed data right logical) instructions perform a logical left or right shift of the data elements and fill the empty high or low order bit positions with zeros. These instructions operate on packed words, doublewords, and quadwords.

The PSRAW/PSRAD (shift packed data right arithmetic) instructions perform an arithmetic right shift, copying the sign bit for each data element into empty bit positions on the upper end of each data element. This instruction operates on packed words and doublewords.

9.4.8 EMMS Instruction

The EMMS instruction empties the MMX state by setting the tags in x87 FPU tag word to 11B, indicating empty registers. This instruction must be executed at the end of an MMX routine before calling other routines that can execute floating-point instructions. See Section 9.6.3, "Using the EMMS Instruction," for more information on the use of this instruction.

9.5 COMPATIBILITY WITH X87 FPU ARCHITECTURE

The MMX state is aliased to the x87 FPU state. No new states or modes have been added to IA-32 architecture to support the MMX technology. The same floating-point instructions that save and restore the x87 FPU state also handle the MMX state (for example, during context switching).

MMX technology uses the same interface techniques between the x87 FPU and the operating system (primarily for task switching purposes). For more details, see Chapter 13, "Intel® MMX™ Technology System Programming," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

9.5.1 MMX Instructions and the x87 FPU Tag Word

After each MMX instruction, the entire x87 FPU tag word is set to valid (00B). The EMMS instruction (empty MMX state) sets the entire x87 FPU tag word to empty (11B).

Chapter 13, "Intel® MMX™ Technology System Programming," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, provides additional information about the effects of x87 FPU and MMX instructions on the x87 FPU tag word. For a description of the tag word, see Section 8.1.7, "x87 FPU Tag Word."

9.6 WRITING APPLICATIONS WITH MMX CODE

The following sections give guidelines for writing application code that uses MMX technology.

9.6.1 Checking for MMX Technology Support

Before an application attempts to use the MMX technology, it should check that it is present on the processor. Check by following these steps:

1. Check that the processor supports the CPUID instruction by attempting to execute the CPUID instruction. If the processor does not support the CPUID instruction, this will generate an invalid-opcode exception (#UD).
2. Check that the processor supports the MMX technology (if CPUID.01H:EDX.MMX[bit 23] = 1).
3. Check that emulation of the x87 FPU is disabled (if CR0.EM[bit 2] = 0).

If the processor attempts to execute an unsupported MMX instruction or attempts to execute an MMX instruction with CR0.EM[bit 2] set, this generates an invalid-opcode exception (#UD).

Example 9-1 illustrates how to use the CPUID instruction to detect the MMX technology. This example does not represent the entire CPUID sequence, but shows the portion used for detection of MMX technology.

Example 9-1. Partial Routine for Detecting MMX Technology with the CPUID Instruction

```
...                             ; identify existence of CPUID instruction
...                             ; identify Intel processor
mov     EAX, 1                  ; request for feature flags
CPUID                           ; 0FH, 0A2H CPUID instruction
test    EDX, 00800000H          ; Is IA MMX technology bit (Bit 23 of EDX) set?
jnz                         ; MMX_Technology_Found
```

9.6.2 Transitions Between x87 FPU and MMX Code

Applications can contain both x87 FPU floating-point and MMX instructions. However, because the MMX registers are aliased to the x87 FPU register stack, care must be taken when making transitions between x87 FPU instructions and MMX instructions to prevent incoherent or unexpected results.

When an MMX instruction (other than the EMMS instruction) is executed, the processor changes the x87 FPU state as follows:

- The TOS (top of stack) value of the x87 FPU status word is set to 0.
- The entire x87 FPU tag word is set to the valid state (00B in all tag fields).
- When an MMX instruction writes to an MMX register, it writes ones (11B) to the exponent part of the corresponding floating-point register (bits 64 through 79).

The net result of these actions is that any x87 FPU state prior to the execution of the MMX instruction is essentially lost.

When an x87 FPU instruction is executed, the processor assumes that the current state of the x87 FPU register stack and control registers is valid and executes the instruction without any preparatory modifications to the x87 FPU state.

If the application contains both x87 FPU floating-point and MMX instructions, the following guidelines are recommended:

- When transitioning between x87 FPU and MMX code, save the state of any x87 FPU data or control registers that need to be preserved for future use. The FSAVE and FXSAVE instructions save the entire x87 FPU state.
- When transitioning between MMX and x87 FPU code, do the following:
  - Save any data in the MMX registers that needs to be preserved for future use. FSAVE and FXSAVE also save the state of MMX registers.
  - Execute the EMMS instruction to clear the MMX state from the x87 data and control registers.

The following sections describe the use of the EMMS instruction and give additional guidelines for mixing x87 FPU and MMX code.

9.6.3 Using the EMMS Instruction

As described in Section 9.6.2, "Transitions Between x87 FPU and MMX Code," when an MMX instruction executes, the x87 FPU tag word is marked valid (00B). In this state, the execution of subsequent x87 FPU instructions may produce unexpected x87 FPU floating-point exceptions and/or incorrect results because the x87 FPU register stack appears to contain valid data. The EMMS instruction is provided to prevent this problem by marking the x87 FPU tag word as empty.

The EMMS instruction should be used in each of the following cases:

- When an application using the x87 FPU instructions calls an MMX technology library/DLL (use the EMMS instruction at the end of the MMX code).

- When an application using MMX instructions calls a x87 FPU floating-point library/DLL (use the EMMS instruction before calling the x87 FPU code).
- When a switch is made between MMX code in a task or thread and other tasks or threads in cooperative operating systems, unless it is certain that more MMX instructions will be executed before any x87 FPU code.

EMMS is not required when mixing MMX technology instructions with Intel SSE/SSE2/SSE3 instructions; see Section 11.6.7, "Interaction of Intel® SSE and SSE2 Instructions with x87 FPU and MMX Instructions."

9.6.4 Mixing MMX and x87 FPU Instructions

An application can contain both x87 FPU floating-point and MMX instructions. However, frequent transitions between MMX and x87 FPU instructions are not recommended, because they can degrade performance in some processor implementations. When mixing MMX code with x87 FPU code, follow these guidelines:

- Keep the code in separate modules, procedures, or routines.
- Do not rely on register contents across transitions between x87 FPU and MMX code modules.
- When transitioning between MMX code and x87 FPU code, save the MMX register state (if it will be needed in the future) and execute an EMMS instruction to empty the MMX state.
- When transitioning between x87 FPU code and MMX code, save the x87 FPU state if it will be needed in the future.

9.6.5 Interfacing with MMX Code

MMX technology enables direct access to all the MMX registers. This means that all existing interface conventions that apply to the use of the processor's general-purpose registers (EAX, EBX, etc.) also apply to the use of MMX registers.

An efficient interface to MMX routines might pass parameters and return values through the MMX registers or through a combination of memory locations (via the stack) and MMX registers. Do not use the EMMS instruction or mix MMX and x87 FPU code when using to the MMX registers to pass parameters.

If a high-level language that does not support the MMX data types directly is used, the MMX data types can be defined as a 64-bit structure containing packed data types.

When implementing MMX instructions in high-level languages, other approaches can be taken, such as:

- Passing parameters to an MMX routine by passing a pointer to a structure via the stack.
- Returning a value from a function by returning a pointer to a structure.

9.6.6 Using MMX Code in a Multitasking Operating System Environment

An application needs to identify the nature of the multitasking operating system on which it runs. Each task retains its own state which must be saved when a task switch occurs. The processor state (context) consists of the general-purpose registers and the floating-point and MMX registers.

Operating systems can be classified into two types:

- Cooperative multitasking operating system.
- Preemptive multitasking operating system.

Cooperative multitasking operating systems do not save the FPU or MMX state when performing a context switch. Therefore, the application needs to save the relevant state before relinquishing direct or indirect control to the operating system.

Preemptive multitasking operating systems are responsible for saving and restoring the FPU and MMX state when performing a context switch. Therefore, the application does not have to save or restore the FPU and MMX state.

9.6.7 Exception Handling in MMX Code

MMX instructions generate the same type of memory-access exceptions as other IA-32 instructions (page fault, segment not present, and limit violations). Existing exception handlers do not have to be modified to handle these types of exceptions for MMX code.

Unless there is a pending floating-point exception, MMX instructions do not generate numeric exceptions. Therefore, there is no need to modify existing exception handlers or add new ones to handle numeric exceptions.

If a floating-point exception is pending, the subsequent MMX instruction generates a numeric error exception (interrupt 16 and/or assertion of the FERR# pin). The MMX instruction resumes execution upon return from the exception handler.

9.6.8 Register Mapping

MMX registers and their tags are mapped to physical locations of the floating-point registers and their tags. Register aliasing and mapping is described in more detail in Chapter 13, "Intel® MMX™ Technology System Programming," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

9.6.9 Effect of Instruction Prefixes on MMX Instructions

Table 9-3 describes the effect of instruction prefixes on MMX instructions. Unpredictable behavior can range from being treated as a reserved operation on one generation of IA-32 processors to generating an invalid opcode exception on another generation of processors.

Table 9-3. Effect of Prefixes on MMX Instructions

Prefix Type	Effect on MMX Instructions
Address Size Prefix (67H)	Affects instructions with a memory operand.
	Reserved for instructions without a memory operand and may result in unpredictable behavior.
Operand Size (66H)	Reserved and may result in unpredictable behavior.
Segment Override (2EH, 36H, 3EH, 26H, 64H, 65H)	Affects instructions with a memory operand.
	Reserved for instructions without a memory operand and may result in unpredictable behavior.
Repeat Prefix (F3H)	Reserved and may result in unpredictable behavior.
Repeat NE Prefix(F2H)	Reserved and may result in unpredictable behavior.
Lock Prefix (F0H)	Reserved; generates invalid opcode exception (#UD).

See "Instruction Prefixes" in Chapter 2, "Instruction Format," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a description of the instruction prefixes.

3. Updates to Chapter 11, Volume 1

Change bars and violet text show changes to Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

Changes to this chapter:

- In Section 11.6.14, "Effect of Instruction Prefixes on Intel® SSE and SSE2 Instructions," Table 11-3 was updated to remove the "Branch Hint Prefixes" entry as these do not apply to SSE instructions.

CHAPTER 11
PROGRAMMING WITH INTEL® STREAMING SIMD EXTENSIONS 2 (INTEL® SSE2)

The streaming SIMD extensions 2 (SSE2) were introduced into the IA-32 architecture in the Pentium 4 and Intel Xeon processors. These extensions enhance the performance of IA-32 processors for advanced 3-D graphics, video decoding/encoding, speech recognition, E-commerce, Internet, scientific, and engineering applications.

This chapter describes the SSE2 extensions and provides information to assist in writing application programs that use these and the SSE extensions.

11.1 OVERVIEW OF INTEL® SSE2

Intel SSE2 uses the single instruction multiple data (SIMD) execution model that is used with MMX technology and Intel SSE. They extend this model with support for packed double precision floating-point values and for 128-bit packed integers.

If CPUID.01H:EDX.SSE2[bit 26] = 1, Intel SSE2 is present.

Intel SSE2 adds the following features to the IA-32 architecture, while maintaining backward compatibility with all existing IA-32 processors, applications, and operating systems.

- Six data types:
  - 128-bit packed double precision floating-point (two IEEE Standard 754 double precision floating-point values packed into a double quadword).
  - 128-bit packed byte integers.
  - 128-bit packed word integers.
  - 128-bit packed doubleword integers.
  - 128-bit packed quadword integers.
- Instructions to support the additional data types and extend existing SIMD integer operations:
  - Packed and scalar double precision floating-point instructions.
  - Additional 64-bit and 128-bit SIMD integer instructions.
  - 128-bit versions of SIMD integer instructions introduced with the MMX technology and Intel SSE.
  - Additional cacheability-control and instruction-ordering instructions.
- Modifications to existing IA-32 instructions to support Intel SSE2 features:
  - Extensions and modifications to the CPUID instruction.
  - Modifications to the RDPMC instruction.

These new features extend the IA-32 architecture's SIMD programming model in three important ways:

- They provide the ability to perform SIMD operations on pairs of packed double precision floating-point values. This permits higher precision computations to be carried out in XMM registers, which enhances processor performance in scientific and engineering applications and in applications that use advanced 3-D geometry techniques (such as ray tracing). Additional flexibility is provided with instructions that operate on single (scalar) double precision floating-point values located in the low quadword of an XMM register.
- They provide the ability to operate on 128-bit packed integers (bytes, words, doublewords, and quadwords) in XMM registers. This provides greater flexibility and greater throughput when performing SIMD operations on packed integers. The capability is particularly useful for applications such as RSA authentication and RC5 encryption. Using the full set of SIMD registers, data types, and instructions provided with the MMX technology and Intel SSE/SSE2, programmers can develop algorithms that finely mix packed single- and double precision floating-point data and 64- and 128-bit packed integer data.
- Intel SSE2 enhances the support introduced with Intel SSE for controlling the cacheability of SIMD data. Intel SSE2 cache control instructions provide the ability to stream data in and out of the XMM registers without polluting the caches and the ability to prefetch data before it is actually used.

Intel SSE2 is fully compatible with all software written for IA-32 processors. All existing software continues to run correctly, without modification, on processors that incorporate Intel SSE2, as well as in the presence of applications that incorporate these extensions. Enhancements to the CPUID instruction permit detection of Intel SSE2. Also, because Intel SSE2 uses the same registers as Intel SSE, no new operating-system support is required for saving and restoring program state during a context switch beyond that provided for Intel SSE.

Intel SSE2 is accessible from all IA-32 execution modes: protected mode, real address mode, and virtual 8086 mode.

The following sections in this chapter describe the programming environment for Intel SSE2, including: the 128-bit XMM floating-point register set, data types, and Intel SSE2 instructions. The chapter also describes exceptions that can be generated with the Intel SSE and SSE2 instructions and gives guidelines for writing applications with Intel SSE and SSE2.

For additional information about Intel SSE2, see:

- The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D, provides a detailed description of individual Intel SSE2 instructions.
- Chapter 14, "System Programming for Instruction Set Extensions and Processor Extended States," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, gives guidelines for integrating Intel SSE and SSE2 into an operating-system environment.

11.2 INTEL® SSE2 PROGRAMMING ENVIRONMENT

Figure 11-1 shows the programming environment for Intel SSE2. No new registers or other instruction execution state are defined with Intel SSE2. Intel SSE2 instructions use XMM registers, MMX registers, and/or IA-32 general-purpose registers, as follows:

- **XMM registers —** These eight registers (see Figure 10-2) are used to operate on packed or scalar double precision floating-point data. Scalar operations are operations performed on individual (unpacked) double precision floating-point values stored in the low quadword of an XMM register. XMM registers are also used to perform operations on 128-bit packed integer data. They are referenced by the names XMM0 through XMM7.

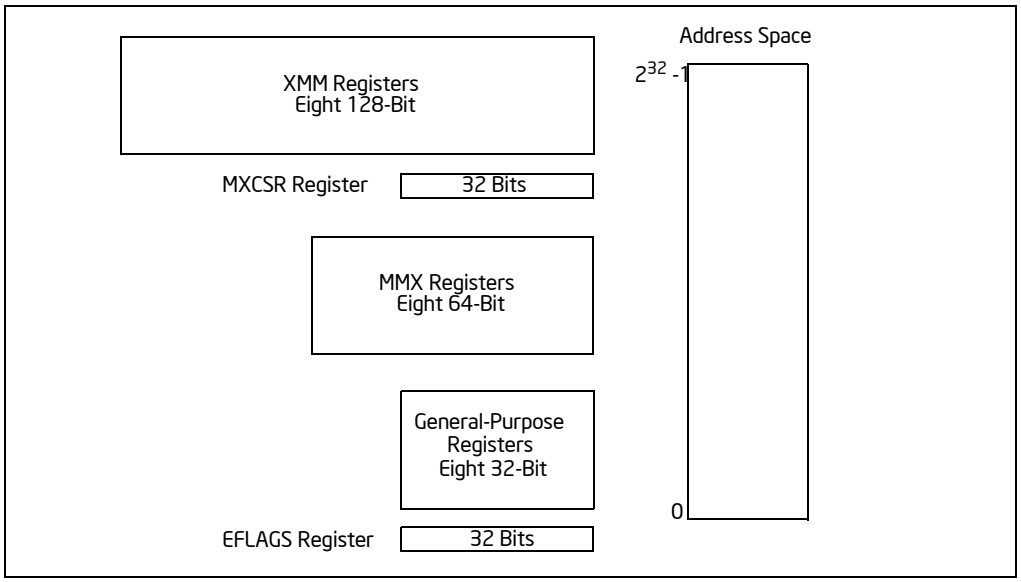


Figure 11-1. Intel® Steaming SIMD Extensions 2 Execution Environment

- **MXCSR register —** This 32-bit register (see Figure 10-3) provides status and control bits used in floating-point operations. The denormals-are-zeros and flush-to-zero flags in this register provide a higher performance alternative for the handling of denormal source operands and denormal (underflow) results. For more

information on the functions of these flags see Section 10.2.3.4, "Denormals-Are-Zeros," and Section 10.2.3.3, "Flush-To-Zero."

- **MMX registers —** These eight registers (see Figure 9-2) are used to perform operations on 64-bit packed integer data. They are also used to hold operands for some operations performed between MMX and XMM registers. MMX registers are referenced by the names MM0 through MM7.
- **General-purpose registers —** The eight general-purpose registers (see Figure 3-5) are used along with the existing IA-32 addressing modes to address operands in memory. MMX and XMM registers cannot be used to address memory. The general-purpose registers are also used to hold operands for some SSE2 instructions. These registers are referenced by the names EAX, EBX, ECX, EDX, EBP, ESI, EDI, and ESP.
- **EFLAGS register —** This 32-bit register (see Figure 3-8) is used to record the results of some compare operations.

11.2.1 Intel® SSE2 in 64-Bit Mode and Compatibility Mode

In compatibility mode, Intel SSE2 functions like it does in protected mode. In 64-bit mode, eight additional XMM registers are accessible. Registers XMM8-XMM15 are accessed by using REX prefixes.

Memory operands are specified using the ModR/M, SIB encoding described in Section 3.7.5.

Some Intel SSE2 instructions may be used to operate on general-purpose registers. Use the REX.W prefix to access 64-bit general-purpose registers. Note that if a REX prefix is used when it has no meaning, the prefix is ignored.

11.2.2 Compatibility of Intel® SSE2 with Intel® SSE, MMX Technology, and x87 FPU Programming Environment

Intel SSE2 does not introduce any new state to the IA-32 execution environment beyond that of Intel SSE. Intel SSE2 represents an enhancement of Intel SSE; they are fully compatible and share the same state information. Intel SSE and SSE2 instructions can be executed together in the same instruction stream without the need to save state when switching between instruction sets.

XMM registers are independent of the x87 FPU and MMX registers; so Intel SSE and SSE2 operations performed on XMM registers can be performed in parallel with x87 FPU or MMX technology operations; see Section 11.6.7, "Interaction of Intel® SSE and SSE2 Instructions with x87 FPU and MMX Instructions."

The FXSAVE and FXRSTOR instructions save and restore the SSE and SSE2 states along with the x87 FPU and MMX states.

11.2.3 Denormals-Are-Zeros Flag

The denormals-are-zeros flag (bit 6 in the MXCSR register) was introduced into the IA-32 architecture with Intel SSE2. See Section 10.2.3.4, "Denormals-Are-Zeros," for a description of this flag.

11.3 INTEL® SSE2 DATA TYPES

Intel SSE2 introduced one 128-bit packed floating-point data type and four 128-bit SIMD integer data types to the IA-32 architecture (see Figure 11-2).

- **Packed double precision floating-point —** This 128-bit data type consists of two IEEE 64-bit double precision floating-point values packed into a double quadword. See Figure 4-3 for the layout of a 64-bit double precision floating-point value; refer to Section 4.2.2, "Floating-Point Data Types," for a detailed description of double precision floating-point values.
- **128-bit packed integers —** The four 128-bit packed integer data types can contain 16 byte integers, 8 word integers, 4 doubleword integers, or 2 quadword integers. Refer to Section 4.6.2, "128-Bit Packed SIMD Data Types," for a detailed description of the 128-bit packed integers.

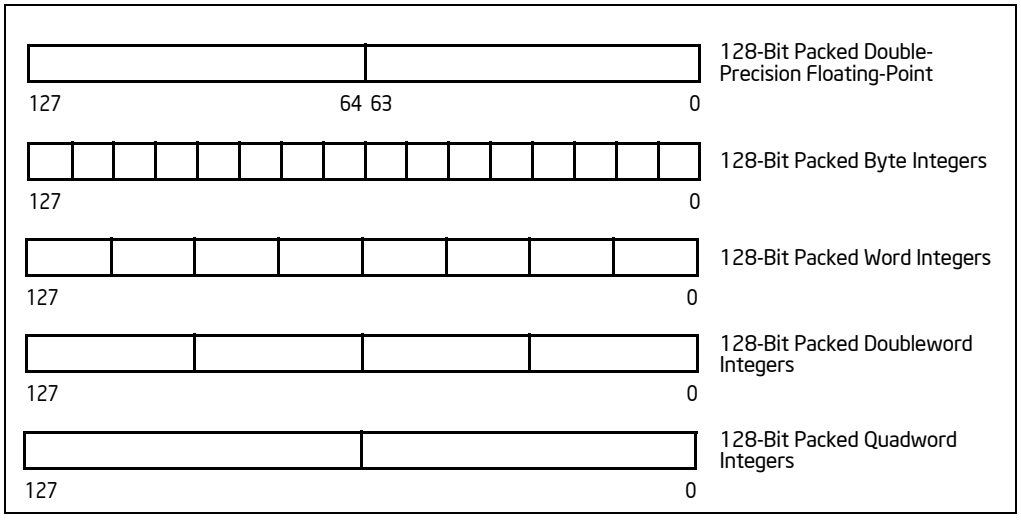


Figure 11-2. Data Types Introduced with Intel® SSE2

All of these data types are operated on in XMM registers or memory. Instructions are provided to convert between these 128-bit data types and the 64-bit and 32-bit data types.

The address of a 128-bit packed memory operand must be aligned on a 16-byte boundary, except in the following cases:

- A MOVUPD instruction that supports unaligned accesses.
- Scalar instructions that use an 8-byte memory operand that is not subject to alignment requirements.

Figure 4-2 shows the byte order of 128-bit (double quadword) and 64-bit (quadword) data types in memory.

11.4 INTEL® SSE2 INSTRUCTIONS

The Intel SSE2 instructions are divided into four functional groups:

- Packed and scalar double precision floating-point instructions.
- 64-bit and 128-bit SIMD integer instructions.
- 128-bit extensions of SIMD integer instructions introduced with the MMX technology and Intel SSE.
- Cacheability-control and instruction-ordering instructions.

The following sections provide more information about each group.

11.4.1 Packed and Scalar Double Precision Floating-Point Instructions

The packed and scalar double precision floating-point instructions are divided into the following sub-groups:

- Data movement instructions.
- Arithmetic instructions.
- Comparison instructions.
- Conversion instructions.
- Logical instructions.
- Shuffle instructions.

The packed double precision floating-point instructions perform SIMD operations similarly to the packed single precision floating-point instructions (see Figure 11-3). Each source operand contains two double precision floating-

point values, and the destination operand contains the results of the operation (OP) performed in parallel on the corresponding values (X0 and Y0, and X1 and Y1) in each operand.

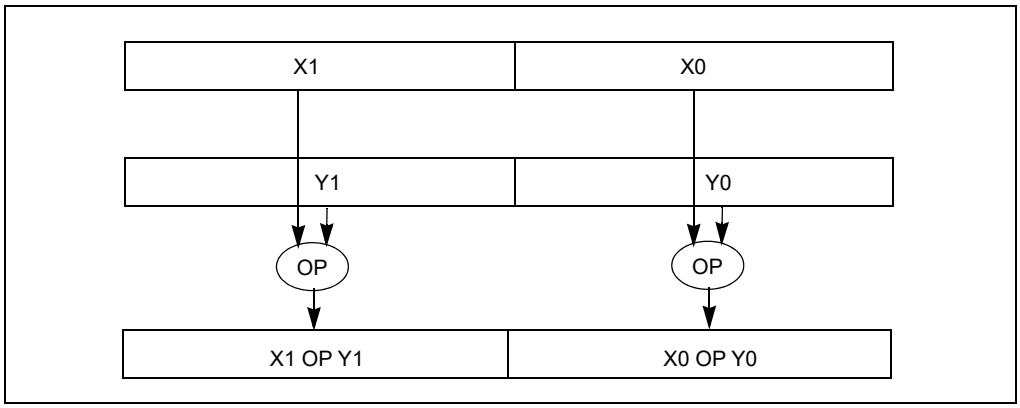


Figure 11-3. Packed Double Precision Floating-Point Operations

The scalar double precision floating-point instructions operate on the low (least significant) quadwords of two source operands (X0 and Y0), as shown in Figure 11-4. The high quadword (X1) of the first source operand is passed through to the destination. The scalar operations are similar to the floating-point operations performed in x87 FPU data registers with the precision control field in the x87 FPU control word set for double precision (53-bit significand), except that x87 stack operations use a 15-bit exponent range for the result while Intel SSE2 operations use an 11-bit exponent range.

See Section 11.6.8, "Compatibility of SIMD and x87 FPU Floating-Point Data Types," for more information about obtaining compatible results when performing both scalar double precision floating-point operations in XMM registers and in x87 FPU data registers.

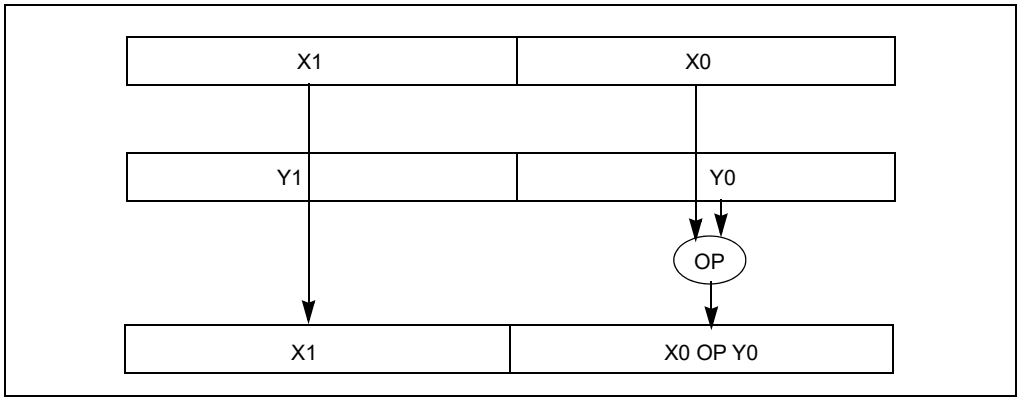


Figure 11-4. Scalar Double Precision Floating-Point Operations

11.4.1.1 Data Movement Instructions

Data movement instructions move double precision floating-point data between XMM registers and between XMM registers and memory.

The MOVAPD (move aligned packed double precision floating-point) instruction transfers a 128-bit packed double precision floating-point operand from memory to an XMM register or vice versa, or between XMM registers. The memory address must be aligned to a 16-byte boundary; if not, a general-protection exception (GP#) is generated.

The MOVUPD (move unaligned packed double precision floating-point) instruction transfers a 128-bit packed double precision floating-point operand from memory to an XMM register or vice versa, or between XMM registers. Alignment of the memory address is not required.

The MOVSD (move scalar double precision floating-point) instruction transfers a 64-bit double precision floating-point operand from memory to the low quadword of an XMM register or vice versa, or between XMM registers. Alignment of the memory address is not required, unless alignment checking is enabled.

The MOVHPD (move high packed double precision floating-point) instruction transfers a 64-bit double precision floating-point operand from memory to the high quadword of an XMM register or vice versa. The low quadword of the register is left unchanged. Alignment of the memory address is not required, unless alignment checking is enabled.

The MOVLPD (move low packed double precision floating-point) instruction transfers a 64-bit double precision floating-point operand from memory to the low quadword of an XMM register or vice versa. The high quadword of the register is left unchanged. Alignment of the memory address is not required, unless alignment checking is enabled.

The MOVMSKPD (move packed double precision floating-point mask) instruction extracts the sign bit of each of the two packed double precision floating-point numbers in an XMM register and saves them in a general-purpose register. This 2-bit value can then be used as a condition to perform branching.

11.4.1.2 Intel® SSE2 Arithmetic Instructions

Intel SSE2 arithmetic instructions perform addition, subtraction, multiply, divide, square root, and maximum/minimum operations on packed and scalar double precision floating-point values.

The ADDPD (add packed double precision floating-point values) and SUBPD (subtract packed double precision floating-point values) instructions add and subtract, respectively, two packed double precision floating-point operands.

The ADDSD (add scalar double precision floating-point values) and SUBSD (subtract scalar double precision floating-point values) instructions add and subtract, respectively, the low double precision floating-point values of two operands and stores the result in the low quadword of the destination operand.

The MULPD (multiply packed double precision floating-point values) instruction multiplies two packed double precision floating-point operands.

The MULSD (multiply scalar double precision floating-point values) instruction multiplies the low double precision floating-point values of two operands and stores the result in the low quadword of the destination operand.

The DIVPD (divide packed double precision floating-point values) instruction divides two packed double precision floating-point operands.

The DIVSD (divide scalar double precision floating-point values) instruction divides the low double precision floating-point values of two operands and stores the result in the low quadword of the destination operand.

The SQRTPD (compute square roots of packed double precision floating-point values) instruction computes the square roots of the values in a packed double precision floating-point operand.

The SQRTSD (compute square root of scalar double precision floating-point values) instruction computes the square root of the low double precision floating-point value in the source operand and stores the result in the low quadword of the destination operand.

The MAXPD (return maximum of packed double precision floating-point values) instruction compares the corresponding values in two packed double precision floating-point operands and returns the numerically greater value from each comparison to the destination operand.

The MAXSD (return maximum of scalar double precision floating-point values) instruction compares the low double precision floating-point values from two packed double precision floating-point operands and returns the numerically higher value from the comparison to the low quadword of the destination operand.

The MINPD (return minimum of packed double precision floating-point values) instruction compares the corresponding values from two packed double precision floating-point operands and returns the numerically lesser value from each comparison to the destination operand.

The MINSD (return minimum of scalar double precision floating-point values) instruction compares the low values from two packed double precision floating-point operands and returns the numerically lesser value from the comparison to the low quadword of the destination operand.

11.4.1.3 Intel® SSE2 Logical Instructions

intel SSE2 logical instructions perform AND, AND NOT, OR, and XOR operations on packed double precision floating-point values.

The ANDPD (bitwise logical AND of packed double precision floating-point values) instruction returns the logical AND of two packed double precision floating-point operands.

The ANDNPD (bitwise logical AND NOT of packed double precision floating-point values) instruction returns the logical AND NOT of two packed double precision floating-point operands.

The ORPD (bitwise logical OR of packed double precision floating-point values) instruction returns the logical OR of two packed double precision floating-point operands.

The XORPD (bitwise logical XOR of packed double precision floating-point values) instruction returns the logical XOR of two packed double precision floating-point operands.

11.4.1.4 Intel® SSE2 Comparison Instructions

Intel SSE2 compare instructions compare packed and scalar double precision floating-point values and return the results of the comparison either to the destination operand or to the EFLAGS register.

The CMPPD (compare packed double precision floating-point values) instruction compares the corresponding values from two packed double precision floating-point operands, using an immediate operand as a predicate, and returns a 64-bit mask result of all 1s or all 0s for each comparison to the destination operand. The value of the immediate operand allows the selection of any of eight compare conditions: equal, less than, less than equal, unordered, not equal, not less than, not less than or equal, or ordered.

The CMPSD (compare scalar double precision floating-point values) instruction compares the low values from two packed double precision floating-point operands, using an immediate operand as a predicate, and returns a 64-bit mask result of all 1s or all 0s for the comparison to the low quadword of the destination operand. The immediate operand selects the compare condition as with the CMPPD instruction.

The COMISD (compare scalar double precision floating-point values and set EFLAGS) and UCOMISD (unordered compare scalar double precision floating-point values and set EFLAGS) instructions compare the low values of two packed double precision floating-point operands and set the ZF, PF, and CF flags in the EFLAGS register to show the result (greater than, less than, equal, or unordered). These two instructions differ as follows: the COMISD instruction signals a floating-point invalid-operation (#I) exception when a source operand is either a QNaN or an SNaN; the UCOMISD instruction only signals an invalid-operation exception when a source operand is an SNaN.

11.4.1.5 Intel® SSE2 Shuffle and Unpack Instructions

Intel SSE2 shuffle instructions shuffle the contents of two packed double precision floating-point values and store the results in the destination operand.

The SHUFPD (shuffle packed double precision floating-point values) instruction places either of the two packed double precision floating-point values from the destination operand in the low quadword of the destination operand, and places either of the two packed double precision floating-point values from source operand in the high quadword of the destination operand (see Figure 11-5). By using the same register for the source and destination operands, the SHUFPD instruction can swap two packed double precision floating-point values.

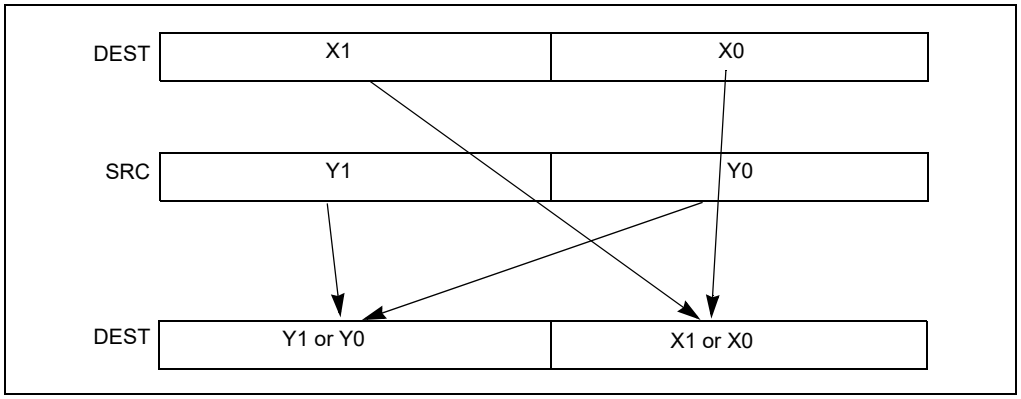


Figure 11-5. SHUFPD Instruction, Packed Shuffle Operation

The UNPCKHPD (unpack and interleave high packed double precision floating-point values) instruction performs an interleaved unpack of the high values from the source and destination operands and stores the result in the destination operand (see Figure 11-6).

The UNPCKLPD (unpack and interleave low packed double precision floating-point values) instruction performs an interleaved unpack of the low values from the source and destination operands and stores the result in the destination operand (see Figure 11-7).

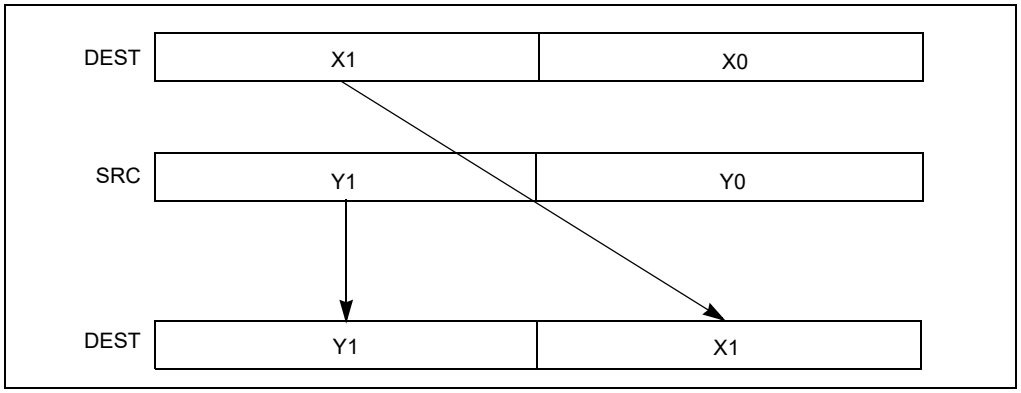


Figure 11-6. UNPCKHPD Instruction, High Unpack, and Interleave Operation

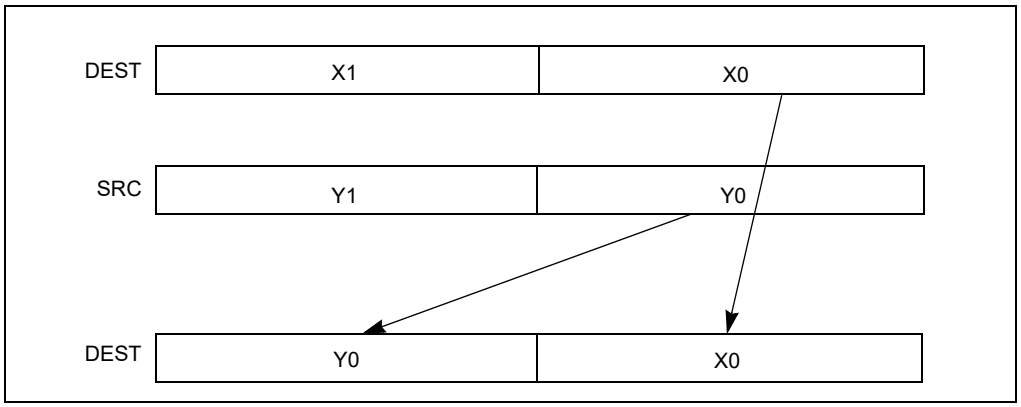


Figure 11-7. UNPCKLPD Instruction, Low Unpack, and Interleave Operation

11.4.1.6 Intel® SSE2 Conversion Instructions

Intel SSE2 conversion instructions (see Figure 11-8) support packed and scalar conversions between:

- Double precision and single precision floating-point formats.
- Double precision floating-point and doubleword integer formats.
- Single precision floating-point and doubleword integer formats.

Conversion between double precision and single precision floating-points values — The following instructions convert operands between double precision and single precision floating-point formats. The operands being operated on are contained in XMM registers or memory (at most, one operand can reside in memory; the destination is always an MMX register).

The CVTPS2PD (convert packed single precision floating-point values to packed double precision floating-point values) instruction converts two packed single precision floating-point values to two double precision floating-point values.

The CVTPD2PS (convert packed double precision floating-point values to packed single precision floating-point values) instruction converts two packed double-precision floating-point values to two single precision floating-point values. When a conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register.

The CVTSS2SD (convert scalar single precision floating-point value to scalar double precision floating-point value) instruction converts a single precision floating-point value to a double precision floating-point value.

The CVTSD2SS (convert scalar double precision floating-point value to scalar single precision floating-point value) instruction converts a double precision floating-point value to a single precision floating-point value. When the conversion is inexact, the result is rounded according to the rounding mode selected in the MXCSR register.

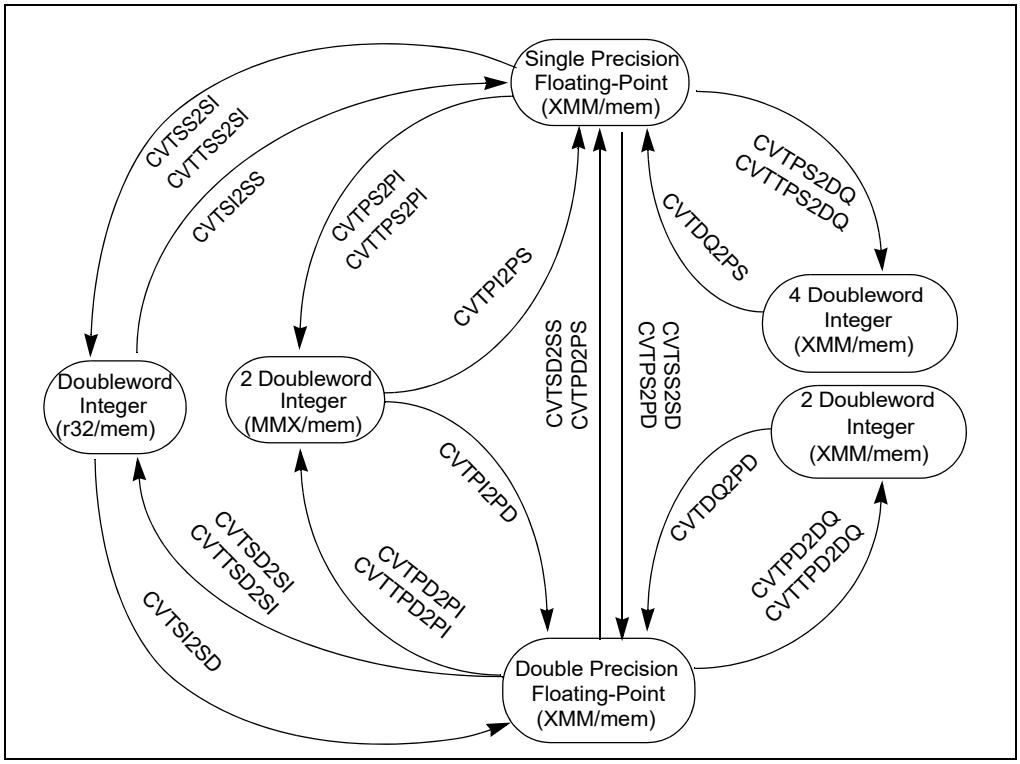


Figure 11-8. Intel® SSE and SSE2 Conversion Instructions

Conversion between double precision floating-point values and doubleword integers — The following instructions convert operands between double precision floating-point and doubleword integer formats. Operands

are housed in XMM registers, MMX registers, general registers or memory (at most one operand can reside in memory; the destination is always an XMM, MMX, or general register).

The CVTPD2PI (convert packed double precision floating-point values to packed doubleword integers) instruction converts two packed double precision floating-point numbers to two packed signed doubleword integers, with the result stored in an MMX register. When rounding to an integer value, the source value is rounded according to the rounding mode in the MXCSR register. The CVTTPD2PI (convert with truncation packed double precision floating-point values to packed doubleword integers) instruction is similar to the CVTPD2PI instruction except that truncation is used to round a source value to an integer value; see Section 4.8.4.2, "Truncation with Intel® SSE, SSE2, and AVX Conversion Instructions."

The CVTPI2PD (convert packed doubleword integers to packed double precision floating-point values) instruction converts two packed signed doubleword integers to two double precision floating-point values.

The CVTPD2DQ (convert packed double precision floating-point values to packed doubleword integers) instruction converts two packed double precision floating-point numbers to two packed signed doubleword integers, with the result stored in the low quadword of an XMM register. When rounding an integer value, the source value is rounded according to the rounding mode selected in the MXCSR register. The CVTTPD2DQ (convert with truncation packed double precision floating-point values to packed doubleword integers) instruction is similar to the CVTPD2DQ instruction except that truncation is used to round a source value to an integer value; see Section 4.8.4.2, "Truncation with Intel® SSE, SSE2, and AVX Conversion Instructions."

The CVTDQ2PD (convert packed doubleword integers to packed double precision floating-point values) instruction converts two packed signed doubleword integers located in the low-order doublewords of an XMM register to two double precision floating-point values.

The CVTSD2SI (convert scalar double precision floating-point value to doubleword integer) instruction converts a double precision floating-point value to a doubleword integer, and stores the result in a general-purpose register. When rounding an integer value, the source value is rounded according to the rounding mode selected in the MXCSR register. The CVTTSD2SI (convert with truncation scalar double precision floating-point value to doubleword integer) instruction is similar to the CVTSD2SI instruction except that truncation is used to round the source value to an integer value; see Section 4.8.4.2, "Truncation with Intel® SSE, SSE2, and AVX Conversion Instructions."

The CVTSI2SD (convert doubleword integer to scalar double precision floating-point value) instruction converts a signed doubleword integer in a general-purpose register to a double precision floating-point number, and stores the result in an XMM register.

Conversion between single precision floating-point and doubleword integer formats — These instructions convert between packed single precision floating-point and packed doubleword integer formats. Operands are housed in XMM registers, MMX registers, general registers, or memory (the latter for at most one source operand). The destination is always an XMM, MMX, or general register. These SSE2 instructions supplement conversion instructions (CVTPI2PS, CVTPS2PI, CVTTPS2PI, CVTSI2SS, CVTSS2SI, and CVTTSS2SI) introduced with Intel SSE extensions.

The CVTPS2DQ (convert packed single precision floating-point values to packed doubleword integers) instruction converts four packed single precision floating-point values to four packed signed doubleword integers, with the source and destination operands in XMM registers or memory (the latter for at most one source operand). When the conversion is inexact, the rounded value according to the rounding mode selected in the MXCSR register is returned. The CVTTPS2DQ (convert with truncation packed single precision floating-point values to packed doubleword integers) instruction is similar to the CVTPS2DQ instruction except that truncation is used to round a source value to an integer value; see Section 4.8.4.2, "Truncation with Intel® SSE, SSE2, and AVX Conversion Instructions."

The CVTDQ2PS (convert packed doubleword integers to packed single precision floating-point values) instruction converts four packed signed doubleword integers to four packed single precision floating-point numbers, with the source and destination operands in XMM registers or memory (the latter for at most one source operand). When the conversion is inexact, the rounded value according to the rounding mode selected in the MXCSR register is returned.

11.4.2 Intel® SSE2 64-Bit and 128-Bit SIMD Integer Instructions

Intel SSE2 adds several 128-bit packed integer instructions to the IA-32 architecture. Where appropriate, a 64-bit version of each of these instructions is also provided. The 128-bit versions of instructions operate on data in XMM registers; 64-bit versions operate on data in MMX registers. The instructions follow.

The MOVDQA (move aligned double quadword) instruction transfers a double quadword operand from memory to an XMM register or vice versa; or between XMM registers. The memory address must be aligned to a 16-byte boundary; otherwise, a general-protection exception (#GP) is generated.

The MOVDQU (move unaligned double quadword) instruction performs the same operations as the MOVDQA instruction, except that 16-byte alignment of a memory address is not required.

The PADDQ (packed quadword add) instruction adds two packed quadword integer operands or two single quadword integer operands, and stores the results in an XMM or MMX register, respectively. This instruction can operate on either unsigned or signed (two's complement notation) integer operands.

The PSUBQ (packed quadword subtract) instruction subtracts two packed quadword integer operands or two single quadword integer operands, and stores the results in an XMM or MMX register, respectively. Like the PADDQ instruction, PSUBQ can operate on either unsigned or signed (two's complement notation) integer operands.

The PMULUDQ (multiply packed unsigned doubleword integers) instruction performs an unsigned multiply of unsigned doubleword integers and returns a quadword result. Both 64-bit and 128-bit versions of this instruction are available. The 64-bit version operates on two doubleword integers stored in the low doubleword of each source operand, and the quadword result is returned to an MMX register. The 128-bit version performs a packed multiply of two pairs of doubleword integers. Here, the doublewords are packed in the first and third doublewords of the source operands, and the quadword results are stored in the low and high quadwords of an XMM register.

The PSHUFLW (shuffle packed low words) instruction shuffles the word integers packed into the low quadword of the source operand and stores the shuffled result in the low quadword of the destination operand. An 8-bit immediate operand specifies the shuffle order.

The PSHUFHW (shuffle packed high words) instruction shuffles the word integers packed into the high quadword of the source operand and stores the shuffled result in the high quadword of the destination operand. An 8-bit immediate operand specifies the shuffle order.

The PSHUFD (shuffle packed doubleword integers) instruction shuffles the doubleword integers packed into the source operand and stores the shuffled result in the destination operand. An 8-bit immediate operand specifies the shuffle order.

The PSLLDQ (shift double quadword left logical) instruction shifts the contents of the source operand to the left by the amount of bytes specified by an immediate operand. The empty low-order bytes are cleared (set to 0).

The PSRLDQ (shift double quadword right logical) instruction shifts the contents of the source operand to the right by the amount of bytes specified by an immediate operand. The empty high-order bytes are cleared (set to 0).

The PUNPCKHQDQ (Unpack high quadwords) instruction interleaves the high quadword of the source operand and the high quadword of the destination operand and writes them to the destination register.

The PUNPCKLQDQ (Unpack low quadwords) instruction interleaves the low quadwords of the source operand and the low quadwords of the destination operand and writes them to the destination register.

Two additional SSE instructions enable data movement from the MMX registers to the XMM registers.

The MOVQ2DQ (move quadword integer from MMX to XMM registers) instruction moves the quadword integer from an MMX source register to an XMM destination register.

The MOVDQ2Q (move quadword integer from XMM to MMX registers) instruction moves the low quadword integer from an XMM source register to an MMX destination register.

11.4.3 128-Bit SIMD Integer Instruction Extensions

All of 64-bit SIMD integer instructions introduced with MMX technology and Intel SSE (with the exception of the PSHUFW instruction) have been extended by Intel SSE2 to operate on 128-bit packed integer operands located in XMM registers. The 128-bit versions of these instructions follow the same SIMD conventions regarding packed

operands as the 64-bit versions. For example, where the 64-bit version of the PADDB instruction operates on 8 packed bytes, the 128-bit version operates on 16 packed bytes.

11.4.4 Cacheability Control and Memory Ordering Instructions

Intel SSE2 instructions that give programs more control over the caching, loading, and storing of data. are described below.

11.4.4.1 FLUSH Cache Line

The CLFLUSH (flush cache line) instruction writes and invalidates the cache line associated with a specified linear address. The invalidation is for all levels of the processor's cache hierarchy, and it is broadcast throughout the cache coherency domain.

NOTE

CLFLUSH was introduced with Intel SSE2. However, the instruction can be implemented in IA-32 processors that do not implement Intel SSE2. Detect CLFLUSH using the feature bit (if CPUID.01H:EDX.CLFSH[bit 19] = 1).

11.4.4.2 Cacheability Control Instructions

The following four instructions enable data from XMM and general-purpose registers to be stored to memory using a non-temporal hint. The non-temporal hint directs the processor to store data to memory without writing the data into the cache hierarchy. See Section 10.4.6.2, "Caching of Temporal vs. Non-Temporal Data," for more information about non-temporal stores and hints.

The MOVNTDQ (store double quadword using non-temporal hint) instruction stores packed integer data from an XMM register to memory, using a non-temporal hint.

The MOVNTPD (store packed double precision floating-point values using non-temporal hint) instruction stores packed double precision floating-point data from an XMM register to memory, using a non-temporal hint.

The MOVNTI (store doubleword using non-temporal hint) instruction stores integer data from a general-purpose register to memory, using a non-temporal hint.

The MASKMOVDQU (store selected bytes of double quadword) instruction stores selected byte integers from an XMM register to memory, using a byte mask to selectively write the individual bytes. The memory location does not need to be aligned on a natural boundary. This instruction also uses a non-temporal hint.

11.4.4.3 Memory Ordering Instructions

Intel SSE2 introduced two fence instructions (LFENCE and MFENCE) as companions to the SFENCE instruction introduced with Intel SSE.

The LFENCE instruction establishes a memory fence for loads. It guarantees ordering between two loads and prevents speculative loads from passing the load fence (that is, no speculative loads are allowed until all loads specified before the load fence have been carried out).

The MFENCE instruction establishes a memory fence for both loads and stores. The processor ensures that no load or store after MFENCE will become globally visible until all loads and stores before MFENCE are globally visible.[1] Note that the sequences LFENCE;SFENCE and SFENCE;LFENCE are not equivalent to MFENCE because neither ensures that older stores are globally observed prior to younger loads.

11.4.4.4 Pause

The PAUSE instruction is provided to improve the performance of "spin-wait loops" executed on a Pentium 4 or Intel Xeon processor. On a Pentium 4 processor, it also provides the added benefit of reducing processor power

1. A load is considered to become globally visible when the value to be loaded is determined.

consumption while executing a spin-wait loop. It is recommended that a PAUSE instruction always be included in the code sequence for a spin-wait loop.

11.4.5 Branch Hints

Intel SSE2 designates two instruction prefixes (2EH and 3EH) to provide branch hints to the processor (see "Instruction Prefixes" in Chapter 2 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). These prefixes can only be used with the Jcc instruction and only at the machine code level (that is, there are no mnemonics for the branch hints).

11.5 INTEL® SSE, SSE2, AND SSE3 EXCEPTIONS

Intel SSE, SSE2, and SSE3 instructions generate two general types of exceptions:

- Non-numeric exceptions.
- SIMD floating-point exceptions.[1]

Intel SSE, SSE2, and SSE3 instructions can generate the same type of memory-access and non-numeric exceptions as other IA-32 architecture instructions. Existing exception handlers can generally handle these exceptions without any code modification. See "Providing Non-Numeric Exception Handlers for Exceptions Generated by the SSE, SSE2, and SSE3 Instructions" in Chapter 14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for a list of the non-numeric exceptions that can be generated by the Intel SSE, SSE2, SSE3 instructions and for guidelines for handling these exceptions.

Intel SSE, SSE2, and SSE3 instructions do not generate numeric exceptions on packed integer operations; however, they can generate numeric (SIMD floating-point) exceptions on packed single precision and double precision floating-point operations. These SIMD floating-point exceptions are defined in the IEEE Standard 754 for Floating-Point Arithmetic and are the same exceptions that are generated for x87 FPU instructions. See Section 11.5.1, "SIMD Floating-Point Exceptions," for a description of these exceptions.

11.5.1 SIMD Floating-Point Exceptions

SIMD floating-point exceptions are those exceptions that can be generated by Intel SSE, SSE2, and SSE3 instructions that operate on packed or scalar floating-point operands.

Six classes of SIMD floating-point exceptions can be generated:

- Invalid operation (#I).
- Divide-by-zero (#Z).
- Denormal operand (#D).
- Numeric overflow (#O).
- Numeric underflow (#U).
- Inexact result (Precision) (#P).

All of these exceptions (except the denormal operand exception) are defined in IEEE Standard 754, and they are the same exceptions that are generated with the x87 floating-point instructions. Section 4.9, "Overview of Floating-Point Exceptions," gives a detailed description of these exceptions and of how and when they are generated. The following sections discuss the implementation of these exceptions in the Intel SSE/SSE2/SSE3 extensions.

All SIMD floating-point exceptions are precise and occur as soon as the instruction completes execution.

1. The FISTTP instruction in Intel SSE3 does not generate SIMD floating-point exceptions, but it can generate x87 FPU floating-point exceptions.

Each of the six exception conditions has a corresponding flag (IE, DE, ZE, OE, UE, and PE) and mask bit (IM, DM, ZM, OM, UM, and PM) in the MXCSR register (see Figure 10-3). The mask bits can be set with the LDMXCSR or FXRSTOR instruction; the mask and flag bits can be read with the STMXCSR or FXSAVE instruction.

The OSXMMEXCEPT flag (bit 10) of control register CR4 provides additional control over generation of SIMD floating-point exceptions by allowing the operating system to indicate whether or not it supports software exception handlers for SIMD floating-point exceptions. If an unmasked SIMD floating-point exception is generated and the OSXMMEXCEPT flag is set, the processor invokes a software exception handler by generating a SIMD floating-point exception (#XM). If the OSXMMEXCEPT bit is clear, the processor generates an invalid-opcode exception (#UD) on the first Intel SSE or SSE2 instruction that detects a SIMD floating-point exception condition. See Section 11.6.2, "Checking for Intel® SSE and SSE2 Support."

11.5.2 SIMD Floating-Point Exception Conditions

The following sections describe the conditions that cause a SIMD floating-point exception to be generated and the masked response of the processor when these conditions are detected.

See Section 4.9.2, "Floating-Point Exception Priority," for a description of the rules for exception precedence when more than one floating-point exception condition is detected for an instruction.

11.5.2.1 Invalid Operation Exception (#I)

The floating-point invalid-operation exception (#I) occurs in response to an invalid arithmetic operand. The flag (IE) and mask (IM) bits for the invalid operation exception are bits 0 and 7, respectively, in the MXCSR register.

If the invalid-operation exception is masked, the processor returns a QNaN, QNaN floating-point indefinite, integer indefinite, one of the source operands to the destination operand, or it sets the EFLAGS, depending on the operation being performed. When a value is returned to the destination operand, it overwrites the destination register specified by the instruction. Table 11-1 lists the invalid-arithmetic operations that the processor detects for instructions and the masked responses to these operations.

Table 11-1. Masked Responses of Intel® SSE, SSE2, and SSE3 Instructions to Invalid Arithmetic Operations

Condition	Masked Response
ADDPS, ADDSS, ADDPD, ADDSD, SUBPS, SUBSS, SUBPD, SUBSD, MULPS, MULSS, MULPD, MULSD, DIVPS, DIVSS, DIVPD, DIVSD, ADDSUBPD, ADDSUBPD, HADDPD, HADDPS, HSUBPD or HSUBPS instruction with an SNaN operand	Return the SNaN converted to a QNaN; Refer to Table 4-7 for more details.
SQRTPS, SQRTSS, SQRTPD, or SQRTSD with SNaN operands	Return the SNaN converted to a QNaN.
SQRTPS, SQRTSS, SQRTPD, or SQRTSD with negative operands (except zero)	Return the QNaN floating-point Indefinite.
MAXPS, MAXSS, MAXPD, MAXSD, MINPS, MINSS, MINPD, or MINSD instruction with QNaN or SNaN operands	Return the source 2 operand value.
CMPPS, CMPSS, CMPPD or CMPSD instruction with QNaN or SNaN operands	Return a mask of all 0s (except for the predicates "not-equal," "unordered," "not-less-than," or "not-less-than-or-equal," which returns a mask of all 1s).
CVTPD2PS, CVTSD2SS, CVTPS2PD, CVTSS2SD with SNaN operands	Return the SNaN converted to a QNaN.
COMISS or COMISD with QNaN or SNaN operand(s)	Set EFLAGS values to "not comparable."
Addition of opposite signed infinities or subtraction of like-signed infinities	Return the QNaN floating-point Indefinite.
Multiplication of infinity by zero	Return the QNaN floating-point Indefinite.
Divide of (0/0) or (∞ / ∞)	Return the QNaN floating-point Indefinite.

Table 11-1. Masked Responses of Intel® SSE, SSE2, and SSE3 Instructions to Invalid Arithmetic Operations (Contd.)

Condition	Masked Response
Conversion to integer when the value in the source register is a NaN, ∞, or exceeds the representable range for CVTPS2PI, CVTTPS2PI, CVTSS2SI, CVTTSS2SI, CVTPD2PI, CVTSD2SI, CVTPD2DQ, CVTTPD2PI, CVTTSD2SI, CVTTPD2DQ, CVTPS2DQ, or CVTTPS2DQ	Return the integer Indefinite.

If the invalid operation exception is not masked, a software exception handler is invoked and the operands remain unchanged. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software."

Normally, when one or more of the source operands are QNaNs (and neither is an SNaN or in an unsupported format), an invalid-operation exception is not generated. The following instructions are exceptions to this rule: the COMISS and COMISD instructions; and the CMPPS, CMPSS, CMPPD, and CMPSD instructions (when the predicate is less than, less-than or equal, not less-than, or not less-than or equal). With these instructions, a QNaN source operand will generate an invalid-operation exception.

The invalid-operation exception is not affected by the flush-to-zero mode or by the denormals-are-zeros mode.

11.5.2.2 Denormal-Operand Exception (#D)

The processor signals the denormal-operand exception if an arithmetic instruction attempts to operate on a denormal operand. The flag (DE) and mask (DM) bits for the denormal-operand exception are bits 1 and 8, respectively, in the MXCSR register.

The CVTPI2PD, CVTPD2PI, CVTTPD2PI, CVTDQ2PD, CVTPD2DQ, CVTTPD2DQ, CVTSI2SD, CVTSD2SI, CVTTSD2SI, CVTPI2PS, CVTPS2PI, CVTTPS2PI, CVTSS2SI, CVTTSS2SI, CVTSI2SS, CVTDQ2PS, CVTPS2DQ, and CVTTPS2DQ conversion instructions do not signal denormal exceptions. The RCPSS, RCPPS, RSQRTSS, and RSQRTPS instructions do not signal any kind of floating-point exception.

The denormals-are-zero flag (bit 6) of the MXCSR register provides an additional option for handling denormal-operand exceptions. When this flag is set, denormal source operands are automatically converted to zeros with the sign of the source operand (see Section 10.2.3.4, "Denormals-Are-Zeros"). The denormal operand exception is not affected by the flush-to-zero mode.

See Section 4.9.1.2, "Denormal Operand Exception (#D)," for more information about the denormal exception. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software," for information on handling unmasked exceptions.

11.5.2.3 Divide-By-Zero Exception (#Z)

The processor reports a divide-by-zero exception when a DIVPS, DIVSS, DIVPD or DIVSD instruction attempts to divide a finite non-zero operand by 0. The flag (ZE) and mask (ZM) bits for the divide-by-zero exception are bits 2 and 9, respectively, in the MXCSR register.

See Section 4.9.1.3, "Divide-By-Zero Exception (#Z)," for more information about the divide-by-zero exception. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software," for information on handling unmasked exceptions.

The divide-by-zero exception is not affected by the flush-to-zero mode at a single-instruction boundary.

While DAZ does not affect the rules for signaling IEEE exceptions, operations on denormal inputs might have different results when DAZ=1. As a consequence, DAZ can have an effect on the floating-point exceptions - including the divide-by-zero exception - when observed for a given operation involving denormal inputs.

11.5.2.4 Numeric Overflow Exception (#O)

The processor reports a numeric overflow exception whenever the rounded result of an arithmetic instruction exceeds the largest allowable finite value that fits in the destination operand. This exception can be generated with the ADDPS, ADDSS, ADDPD, ADDSD, SUBPS, SUBSS, SUBPD, SUBSD, MULPS, MULSS, MULPD, MULSD, DIVPS, DIVSS, DIVPD, DIVSD, CVTPD2PS, CVTSD2SS, ADDSUBPD, ADDSUBPS, HADDPD, HADDPS, HSUBPD, and

HSUBPS instructions. The flag (OE) and mask (OM) bits for the numeric overflow exception are bits 3 and 10, respectively, in the MXCSR register.

See Section 4.9.1.4, "Numeric Overflow Exception (#O)," for more information about the numeric-overflow exception. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software," for information on handling unmasked exceptions.

The numeric overflow exception is not affected by the flush-to-zero mode or by the denormals-are-zeros mode.

11.5.2.5 Numeric Underflow Exception (#U)

The processor reports a numeric underflow exception whenever the magnitude of the rounded result of an arithmetic instruction, with unbounded exponent, is less than the smallest possible normalized, finite value that will fit in the destination operand and the numeric-underflow exception is not masked. If the numeric underflow exception is masked, both underflow and the inexact-result condition must be detected before numeric underflow is reported. This exception can be generated with the ADDPS, ADDSS, ADDPD, ADDSD, SUBPS, SUBSS, SUBPD, SUBSD, MULPS, MULSS, MULPD, MULSD, DIVPS, DIVSS, DIVPD, DIVSD, CVTPD2PS, CVTSD2SS, ADDSUBPD, ADDSUBPS, HADDPD, HADDPS, HSUBPD, and HSUBPS instructions. The flag (UE) and mask (UM) bits for the numeric underflow exception are bits 4 and 11, respectively, in the MXCSR register.

The flush-to-zero flag (bit 15) of the MXCSR register provides an additional option for handling numeric underflow exceptions. When this flag is set and the numeric underflow exception is masked, tiny results are returned as a zero with the sign of the true result; see Section 10.2.3.3, "Flush-To-Zero."

Underflow will occur when a tiny non-zero result is detected (the result has to be also inexact if underflow exceptions are masked), as described in the IEEE Standard 754-2008. While DAZ does not affect the rules for signaling IEEE exceptions, operations on denormal inputs might have different results when DAZ=1. As a consequence, DAZ can have an effect on the floating-point exceptions - including the underflow exception - when observed for a given operation involving denormal inputs.

See Section 4.9.1.5, "Numeric Underflow Exception (#U)," for more information about the numeric underflow exception. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software," for information on handling unmasked exceptions.

11.5.2.6 Inexact-Result (Precision) Exception (#P)

The inexact-result exception (also called the precision exception) occurs if the result of an operation is not exactly representable in the destination format. For example, the fraction 1/3 cannot be precisely represented in binary form. This exception occurs frequently and indicates that some (normally acceptable) accuracy has been lost. The exception is supported for applications that need to perform exact arithmetic only. Because the rounded result is generally satisfactory for most applications, this exception is commonly masked.

The flag (PE) and mask (PM) bits for the inexact-result exception are bits 5 and 12, respectively, in the MXCSR register.

See Section 4.9.1.6, "Inexact-Result (Precision) Exception (#P)," for more information about the inexact-result exception. See Section 11.5.4, "Handling SIMD Floating-Point Exceptions in Software," for information on handling unmasked exceptions.

In flush-to-zero mode, the inexact result exception is reported.

11.5.3 Generating SIMD Floating-Point Exceptions

When the processor executes a packed or scalar floating-point instruction, it looks for and reports on SIMD floating-point exception conditions using two sequential steps:

1. Looks for, reports on, and handles pre-computation exception conditions (invalid-operand, divide-by-zero, and denormal operand)
2. Looks for, reports on, and handles post-computation exception conditions (numeric overflow, numeric underflow, and inexact result)

If both pre- and post-computational exceptions are unmasked, it is possible for the processor to generate a SIMD floating-point exception (#XM) twice during the execution of an SSE, SSE2 or SSE3 instruction: once when it detects and handles a pre-computational exception and when it detects a post-computational exception.

11.5.3.1 Handling Masked Exceptions

If all exceptions are masked, the processor handles the exceptions it detects by placing the masked result (or results for packed operands) in a destination operand and continuing program execution. The masked result may be a rounded normalized value, signed infinity, a denormal finite number, zero, a QNaN floating-point indefinite, or a QNaN depending on the exception condition detected. In most cases, the corresponding exception flag bit in MXCSR is also set. The one situation where an exception flag is not set is when an underflow condition is detected and it is not accompanied by an inexact result.

When operating on packed floating-point operands, the processor returns a masked result for each of the sub-operand computations and sets a separate set of internal exception flags for each computation. It then performs a logical-OR on the internal exception flag settings and sets the exception flags in the MXCSR register according to the results of OR operations.

For example, Figure 11-9 shows the results of an MULPS instruction. In the example, all SIMD floating-point exceptions are masked. Assume that a denormal exception condition is detected prior to the multiplication of sub-operands X0 and Y0, no exception condition is detected for the multiplication of X1 and Y1, a numeric overflow exception condition is detected for the multiplication of X2 and Y2, and another denormal exception is detected prior to the multiplication of sub-operands X3 and Y3. Because denormal exceptions are masked, the processor uses the denormal source values in the multiplications of (X0 and Y0) and of (X3 and Y3) passing the results of the multiplications through to the destination operand. With the denormal operand, the result of the X0 and Y0 computation is a normalized finite value, with no exceptions detected. However, the X3 and Y3 computation produces a tiny and inexact result. This causes the corresponding internal numeric underflow and inexact-result exception flags to be set.

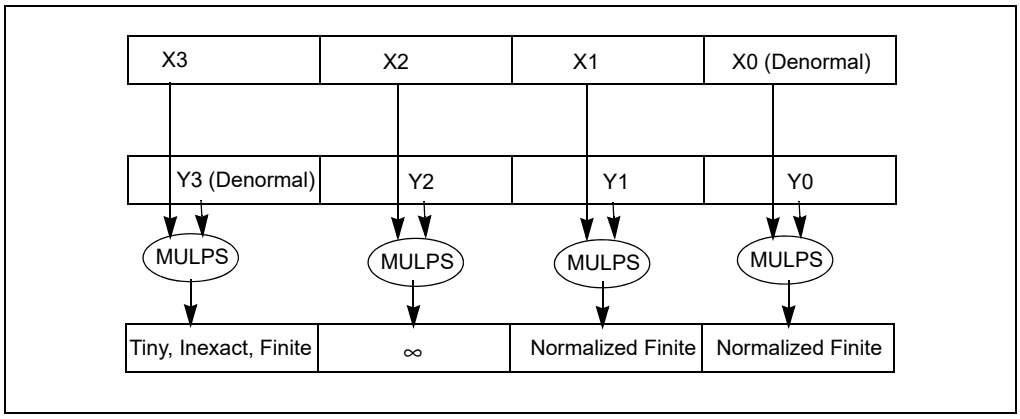


Figure 11-9. Example Masked Response for Packed Operations

For the multiplication of X2 and Y2, the processor stores the floating-point ∞ in the destination operand, and sets the corresponding internal sub-operand numeric overflow flag. The result of the X1 and Y1 multiplication is passed through to the destination operand, with no internal sub-operand exception flags being set. Following the computations, the individual sub-operand exceptions flags for denormal operand, numeric underflow, inexact result, and numeric overflow are OR'd and the corresponding flags are set in the MXCSR register.

The net result of this computation is that:

- Multiplication of X0 and Y0 produces a normalized finite result
- Multiplication of X1 and Y1 produces a normalized finite result
- Multiplication of X2 and Y2 produces a floating-point ∞ result
- Multiplication of X3 and Y3 produces a tiny, inexact, finite result

- Denormal operand, numeric underflow, numeric underflow, and inexact result flags are set in the MXCSR register

11.5.3.2 Handling Unmasked Exceptions

If all exceptions are unmasked, the processor:

1. First detects any pre-computation exceptions: it ORs those exceptions, sets the appropriate exception flags, leaves the source and destination operands unaltered, and goes to step 2. If it does not detect any pre-computation exceptions, it goes to step 5.
2. Checks CR4.OSXMMEXCPT[bit 10]. If this flag is set, the processor goes to step 3; if the flag is clear, it generates an invalid-opcode exception (#UD) and makes an implicit call to the invalid-opcode exception handler.
3. Generates a SIMD floating-point exception (#XM) and makes an implicit call to the SIMD floating-point exception handler.
4. If the exception handler is able to fix the source operands that generated the pre-computation exceptions or mask the condition in such a way as to allow the processor to continue executing the instruction, the processor resumes instruction execution as described in step 5.
5. Upon returning from the exception handler (or if no pre-computation exceptions were detected), the processor checks for post-computation exceptions. If the processor detects any post-computation exceptions: it ORs those exceptions, sets the appropriate exception flags, leaves the source and destination operands unaltered, and repeats steps 2, 3, and 4.
6. Upon returning from the exceptions handler in step 4 (or if no post-computation exceptions were detected), the processor completes the execution of the instruction.

The implication of this procedure is that for unmasked exceptions, the processor can generate a SIMD floating-point exception (#XM) twice: once if it detects pre-computation exception conditions and a second time if it detects post-computation exception conditions. For example, if SIMD floating-point exceptions are unmasked for the computation shown in Figure 11-9, the processor would generate one SIMD floating-point exception for denormal operand conditions and a second SIMD floating-point exception for overflow and underflow (no inexact result exception would be generated because the multiplications of X0 and Y0 and of X1 and Y1 are exact).

11.5.3.3 Handling Combinations of Masked and Unmasked Exceptions

In situations where both masked and unmasked exceptions are detected, the processor will set exception flags for the masked and the unmasked exceptions. However, it will not return masked results until after the processor has detected and handled unmasked post-computation exceptions and returned from the exception handler (as in step 6 above) to finish executing the instruction.

11.5.4 Handling SIMD Floating-Point Exceptions in Software

Section 4.9.3, "Typical Actions of a Floating-Point Exception Handler," shows actions that may be carried out by a SIMD floating-point exception handler. The SSE/SSE2/SSE3 state is saved with the FXSAVE instruction; see Section 11.6.5, "Saving and Restoring the SSE/SSE2 State."

11.5.5 Interaction of SIMD and x87 FPU Floating-Point Exceptions

SIMD floating-point exceptions are generated independently from x87 FPU floating-point exceptions. SIMD floating-point exceptions do not cause assertion of the FERR# pin (independent of the value of CR0.NE[bit 5]). They ignore the assertion and deassertion of the IGNNE# pin.

If applications use Intel SSE/SSE2/SSE3 instructions along with x87 FPU instructions (in the same task or program), consider the following:

- SIMD floating-point exceptions are reported independently from the x87 FPU floating-point exceptions. SIMD and x87 FPU floating-point exceptions can be unmasked independently. Separate x87 FPU and SIMD floating-

point exception handlers must be provided if the same exception is unmasked for x87 FPU and for Intel SSE/SSE2/SSE3 operations.

- The rounding mode specified in the MXCSR register does not affect x87 FPU instructions. Likewise, the rounding mode specified in the x87 FPU control word does not affect the Intel SSE/SSE2/SSE3 instructions. To use the same rounding mode, the rounding control bits in the MXCSR register and in the x87 FPU control word must be set explicitly to the same value.
- The flush-to-zero mode set in the MXCSR register for Intel SSE/SSE2/SSE3 instructions has no counterpart in the x87 FPU. For compatibility with the x87 FPU, set the flush-to-zero bit to 0.
- The denormals-are-zeros mode set in the MXCSR register for Intel SSE/SSE2/SSE3 instructions has no counterpart in the x87 FPU. For compatibility with the x87 FPU, set the denormals-are-zeros bit to 0.
- An application that expects to detect x87 FPU exceptions that occur during the execution of x87 FPU instructions will not be notified if exceptions occurs during the execution of corresponding Intel SSE/SSE2/SSE3[1] instructions, unless the exception masks that are enabled in the x87 FPU control word have also been enabled in the MXCSR register and the application is capable of handling SIMD floating-point exceptions (#XM).
  - — Masked exceptions that occur during an SSE/SSE2/SSE3 library call cannot be detected by unmasking the exceptions after the call (in an attempt to generate the fault based on the fact that an exception flag is set). A SIMD floating-point exception flag that is set when the corresponding exception is unmasked will not generate a fault; only the next occurrence of that unmasked exception will generate a fault.
  - — An application which checks the x87 FPU status word to determine if any masked exception flags were set during an x87 FPU library call will also need to check the MXCSR register to detect a similar occurrence of a masked exception flag being set during an SSE/SSE2/SSE3 library call.

11.6 WRITING APPLICATIONS WITH INTEL® SSE AND SSE2

The following sections give some guidelines for writing application programs and operating-system code that uses Intel SSE and SSE2. Because Intel SSE and SSE2 share the same state and perform companion operations, these guidelines apply to both sets of extensions.

Chapter 14 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, discusses the interface to the processor for context switching as well as other operating system considerations when writing code that uses Intel SSE, SSE2, and SSE3.

11.6.1 General Guidelines for Using Intel® SSE and SSE2

The following guidelines describe how to take full advantage of the performance gains available with Intel SSE and SSE2:

- Ensure that the processor supports Intel SSE and SSE2.
- Ensure that your operating system supports Intel SSE and SSE2. (Operating system support for Intel SSE implies support for Intel SSE2, and vice versa.)
- Use stack and data alignment techniques to keep data properly aligned for efficient memory use.
- Use the non-temporal store instructions offered with Intel SSE and SSE2.
- Employ the optimization and scheduling techniques described in the Intel® 64 and IA-32 Architectures Optimization Reference Manual; see Section 1.4, "Related Literature," for the order number for this manual.

11.6.2 Checking for Intel® SSE and SSE2 Support

Before an application attempts to use Intel SSE and/or Intel SSE2, it should check that they are present on the processor:

1. Intel SSE3 refers to ADDSUBPD, ADDSUBPS, HADDPD, HADDPS, HSUBPD, and HSUBPS. The only other Intel SSE3 instruction that can raise floating-point exceptions is FISTTP; it can generate x87 FPU invalid operation and inexact result exceptions.

1. Check that the processor supports the CPUID instruction. Bit 21 of the EFLAGS register can be used to check processor's support the CPUID instruction.
2. Check that the processor supports Intel SSE and/or SSE2 (true if CPUID.01H:EDX.SSE[bit 25] = 1 and/or CPUID.01H:EDX.SSE2[bit 26] = 1).

The operating system must provide system level support for handling SSE state, exceptions before an application can use Intel SSE and/or Intel SSE2; see Chapter 14 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

If the processor attempts to execute an unsupported Intel SSE or SSE2 instruction, the processor will generate an invalid-opcode exception (#UD). If an operating system did not provide adequate system level support for Intel SSE, executing an Intel SSE or SSE2 instructions can also generate #UD.

11.6.3 Checking for the DAZ Flag in the MXCSR Register

The denormals-are-zero flag in the MXCSR register is available in most of the Pentium 4 processors and in the Intel Xeon processor, with the exception of some early steppings. To check for the presence of the DAZ flag in the MXCSR register, do the following:

1. Establish a 512-byte FXSAVE area in memory.
2. Clear the FXSAVE area to all 0s.
3. Execute the FXSAVE instruction, using the address of the first byte of the cleared FXSAVE area as a source operand. See "FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a description of the FXSAVE instruction and the layout of the FXSAVE image.
4. Check the value in the MXCSR_MASK field in the FXSAVE image (bytes 28 through 31).
   - If the value of the MXCSR_MASK field is 00000000H, the DAZ flag and denormals-are-zero mode are not supported.
   - If the value of the MXCSR_MASK field is non-zero and bit 6 is set, the DAZ flag and denormals-are-zero mode are supported.

If the DAZ flag is not supported, then it is a reserved bit and attempting to write a 1 to it will cause a general-protection exception (#GP). See Section 11.6.6, "Guidelines for Writing to the MXCSR Register," for general guidelines for preventing general-protection exceptions when writing to the MXCSR register.

11.6.4 Initialization of Intel® SSE and SSE2

The SSE and SSE2 state is contained in the XMM and MXCSR registers. Upon a hardware reset of the processor, this state is initialized as follows (see Table 11-2):

- All SIMD floating-point exceptions are masked (bits 7 through 12 of the MXCSR register is set to 1).
- All SIMD floating-point exception flags are cleared (bits 0 through 5 of the MXCSR register is set to 0).
- The rounding control is set to round-nearest (bits 13 and 14 of the MXCSR register are set to 00B).
- The flush-to-zero mode is disabled (bit 15 of the MXCSR register is set to 0).
- The denormals-are-zeros mode is disabled (bit 6 of the MXCSR register is set to 0). If the denormals-are-zeros mode is not supported, this bit is reserved and will be set to 0 on initialization.
- Each of the XMM registers is cleared (set to all zeros).

Table 11-2. SSE and SSE2 State Following a Power-up/Reset or INIT

Registers	Power-Up or Reset	INIT
XMM0 through XMM7	+0.0	Unchanged
MXCSR	1F80H	Unchanged

If the processor is reset by asserting the INIT# pin, the SSE and SSE2 state is not changed.

11.6.5 Saving and Restoring the SSE/SSE2 State

The FXSAVE instruction saves the x87 FPU, MMX, SSE, and SSE2 states (which includes the contents of eight XMM registers and the MXCSR registers) in a 512-byte block of memory. The FXRSTOR instruction restores the saved SSE and SSE2 state from memory. See the FXSAVE instruction in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for the layout of the 512-byte state block.

In addition to saving and restoring the SSE and SSE2 state, FXSAVE and FXRSTOR also save and restore the x87 FPU state (because MMX registers are aliased to the x87 FPU data registers this includes saving and restoring the MMX state). For greater code efficiency, it is suggested that FXSAVE and FXRSTOR be substituted for the FSAVE, FNSAVE, and FRSTOR instructions in the following situations:

- When a context switch is being made in a multitasking environment
- During calls and returns from interrupt and exception handlers

In situations where the code is switching between x87 FPU and MMX technology computations (without a context switch or a call to an interrupt or exception), the FSAVE/FNSAVE and FRSTOR instructions are more efficient than the FXSAVE and FXRSTOR instructions.

11.6.6 Guidelines for Writing to the MXCSR Register

The MXCSR has several reserved bits, and attempting to write a 1 to any of these bits will cause a general-protection exception (#GP) to be generated. To allow software to identify these reserved bits, the MXCSR_MASK value is provided. Software can determine this mask value as follows:

1. Establish a 512-byte FXSAVE area in memory.
2. Clear the FXSAVE area to all 0s.
3. Execute the FXSAVE instruction, using the address of the first byte of the cleared FXSAVE area as a source operand. See "FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a description of FXSAVE and the layout of the FXSAVE image.
4. Check the value in the MXCSR_MASK field in the FXSAVE image (bytes 28 through 31).
   - If the value of the MXCSR_MASK field is 00000000H, then the MXCSR_MASK value is the default value of 0000FFBFH. Note that this value indicates that bit 6 of the MXCSR register is reserved; this setting indicates that the denormals-are-zero mode is not supported on the processor.
   - If the value of the MXCSR_MASK field is non-zero, the MXCSR_MASK value should be used as the MXCSR_MASK.

All bits set to 0 in the MXCSR_MASK value indicate reserved bits in the MXCSR register. Thus, if the MXCSR_MASK value is AND'd with a value to be written into the MXCSR register, the resulting value will be assured of having all its reserved bits set to 0, preventing the possibility of a general-protection exception being generated when the value is written to the MXCSR register.

For example, the default MXCSR_MASK value when 00000000H is returned in the FXSAVE image is 0000FFBFH. If software AND's a value to be written to MXCSR register with 0000FFBFH, bit 6 of the result (the DAZ flag) will be ensured of being set to 0, which is the required setting to prevent general-protection exceptions on processors that do not support the denormals-are-zero mode.

To prevent general-protection exceptions, the MXCSR_MASK value should be AND'd with the value to be written into the MXCSR register in the following situations:

- Operating system routines that receive a parameter from an application program and then write that value to the MXCSR register (either with an FXRSTOR or LDMXCSR instruction)
- Any application program that writes to the MXCSR register and that needs to run robustly on several different IA-32 processors

Note that all bits in the MXCSR_MASK value that are set to 1 indicate features that are supported by the MXCSR register; they can be treated as feature flags for identifying processor capabilities.

11.6.7 Interaction of Intel® SSE and SSE2 Instructions with x87 FPU and MMX Instructions

The XMM registers and the x87 FPU and MMX registers represent separate execution environments, which has certain ramifications when executing Intel SSE, SSE2, MMX, and x87 FPU instructions in the same code module or when mixing code modules that contain these instructions:

- Those Intel SSE and SSE2 instructions that operate only on XMM registers (such as the packed and scalar floating-point instructions and the 128-bit SIMD integer instructions) in the same instruction stream with 64-bit SIMD integer or x87 FPU instructions without any restrictions. For example, an application can perform the majority of its floating-point computations in the XMM registers, using the packed and scalar floating-point instructions, and at the same time use the x87 FPU to perform trigonometric and other transcendental computations. Likewise, an application can perform packed 64-bit and 128-bit SIMD integer operations together without restrictions.
- Those Intel SSE and SSE2 instructions that operate on MMX registers (such as the CVTPS2PI, CVTTPS2PI, CVTPI2PS, CVTPD2PI, CVTTPD2PI, CVTPI2PD, MOVDQ2Q, MOVQ2DQ, PADDQ, and PSUBQ instructions) can also be executed in the same instruction stream as 64-bit SIMD integer or x87 FPU instructions, however, here they are subject to the restrictions on the simultaneous use of MMX technology and x87 FPU instructions, which include:
  - Transition from x87 FPU to MMX technology instructions or to Intel SSE or SSE2 instructions that operate on MMX registers should be preceded by saving the state of the x87 FPU.
  - Transition from MMX technology instructions or from Intel SSE or SSE2 instructions that operate on MMX registers to x87 FPU instructions should be preceded by execution of the EMMS instruction.

11.6.8 Compatibility of SIMD and x87 FPU Floating-Point Data Types

Intel SSE and SSE2 instructions operate on the same single precision and double precision floating-point data types that the x87 FPU operates on. However, when operating on these data types, Intel SSE and SSE2 operate on them in their native format (single precision or double precision), in contrast to the x87 FPU which extends them to double extended precision floating-point format to perform computations and then rounds the result back to a single precision or double precision format before writing results to memory. Because the x87 FPU operates on a higher precision format and then rounds the result to a lower precision format, it may return a slightly different result when performing the same operation on the same single precision or double precision floating-point values than is returned by Intel SSE and SSE2. The difference occurs only in the least-significant bits of the significand.

11.6.9 Mixing Packed and Scalar Floating-Point and 128-Bit SIMD Integer Instructions and Data

Intel SSE and SSE2 define typed operations on packed and scalar floating-point data types and on 128-bit SIMD integer data types, but IA-32 processors do not enforce this typing at the architectural level. They only enforce it at the microarchitectural level. Therefore, when a Pentium 4 or Intel Xeon processor loads a packed or scalar floating-point operand or a 128-bit packed integer operand from memory into an XMM register, it does not check that the actual data being loaded matches the data type specified in the instruction. Likewise, when the processor performs an arithmetic operation on the data in an XMM register, it does not check that the data being operated on matches the data type specified in the instruction.

As a general rule, because data typing of SIMD floating-point and integer data types is not enforced at the architectural level, it is the responsibility of the programmer, assembler, or compiler to ensure that code enforces data typing. Failure to enforce correct data typing can lead to computations that return unexpected results.

For example, in the following code sample, two packed single precision floating-point operands are moved from memory into XMM registers (using MOVAPS instructions); then a double precision packed add operation (using the ADDPD instruction) is performed on the operands:

```
movaps      xmm0, [eax]   ; EAX register contains pointer to packed
                          ; single precision floating-point operand
movaps      xmm1, [ebx]
addpd       xmm0, xmm1
```

Pentium 4 and Intel Xeon processors execute these instructions without generating an invalid-operand exception (#UD) and will produce the expected results in register XMM0 (that is, the high and low 64-bits of each register will be treated as a double precision floating-point value and the processor will operate on them accordingly). Because the data types operated on and the data type expected by the ADDPD instruction were inconsistent, the instruction may result in a SIMD floating-point exception (such as numeric overflow [#O] or invalid operation [#I]) being generated, but the actual source of the problem (inconsistent data types) is not detected.

The ability to operate on an operand that contains a data type that is inconsistent with the typing of the instruction being executed, permits some valid operations to be performed. For example, the following instructions load a packed double precision floating-point operand from memory to register XMM0, and a mask to register XMM1; then they use XORPD to toggle the sign bits of the two packed values in register XMM0.

```
movapd      xmm0, [eax]   ; EAX register contains pointer to packed
                          ; double precision floating-point operand
movaps      xmm1, [ebx]    ; EBX register contains pointer to packed
                           ; double precision floating-point mask
xorpd       xmm0, xmm1    ; XOR operation toggles sign bits using
                          ; the mask in xmm1
```

In this example: XORPS or PXOR can be used in place of XORPD and yield the same correct result. However, because of the type mismatch between the operand data type and the instruction data type, a latency penalty will be incurred due to implementations of the instructions at the microarchitecture level.

Latency penalties can also be incurred by using move instructions of the wrong type. For example, MOVAPS and MOVAPD can both be used to move a packed single precision operand from memory to an XMM register. However, if MOVAPD is used, a latency penalty will be incurred when a correctly typed instruction attempts to use the data in the register.

Note that these latency penalties are not incurred when moving data from XMM registers to memory.

11.6.10 Interfacing with Intel® SSE and SSE2 Procedures and Functions

Intel SSE and SSE2 allow direct access to XMM registers. This means that all existing interface conventions between procedures and functions that apply to the use of the general-purpose registers (EAX, EBX, etc.) also apply to XMM register usage.

11.6.10.1 Passing Parameters in XMM Registers

The state of XMM registers is preserved across procedure (or function) boundaries. Parameters can be passed from one procedure to another using XMM registers.

11.6.10.2 Saving XMM Register State on a Procedure or Function Call

The state of XMM registers can be saved in two ways: using an FXSAVE instruction or a move instruction. FXSAVE saves the state of all XMM registers (along with the state of MXCSR and the x87 FPU registers). This instruction is typically used for major changes in the context of the execution environment, such as a task switch. FXRSTOR restores the XMM, MXCSR, and x87 FPU registers stored with FXSAVE.

In cases where only XMM registers must be saved, or where selected XMM registers need to be saved, move instructions (MOVAPS, MOVUPS, MOVSS, MOVAPD, MOVUPD, MOVSD, MOVDQA, and MOVDQU) can be used. These instructions can also be used to restore the contents of XMM registers. To avoid performance degradation when saving XMM registers to memory or when loading XMM registers from memory, be sure to use the appropriately typed move instructions.

The move instructions can also be used to save the contents of XMM registers on the stack. Here, the stack pointer (in the ESP register) can be used as the memory address to the next available byte in the stack. Note that the stack pointer is not automatically incremented when using a move instruction (as it is with PUSH).

A move-instruction procedure that saves the contents of an XMM register to the stack is responsible for decrementing the value in the ESP register by 16. Likewise, a move-instruction procedure that loads an XMM register from the stack needs also to increment the ESP register by 16. To avoid performance degradation when moving the contents of XMM registers, use the appropriately typed move instructions.

Use the LDMXCSR and STMXCSR instructions to save and restore, respectively, the contents of the MXCSR register on a procedure call and return.

11.6.10.3 Caller-Save Recommendation for Procedure and Function Calls

When making procedure (or function) calls from SSE or SSE2 code, a caller-save convention is recommended for saving the state of the calling procedure. Using this convention, any register whose content must survive intact across a procedure call must be stored in memory by the calling procedure prior to executing the call.

The primary reason for using the caller-save convention is to prevent performance degradation. XMM registers can contain packed or scalar double precision floating-point, packed single precision floating-point, and 128-bit packed integer data types. The called procedure has no way of knowing the data types in XMM registers following a call; so it is unlikely to use the correctly typed move instruction to store the contents of XMM registers in memory or to restore the contents of XMM registers from memory.

As described in Section 11.6.9, "Mixing Packed and Scalar Floating-Point and 128-Bit SIMD Integer Instructions and Data," executing a move instruction that does not match the type for the data being moved to/from XMM registers will be carried out correctly, but can lead to a greater instruction latency.

11.6.11 Updating Existing MMX Technology Routines Using 128-Bit SIMD Integer Instructions

Intel SSE2 extends all 64-bit MMX SIMD integer instructions to operate on 128-bit SIMD integers using XMM registers. The extended 128-bit SIMD integer instructions operate like the 64-bit SIMD integer instructions; this simplifies the porting of MMX technology applications. However, there are considerations:

- To take advantage of wider 128-bit SIMD integer instructions, MMX technology code must be recompiled to reference the XMM registers instead of MMX registers.
- Computation instructions that reference memory operands that are not aligned on 16-byte boundaries should be replaced with an unaligned 128-bit load (MOVUDQ instruction) followed by a version of the same computation operation that uses register instead of memory operands. Use of 128-bit packed integer computation instructions with memory operands that are not 16-byte aligned results in a general protection exception (#GP).
- Extension of the PSHUFW instruction (shuffle word across 64-bit integer operand) across a full 128-bit operand is emulated by a combination of the following instructions: PSHUFHW, PSHUFLW, and PSHUFD.
- Use of the 64-bit shift by bit instructions (PSRLQ, PSLLQ) can be extended to 128 bits in either of two ways:
  - Use of PSRLQ and PSLLQ, along with masking logic operations.
  - Rewriting the code sequence to use PSRLDQ and PSLLDQ (shift double quadword operand by bytes)
- Loop counters need to be updated, since each 128-bit SIMD integer instruction operates on twice the amount of data as its 64-bit SIMD integer counterpart.

11.6.12 Branching on Arithmetic Operations

There are no condition codes in SSE or SSE2 states. A packed-data comparison instruction generates a mask which can then be transferred to an integer register. The following code sequence provides an example of how to perform a conditional branch, based on the result of an Intel SSE2 arithmetic operation.

```
cmppd       XMM0, XMM1          ; generates a mask in XMM0
movmskpd    EAX, XMM0           ; moves a 2 bit mask to eax
test        EAX, 0              ; compare with desired result
jne         BRANCH TARGET
```

The COMISD and UCOMISD instructions update the EFLAGS as the result of a scalar comparison. A conditional branch can then be scheduled immediately following COMISD/UCOMISD.

11.6.13 Cacheability Hint Instructions

Intel SSE and SSE2 cacheability control instructions enable the programmer to control prefetching, caching, loading, and storing of data. When correctly used, these instructions improve application performance.

To make efficient use of the processor's super-scalar microarchitecture, a program needs to provide a steady stream of data to the executing program to avoid stalling the processor. PREFETCH*h* instructions minimize the latency of data accesses in performance-critical sections of application code by allowing data to be fetched into the processor cache hierarchy in advance of actual usage.

PREFETCH*h* instructions do not change the user-visible semantics of a program, although they may affect performance. The operation of these instructions is implementation-dependent. Programmers may need to tune code for each IA-32 processor implementation. Excessive usage of PREFETCH*h* instructions may waste memory bandwidth and reduce performance. For more detailed information on the use of prefetch hints, refer to Chapter 7, "Optimizing Cache Usage," in the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

The non-temporal store instructions (MOVNTI, MOVNTPD, MOVNTPS, MOVNTDQ, MOVNTQ, MASKMOVQ, and MASKMOVDQU) minimize cache pollution when writing non-temporal data to memory (see Section 10.4.6.1, "Cacheability Control Instructions," and Section 10.4.6.2, "Caching of Temporal vs. Non-Temporal Data"). They prevent non-temporal data from being written into processor caches on a store operation.

Besides reducing cache pollution, the use of weakly-ordered memory types can be important under certain data sharing relationships, such as a producer-consumer relationship. The use of weakly ordered memory can make the assembling of data more efficient; but care must be taken to ensure that the consumer obtains the data that the producer intended. Some common usage models that may be affected in this way by weakly-ordered stores are:

- Library functions that use weakly ordered memory to write results.
- Compiler-generated code that writes weakly-ordered results.
- Hand-crafted code.

The degree to which a consumer of data knows that the data is weakly ordered can vary for these cases. As a result, the SFENCE or MFENCE instruction should be used to ensure ordering between routines that produce weakly-ordered data and routines that consume the data. SFENCE and MFENCE provide a performance-efficient way to ensure ordering by guaranteeing that every store instruction that precedes SFENCE/MFENCE in program order is globally visible before a store instruction that follows the fence.

11.6.14 Effect of Instruction Prefixes on Intel® SSE and SSE2 Instructions

Table 11-3 describes the effects of instruction prefixes on Intel SSE and SSE2 instructions. (Table 11-3 also applies to SIMD integer and SIMD floating-point instructions in Intel SSE3.) Unpredictable behavior can range from prefixes being treated as a reserved operation on one generation of IA-32 processors to generating an invalid opcode exception on another generation of processors.

See also "Instruction Prefixes" in Chapter 2 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for complete description of instruction prefixes.

NOTE

Some Intel SSE, SSE2, and SSE3 instructions have two-byte opcodes that are either 2 bytes or 3 bytes in length. Two-byte opcodes that are 3 bytes in length consist of: a mandatory prefix (F2H, F3H, or 66H), 0FH, and an opcode byte. See Table 11-3.

Table 11-3. Effect of Prefixes on the Intel® SSE, SSE2, and SSE3 Instructions

Prefix Type	Effect on the Intel® SSE, SSE2, and SSE3 Instructions
Address Size Prefix (67H)	Affects instructions with a memory operand.
	Reserved for instructions without a memory operand and may result in unpredictable behavior.
Operand Size (66H)	Reserved and may result in unpredictable behavior.
Segment Override (2EH, 36H, 3EH, 26H, 64H, 65H)	Affects instructions with a memory operand.
	Reserved for instructions without a memory operand and may result in unpredictable behavior.
Repeat Prefixes (F2H and F3H)	Reserved and may result in unpredictable behavior.
Lock Prefix (F0H)	Reserved; generates invalid opcode exception (#UD).

4. Updates to Chapter 18, Volume 1

Change bars and violet text show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

Changes to this chapter:

- Updated Section 18.2, "Recommendations for System Software," to clarify that maintaining AMX state in a non-initialized state will prevent the execution of In-Field Scan tests.

CHAPTER 18
PROGRAMMING WITH INTEL® ADVANCED MATRIX EXTENSIONS

18.1 INTRODUCTION

Intel® Advanced Matrix Extensions (Intel® AMX) is a new 64-bit programming paradigm consisting of two components: a set of 2-dimensional registers (tiles) representing sub-arrays from a larger 2-dimensional memory image, and an accelerator able to operate on tiles, the first implementation is called TMUL (tile matrix multiply unit).

An Intel AMX implementation enumerates to the programmer how the tiles can be programmed by providing a palette of options. Two palettes are supported; palette 0 represents the initialized state, and palette 1 consists of 8 KB of storage spread across 8 tile registers named TMM0..TMM7. Each tile has a maximum size of 16 rows x 64 bytes, (1 KB), however the programmer can configure each tile to smaller dimensions appropriate to their algorithm. The tile dimensions supplied by the programmer (rows and bytes_per_row, i.e., **colsb**) are metadata that drives the execution of tile and accelerator instructions. In this way, a single instruction can launch autonomous multi-cycle execution in the tile and accelerator hardware. The palette value (**palette_id**) and metadata are held internally in a tile related control register (TILECFG). The TILECFG contents will be commensurate with that reported in the palette_table (see "CPUID—CPU Identification" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for a description of the available parameters).

Intel AMX is an extensible architecture. New accelerators can be added, or the TMUL accelerator may be enhanced to provide higher performance. In these cases, the state (TILEDATA) provided by tiles may need to be made larger, either in one of the metadata dimensions (more rows or colsb) and/or by supporting more tile registers (names). The extensibility is carried out by adding new palette entries describing the additional state. Since execution is driven through metadata, an existing Intel AMX binary could take advantage of larger storage sizes and higher performance TMUL units by selecting the most powerful palette indicated by CPUID and adjusting loop and pointer updates accordingly.

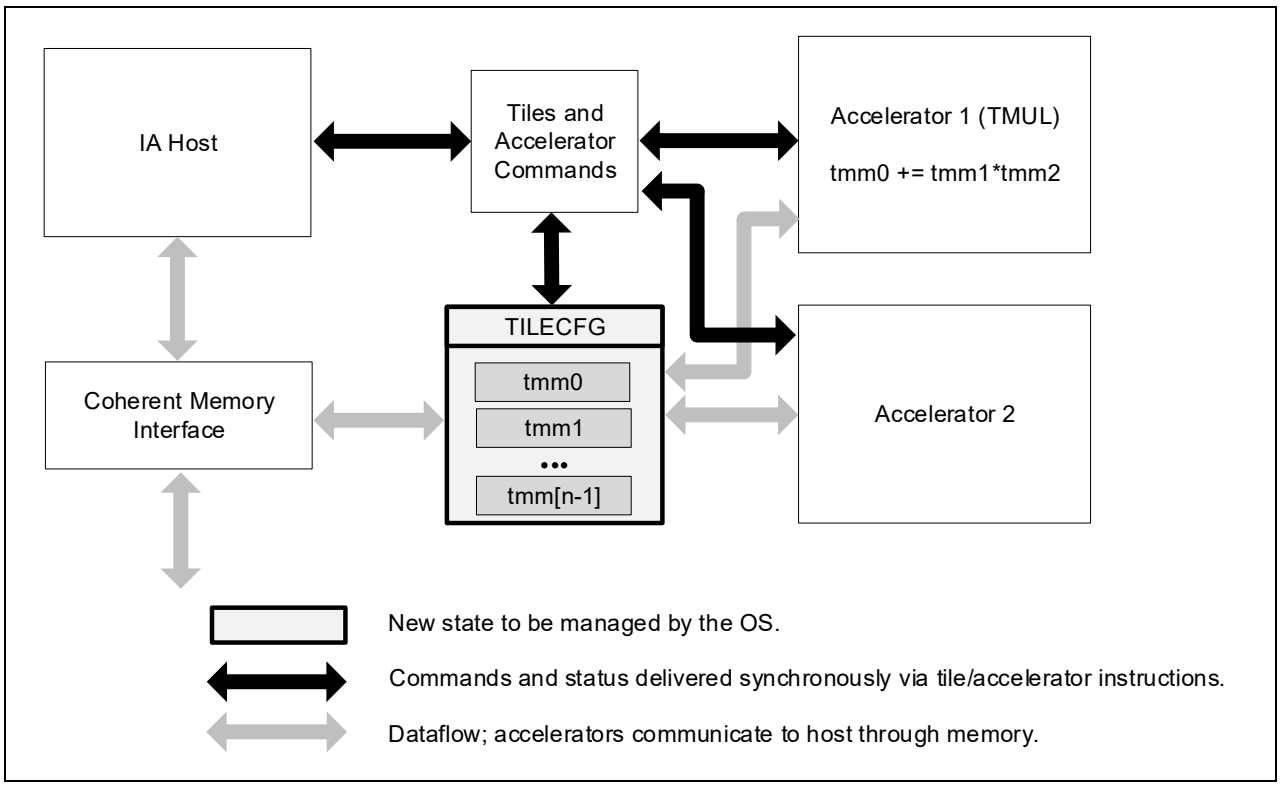


Figure 18-1. Intel® AMX Architecture

Figure 18-1 shows a conceptual diagram of the Intel AMX architecture. An Intel architecture host drives the algorithm, the memory blocking, loop indices and pointer arithmetic. Tile loads and stores and accelerator commands are sent to multi-cycle execution units. Status, if required, is reported back. Intel AMX instructions are synchronous in the Intel architecture instruction stream and the memory loaded and stored by the tile instructions is coherent with respect to the host's memory accesses. There are no restrictions on interleaving of Intel architecture and Intel AMX code or restrictions on the resources the host can use in parallel with Intel AMX (e.g., Intel AVX-512). There is also no architectural requirement on the Intel architecture compute capability of the Intel architecture host other than it supports 64-bit mode.

Intel AMX instructions use new registers and inherit basic behavior from Intel architecture in the same manner that Intel SSE and Intel AVX did. Tile instructions include loads and stores using the traditional Intel architecture register set as pointers. The TMUL instruction set (defined to be CPUID bits AMX-BF16 and AMX-INT8) only supports reg-reg operations.

TILECFG is programmed using the LDTILECFG instruction. The selected palette defines the available storage and general configuration while the rest of the memory data specifies the number of rows and column bytes for each tile. Consistency checks are performed to ensure the TILECFG matches the restrictions of the palette. A General Protection fault (#GP) is reported if the LDTILECFG fails consistency checks. A successful load of TILECFG with a palette_id other than 0 is represented in this document with TILES_CONFIGURED = 1. When the TILECFG is initialized (palette_id = 0), it is represented in the document as TILES_CONFIGURED = 0. Nearly all Intel AMX instructions will generate a #UD exception if TILES_CONFIGURED is not equal to 1; the exceptions are those that do TILECFG maintenance: LDTILECFG, STTILECFG, and TILERELEASE.

If a tile is configured to contain M rows by N column bytes, LDTILECFG will ensure that the metadata values are appropriate to the palette (e.g., that $M \leq 16$ and $N \leq 64$ for palette 1). The four M and N values can all be different as long as they adhere to the restrictions of the palette. Further dynamic checks are done in the tile and the TMUL instruction set to deal with cases where a legally configured tile may be inappropriate for the instruction operation. Tile registers can be set to 'invalid' by configuring the rows and colsb to '0'.

Tile loads and stores are strided accesses from the application memory to packed rows of data. Algorithms are expressed assuming row major data layout. Column major users should translate the terms according to their orientation.

TILELOAD* and TILESTORE* instructions are restartable and can handle (up to) 2*rows page faults per instruction. Restartability is provided by a **start_row** parameter in the TILECFG register.

The TMUL unit is conceptually a grid of fused multiply-add units able to read and write tiles. The dimensions of the TMUL unit (tmul_maxk and tmul_maxn) are enumerated similar to the maximum dimensions of the tiles (see "CPUID—CPU Identification" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for details).

The matrix multiplications in the TMUL instruction set compute C[M][N] += A[M][K] * B[K][N]. The M, N, and K values will cause the TMUL instruction set to generate a #UD exception if the dimensions do not match for matrix multiply or do not match the palette.

In Figure 18-2, the number of rows in tile B matches the K dimension in the matrix multiplication pseudocode. K dimensions smaller than that enumerated in the TMUL grid are also possible and any additional computation the TMUL unit can support will not affect the result.

The number of elements specified by colsb of the B matrix is also less than or equal to tmul_maxn. Any remaining values beyond that specified by the metadata will be set to zero.

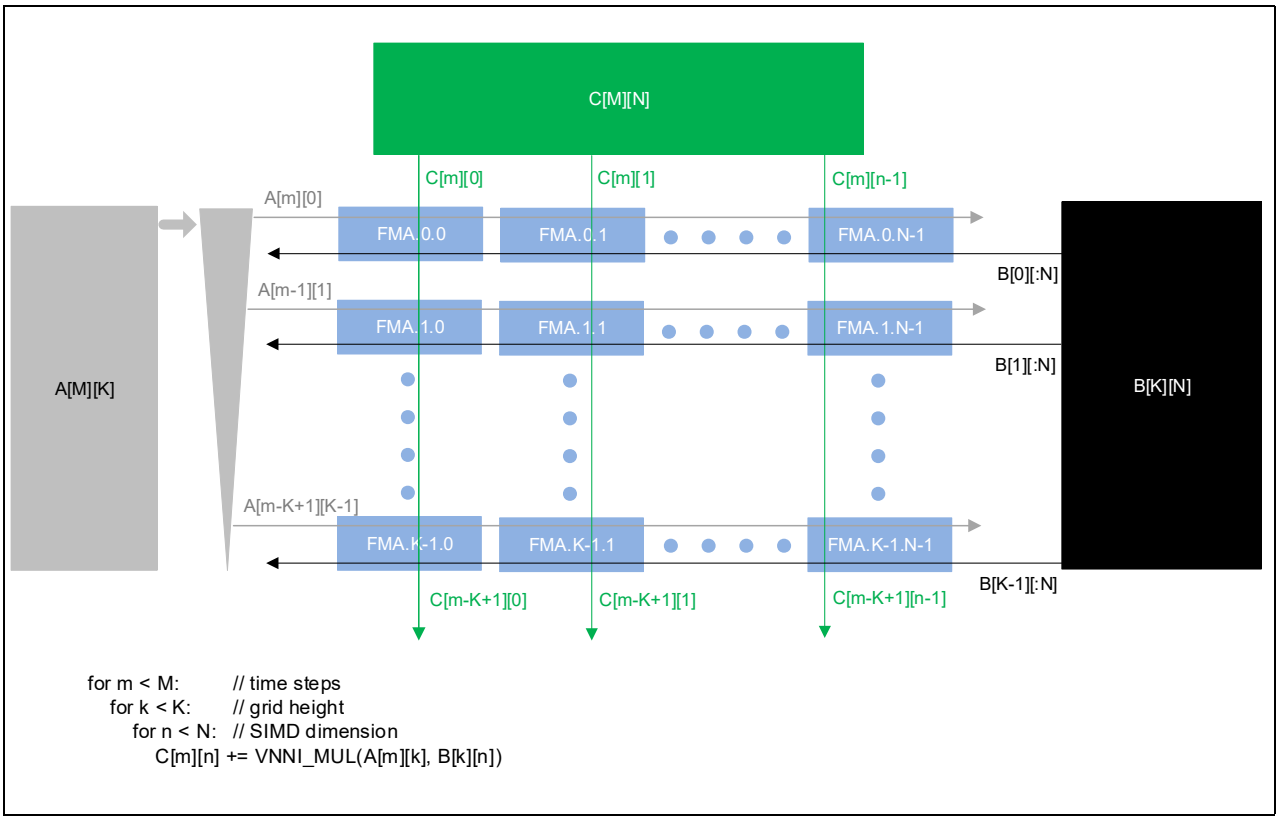


Figure 18-2. The TMUL Unit

The XSAVE feature set supports context management of the new state defined for Intel AMX. This support is described in Section 18.2.

18.1.1 Tile Architecture Details

The supported parameters for the tile architecture are reported via CPUID; this includes information about how the number of tile registers (max_names) can be configured (the palette). Configuring the tile architecture is intended to be done once when entering a region of tile code using the LDTILECFG instruction specifying the selected palette and describing in detail the configuration for each tile. Incorrect assignments will result in a General Protection fault (#GP). Successful LDTILECFG initializes (zeroes) TILEDATA.

Exiting a tile region is done with the TILERELEASE instruction. It takes no parameters and invalidates all tiles (indicating that the data no longer needs any saving or restoring). Essentially, it is an optimization of LDTILECFG with an implicit palette of 0.

For applications that execute consecutive Intel AMX regions with differing configurations, TILERELEASE is not required between them since the second LDTILECFG will clear all the data while loading the new configuration. There is no instruction set support for automatic nesting of tile regions, though with sufficient effort software can accomplish this by saving and restoring TILEDATA and TILECFG either through the XSAVE architecture or the Intel AMX instructions.

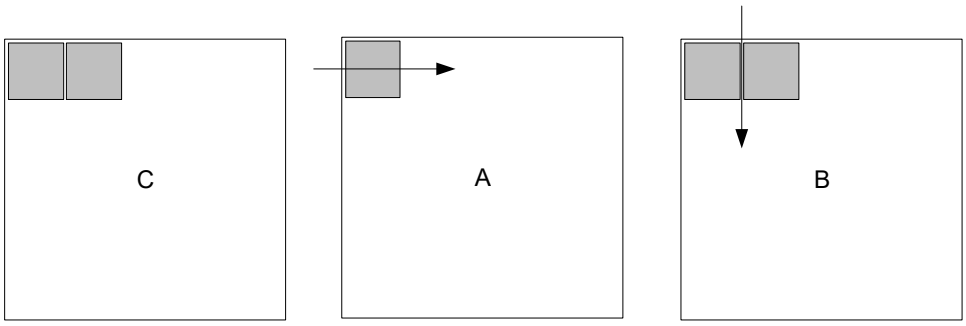
The tile architecture boots in its INIT state, with TILECFG and TILEDATA set to zero. A successfully executing LDTILECFG instruction to a non-zero palette sets the TILES_CONFIGURED=1, indicating the TILECFG is not in the INIT state. The TILERELEASE instruction sets TILES_CONFIGURED = 0 and initializes (zeroes) TILEDATA.

To facilitate handling of tile configuration data, there is a STTILECFG instruction. If the tile configuration is in the INIT state (TILES_CONFIGURED == 0), then STTILECFG will write 64 bytes of zeros. Otherwise STTILECFG will store the TILECFG to memory in the format used by LDTILECFG.

18.1.2 TMUL Architecture Details

The supported parameters for the TMUL architecture are reported via CPUID; see "CPUID—CPU Identification" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for details. These parameters include a maximum height (**tmul_maxk**) and a maximum SIMD dimension (**tmul_maxn**). The metadata that accompanies the srcdest, src1, and src2 tiles to the TMUL unit will be dynamically checked to see that they match the TMUL unit support for the data type and match the requirements of a meaningful matrix multiplication.

Figure 18-3 shows an example of the inner loop of an algorithm of using the TMUL architecture to compute a matrix multiplication. In this example, we use two result tiles, tmm0 and tmm1, from matrix C to accumulate the intermediate results. One tile from the A matrix (tmm2) is re-used twice as we multiply it by two tiles from the B matrix. The algorithm then advances pointers to load a new A tile and two new B tiles from the directions indicated by the arrows. An outer loop, not shown, adjusts the pointers for the C tiles.




```
   LDTILECFG [rax]
// assume some outer loops driving the cache tiling (not shown)
{
   TILELOADD tmm0, [rsi+rdi]     // srcdst, RSI points to C, RDI is strided value
   TILELOADD tmm1, [rsi+rdi+N]   // second tile of C, unrolling in SIMD dimension N
   MOV r14, 0
LOOP:
   TILELOADD tmm2, [r8+r9]       // src2 is strided load of A, reused for 2 TMUL instr.
   TILELOADD tmm3, [r10+r11]     // src1 is strided load of B
   TDPBUSD   tmm0, tmm2, tmm3  // update left tile of C
   TILELOADD tmm3, [r10+r11+N]   // src1 loaded with B from next rightmost tile
   TDPBUSD   tmm1, tmm2, tmm3  // update right tile of C
   ADD r8, K                 // update pointers by constants known outside of loop
   ADD r10, K*r11
   ADD r14, K
   CMP r14, LIMIT
   JNE LOOP

   TILESTORED [rsi+rdi],   tmm0  // update the C matrix in memory
   TILESTORED [rsi+rdi+M], tmm1
} // end of outer loop

   TILERELEASE         // return tiles to INIT state
```

Figure 18-3. Matrix Multiply C+= A*B

18.1.3 Handling of Tile Row and Column Limits

Intel AMX operations will zero any rows and any columns beyond the dimensions specified by TILECFG. Tile operations will zero the data beyond the configured number of column bytes as each row is written. For example, with 64-byte rows and a tile configured with 10 rows and 48 columns, an operation writing dword elements would write

each of the first 10 rows with 48 bytes of output/result data and zero the remaining 16 bytes in each row. Tile operations also fully zero any rows after the first 10 configured rows. When using a 1 KByte tile with 64-byte rows, there would be 16 rows, so in this example, the last 6 rows would also be zeroed.

Intel AMX instructions will always obey the metadata on reads and the zeroing rules on writes, and so a subsequent XSAVE would see zeros in the appropriate locations. Tiles that are not written by Intel AMX instructions between XRSTOR and XSAVE will write back with the same image they were loaded with regardless of the value of TILECFG.

18.1.4 Exceptions and Interrupts

Tile instructions are restartable so that operations that access strided memory can restart after page faults. To support restarting instructions after these events, the instructions store information in the **TILECFG.start_row** register. TILECFG.start_row indicates the row that should be used for restart; i.e., it indicates **next row after** the rows that have already been successfully loaded (on a TILELOAD) or written to memory (on a TILESTORE) and prevents repeating work that was successfully done.

The TMUL instruction set is not sensitive to the TILECFG.start_row value; this is due to there not being TMUL instructions with memory operands or any restartable faults.

18.2 RECOMMENDATIONS FOR SYSTEM SOFTWARE

Intel AMX is an XSAVE-enabled feature, meaning that it requires use of the XSAVE feature set for their enabling. Specifically, Intel AMX instructions and state are available only if system software has set CR4.OSXSAVE and also set XCR0[18:17] to 11B. In addition, use of Intel AMX instructions is disabled if system software has used extended feature disable (XFD) and set either IA32_XFD[17] or IA32_XFD[18] to 1. See Chapter 13, “Managing State Using the XSAVE Feature Set,” for more details.

NOTE

The first processors implementing Intel AMX will support setting IA32_XFD[18] but not IA32_XFD[17].

Once Intel AMX has been enabled, system software can disable it by clearing XCR0[18:17], by clearing CR4.OSXSAVE, or by setting either IA32_XFD[17] or IA32_XFD[18]. Before doing so, system software should first initialize AMX state (e.g., by executing TILERELEASE); maintaining AMX state in a non-initialized state may have negative power and performance implications and will prevent the execution of In-Field Scan tests. In addition, software should not rely on the state of the tile data after setting IA32_XFD[17] or IA32_XFD[18]; software should always reload or reinitialize the tile data after clearing IA32_XFD[17] and IA32_XFD[18].

System software should not use XFD to implement a “lazy restore” approach to management of the TILEDATA state component. This approach will **not** operate correctly for a variety of reasons. One is that the LDTILECFG and TILERELEASE instructions initialize TILEDATA and do not cause an #NM exception. Another is that an execution of XSAVE, XSAVEC, XSAVEOPT, or XSAVES by a user thread will save TILEDATA as initialized instead of the data expected by the user thread.

18.3 IMPLEMENTATION PARAMETERS

The parameters are reported via CPUID leaf 1DH. Index 0 reports all zeros for all fields.

```
define palette_table[id]:
    uint16_t total_tile_bytes
    uint16_t bytes_per_tile
    uint16_t bytes_per_row
    uint16_t max_names
    uint16_t max_rows
```

The tile parameters are set by LDTILECFG or XRSTOR* of TILECFG:

```
define tile[tid]:
    byte rows
    word colsb // bytes_per_row
    bool valid
```

18.4 HELPER FUNCTIONS

The helper functions used in Intel AMX instructions are defined below.

```
define write_row_and_zero(treg, r, data, nbytes):
   for j in 0 ...nbytes-1:
       treg.row[r].byte[j] := data.byte[j]

   // zero the rest of the row
   for j in nbytes ... palette_table[tilecfg.palette_id].bytes_per_row-1:
      treg.row[r].byte[j] := 0
```

```
define zero_upper_rows(treg, r):
   for i in r ... palette_table[tilecfg.palette_id].max_rows-1:
      for j in 0 ... palette_table[tilecfg.palette_id].bytes_per_row-1:
            treg.row[i].byte[j] := 0
```

```
define zero_tilecfg_start():
    tilecfg.start_row :=0
```

```
define zero_all_tile_data():
  if XCR0[TILEDATA]:
      b := CPUID(0xD,TILEDATA).EAX // size of feature
      for j in 0 ...b:
            TILEDATA.byte[j] := 0
```

```
define xcr0_supports_palette(palette_id):
  if palette_id == 0:
     return 1
  elif palette_id == 1:
     if XCR0[TILECFG] and XCR0[TILEDATA]:
        return 1
   return 0
```

5. Updates to Chapter 2, Volume 2A

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

Changes to this chapter:

- Updated Section 2.1.1, "Instruction Prefixes," to clarify when branch hint prefixes are used and removed unnecessary text regarding these prefixes. Updated the footnote in this section associated with branch hint prefixes to direct readers to the Intel® 64 and IA-32 Architectures Optimization Reference Manual.
- Updated Section 2.2.1, "REX Prefixes," to provide additional details about the REX prefix.

CHAPTER 2
INSTRUCTION FORMAT

This chapter describes the instruction format for all Intel 64 and IA-32 processors. The instruction format for protected mode, real-address mode and virtual-8086 mode is described in Section 2.1. Increments provided for IA-32e mode and its sub-modes are described in Section 2.2.

2.1 INSTRUCTION FORMAT FOR PROTECTED MODE, REAL-ADDRESS MODE, AND VIRTUAL-8086 MODE

The Intel 64 and IA-32 architectures instruction encodings are subsets of the format shown in Figure 2-1. Instructions consist of optional instruction prefixes (in any order), primary opcode bytes (up to three bytes), an addressing-form specifier (if required) consisting of the ModR/M byte and sometimes the SIB (Scale-Index-Base) byte, a displacement (if required), and an immediate data field (if required).

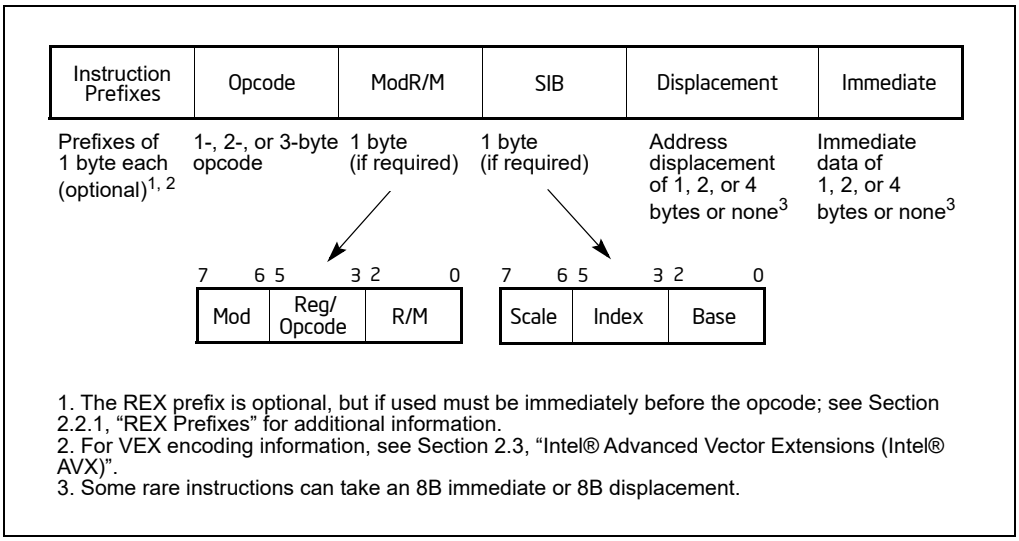


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

2.1.1 Instruction Prefixes

Instruction prefixes are divided into four groups, each with a set of allowable prefix codes. For each instruction, it is only useful to include up to one prefix code from each of the four groups (Groups 1, 2, 3, 4). Groups 1 through 4 may be placed in any order relative to each other.

- Group 1
  - — Lock and repeat prefixes:
    - LOCK prefix is encoded using F0H.
    - REPNE/REPNZ prefix is encoded using F2H. Repeat-Not-Zero prefix applies only to string and input/output instructions. (F2H is also used as a mandatory prefix for some instructions.)
    - REP or REPE/REPZ is encoded using F3H. The repeat prefix applies only to string and input/output instructions. (F3H is also used as a mandatory prefix for some instructions.)

  - BND prefix is encoded using F2H if the following conditions are true:
    - CPUID.(EAX=07H, ECX=0):EBX.MPX[bit 14] is set.
    - BNDCFGU.EN and/or IA32_BNDCFGS.EN is set.
    - When the F2 prefix precedes a near CALL, a near RET, a near JMP, a short Jcc, or a near Jcc instruction (see Appendix E, "Intel® Memory Protection Extensions," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1).
- Group 2
  - Segment override prefixes:
    - 2EH—CS segment override (use with any branch instruction is reserved).
    - 36H—SS segment override prefix (use with any branch instruction is reserved).
    - 3EH—DS segment override prefix (use with any branch instruction is reserved).
    - 26H—ES segment override prefix (use with any branch instruction is reserved).
    - 64H—FS segment override prefix (use with any branch instruction is reserved).
    - 65H—GS segment override prefix (use with any branch instruction is reserved).
  - Branch hints[1]:
    - 2EH—Branch not taken (used only with J*cc* instructions).
    - 3EH—Branch taken (used only with J*cc* instructions).
- Group 3
    - Operand-size override prefix is encoded using 66H (66H is also used as a mandatory prefix for some instructions).
- Group 4
    - 67H—Address-size override prefix.

The LOCK prefix (F0H) forces an operation that ensures exclusive use of shared memory in a multiprocessor environment. See "LOCK—Assert LOCK# Signal Prefix" in Chapter 3, "Instruction Set Reference, A-L," for a description of this prefix.

Repeat prefixes (F2H, F3H) cause an instruction to be repeated for each element of a string. Use these prefixes only with string and I/O instructions (MOVS, CMPS, SCAS, LODS, STOS, INS, and OUTS). Use of repeat prefixes and/or undefined opcodes with other Intel 64 or IA-32 instructions is reserved; such use may cause unpredictable behavior.

Some instructions may use F2H or F3H as a mandatory prefix to express distinct functionality.

Branch hint prefixes (2EH, 3EH) allow a program to give a hint to the processor about the most likely code path for a branch when used on conditional branch instructions (*Jcc*).

The operand-size override prefix allows a program to switch between 16- and 32-bit operand sizes. Either size can be the default; use of the prefix selects the non-default size.

Some SSE2/SSE3/SSSE3/SSE4 instructions and instructions using a three-byte sequence of primary opcode bytes may use 66H as a mandatory prefix to express distinct functionality.

Other use of the 66H prefix is reserved; such use may cause unpredictable behavior.

The address-size override prefix (67H) allows programs to switch between 16- and 32-bit addressing. Either size can be the default; the prefix selects the non-default size. Using this prefix and/or other undefined opcodes when operands for the instruction do not reside in memory is reserved; such use may cause unpredictable behavior.

1. Microarchitectural behavior varies; refer to the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

2.1.2 Opcodes

A primary opcode can be 1, 2, or 3 bytes in length. An additional 3-bit opcode field is sometimes encoded in the ModR/M byte. Smaller fields can be defined within the primary opcode. Such fields define the direction of operation, size of displacements, register encoding, condition codes, or sign extension. Encoding fields used by an opcode vary depending on the class of operation.

Two-byte opcode formats for general-purpose and SIMD instructions consist of one of the following:

- An escape opcode byte 0FH as the primary opcode and a second opcode byte.
- A mandatory prefix (66H, F2H, or F3H), an escape opcode byte, and a second opcode byte (same as previous bullet).

For example, CVTDQ2PD consists of the following sequence: F3 0F E6. The first byte is a mandatory prefix (it is not considered as a repeat prefix).

Three-byte opcode formats for general-purpose and SIMD instructions consist of one of the following:

- An escape opcode byte 0FH as the primary opcode, plus two additional opcode bytes.
- A mandatory prefix (66H, F2H, or F3H), an escape opcode byte, plus two additional opcode bytes (same as previous bullet).

For example, PHADDW for XMM registers consists of the following sequence: 66 0F 38 01. The first byte is the mandatory prefix.

Valid opcode expressions are defined in Appendix A and Appendix B.

2.1.3 ModR/M and SIB Bytes

Many instructions that refer to an operand in memory have an addressing-form specifier byte (called the ModR/M byte) following the primary opcode. The ModR/M byte contains three fields of information:

- The *mod* field combines with the r/m field to form 32 possible values: eight registers and 24 addressing modes.
- The *reg/opcode* field specifies either a register number or three more bits of opcode information. The purpose of the reg/opcode field is specified in the primary opcode.
- The *r/m* field can specify a register as an operand or it can be combined with the mod field to encode an addressing mode. Sometimes, certain combinations of the *mod* field and the *r/m* field are used to express opcode information for some instructions.

Certain encodings of the ModR/M byte require a second addressing byte (the SIB byte). The base-plus-index and scale-plus-index forms of 32-bit addressing require the SIB byte. The SIB byte includes the following fields:

- The *scale* field specifies the scale factor.
- The *index* field specifies the register number of the index register.
- The *base* field specifies the register number of the base register.

See Section 2.1.5 for the encodings of the ModR/M and SIB bytes.

2.1.4 Displacement and Immediate Bytes

Some addressing forms include a displacement immediately following the ModR/M byte (or the SIB byte if one is present). If a displacement is required, it can be 1, 2, or 4 bytes.

If an instruction specifies an immediate operand, the operand always follows any displacement bytes. An immediate operand can be 1, 2 or 4 bytes.

2.1.5 Addressing-Mode Encoding of ModR/M and SIB Bytes

The values and corresponding addressing forms of the ModR/M and SIB bytes are shown in Table 2-1 through Table 2-3: 16-bit addressing forms specified by the ModR/M byte are in Table 2-1 and 32-bit addressing forms are in Table 2-2. Table 2-3 shows 32-bit addressing forms specified by the SIB byte. In cases where the reg/opcode field in the ModR/M byte represents an extended opcode, valid encodings are shown in Appendix B.

In Table 2-1 and Table 2-2, the Effective Address column lists 32 effective addresses that can be assigned to the first operand of an instruction by using the Mod and R/M fields of the ModR/M byte. The first 24 options provide ways of specifying a memory location; the last eight (Mod = 11B) provide ways of specifying general-purpose, MMX technology and XMM registers.

The Mod and R/M columns in Table 2-1 and Table 2-2 give the binary encodings of the Mod and R/M fields required to obtain the effective address listed in the first column. For example: see the row indicated by Mod = 11B, R/M = 000B. The row identifies the general-purpose registers EAX, AX or AL; MMX technology register MM0; or XMM register XMM0. The register used is determined by the opcode byte and the operand-size attribute.

Now look at the seventh row in either table (labeled "REG ="). This row specifies the use of the 3-bit Reg/Opcode field when the field is used to give the location of a second operand. The second operand must be a general-purpose, MMX technology, or XMM register. Rows one through five list the registers that may correspond to the value in the table. Again, the register used is determined by the opcode byte along with the operand-size attribute.

If the instruction does not require a second operand, then the Reg/Opcode field may be used as an opcode extension. This use is represented by the sixth row in the tables (labeled "/digit (Opcode)"). Note that values in row six are represented in decimal form.

The body of Table 2-1 and Table 2-2 (under the label "Value of ModR/M Byte (in Hexadecimal)") contains a 32 by 8 array that presents all of 256 values of the ModR/M byte (in hexadecimal). Bits 3, 4, and 5 are specified by the column of the table in which a byte resides. The row specifies bits 0, 1, and 2; and bits 6 and 7. The figure below demonstrates interpretation of one table value.

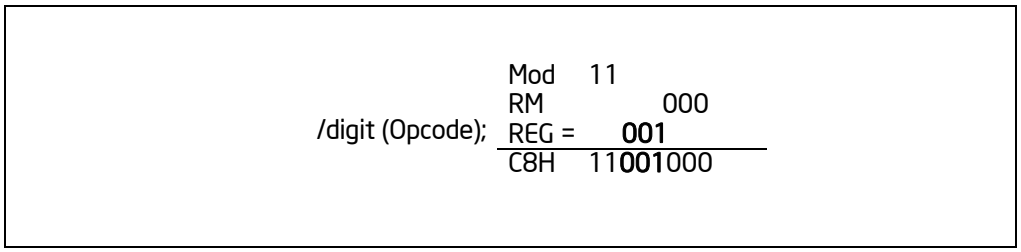


Figure 2-2. Table Interpretation of ModR/M Byte (C8H)

Table 2-1. 16-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP[1] EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[BX+SI] [BX+DI] [BP+SI] [BP+DI] [SI] [DI] disp16[2] [BX]	00	000 001 010 011 100 101 110 111	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	10 11 12 13 14 15 16 17	18 19 1A 1B 1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B 2C 2D 2E 2F	30 31 32 33 34 35 36 37	38 39 3A 3B 3C 3D 3E 3F
[BX+SI]+disp8[3] [BX+DI]+disp8 [BP+SI]+disp8 [BP+DI]+disp8 [SI]+disp8 [DI]+disp8 [BP]+disp8 [BX]+disp8	01	000 001 010 011 100 101 110 111	40 41 42 43 44 45 46 47	48 49 4A 4B 4C 4D 4E 4F	50 51 52 53 54 55 56 57	58 59 5A 5B 5C 5D 5E 5F	60 61 62 63 64 65 66 67	68 69 6A 6B 6C 6D 6E 6F	70 71 72 73 74 75 76 77	78 79 7A 7B 7C 7D 7E 7F
[BX+SI]+disp16 [BX+DI]+disp16 [BP+SI]+disp16 [BP+DI]+disp16 [SI]+disp16 [DI]+disp16 [BP]+disp16 [BX]+disp16	10	000 001 010 011 100 101 110 111	80 81 82 83 84 85 86 87	88 89 8A 8B 8C 8D 8E 8F	90 91 92 93 94 95 96 97	98 99 9A 9B 9C 9D 9E 9F	A0 A1 A2 A3 A4 A5 A6 A7	A8 A9 AA AB AC AD AE AF	B0 B1 B2 B3 B4 B5 B6 B7	B8 B9 BA BB BC BD BE BF
EAX/AX/AL/MM0/XMM0 ECX/CX/CL/MM1/XMM1 EDX/DX/DL/MM2/XMM2 EBX/BX/BL/MM3/XMM3 ESP/SP/AHMM4/XMM4 EBP/BP/CH/MM5/XMM5 ESI/SI/DH/MM6/XMM6 EDI/DI/BH/MM7/XMM7	11	000 001 010 011 100 101 110 111	C0 C1 C2 C3 C4 C5 C6 C7	C8 C9 CA CB CC CD CE CF	D0 D1 D2 D3 D4 D5 D6 D7	D8 D9 DA DB DC DD DE DF	E0 E1 E2 E3 E4 E5 E6 E7	E8 E9 EA EB EC ED EE EF	F0 F1 F2 F3 F4 F5 F6 F7	F8 F9 FA FB FC FD FE FF

NOTES:

1. The default segment register is SS for the effective addresses containing a BP index, DS for other effective addresses.
2. The disp16 nomenclature denotes a 16-bit displacement that follows the ModR/M byte and that is added to the index.
3. The disp8 nomenclature denotes an 8-bit displacement that follows the ModR/M byte and that is sign-extended and added to the index.

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[EAX] [ECX] [EDX] [EBX] [--][--][1] disp32[2] [ESI] [EDI]	00	000 001 010 011 100 101 110 111	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	10 11 12 13 14 15 16 17	18 19 1A 1B 1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B 2C 2D 2E 2F	30 31 32 33 34 35 36 37	38 39 3A 3B 3C 3D 3E 3F
[EAX]+disp8[3] [ECX]+disp8 [EDX]+disp8 [EBX]+disp8 [--][--]+disp8 [EBP]+disp8 [ESI]+disp8 [EDI]+disp8	01	000 001 010 011 100 101 110 111	40 41 42 43 44 45 46 47	48 49 4A 4B 4C 4D 4E 4F	50 51 52 53 54 55 56 57	58 59 5A 5B 5C 5D 5E 5F	60 61 62 63 64 65 66 67	68 69 6A 6B 6C 6D 6E 6F	70 71 72 73 74 75 76 77	78 79 7A 7B 7C 7D 7E 7F
[EAX]+disp32 [ECX]+disp32 [EDX]+disp32 [EBX]+disp32 [--][--]+disp32 [EBP]+disp32 [ESI]+disp32 [EDI]+disp32	10	000 001 010 011 100 101 110 111	80 81 82 83 84 85 86 87	88 89 8A 8B 8C 8D 8E 8F	90 91 92 93 94 95 96 97	98 99 9A 9B 9C 9D 9E 9F	A0 A1 A2 A3 A4 A5 A6 A7	A8 A9 AA AB AC AD AE AF	B0 B1 B2 B3 B4 B5 B6 B7	B8 B9 BA BB BC BD BE BF
EAX/AX/AL/MM0/XMM0 ECX/CX/CL/MM/XMM1 EDX/DX/DL/MM2/XMM2 EBX/BX/BL/MM3/XMM3 ESP/SP/AH/MM4/XMM4 EBP/BP/CH/MM5/XMM5 ESI/SI/DH/MM6/XMM6 EDI/DI/BH/MM7/XMM7	11	000 001 010 011 100 101 110 111	C0 C1 C2 C3 C4 C5 C6 C7	C8 C9 CA CB CC CD CE CF	D0 D1 D2 D3 D4 D5 D6 D7	D8 D9 DA DB DC DD DE DF	E0 E1 E2 E3 E4 E5 E6 E7	E8 E9 EA EB EC ED EE EF	F0 F1 F2 F3 F4 F5 F6 F7	F8 F9 FA FB FC FD FE FF

NOTES:

1. The [--][--] nomenclature means a SIB follows the ModR/M byte.
2. The disp32 nomenclature denotes a 32-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is added to the index.
3. The disp8 nomenclature denotes an 8-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is sign-extended and added to the index.

Table 2-3 is organized to give 256 possible values of the SIB byte (in hexadecimal). General purpose registers used as a base are indicated across the top of the table, along with corresponding values for the SIB byte's base field. Table rows in the body of the table indicate the register used as the index (SIB byte bits 3, 4, and 5) and the scaling factor (determined by SIB byte bits 6 and 7).

Table 2-3. 32-Bit Addressing Forms with the SIB Byte

r32 (In decimal) Base = (In binary) Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
Scaled Index	**SS**	**Index**	**Value of SIB Byte (in Hexadecimal)**							
[EAX] [ECX] [EDX] [EBX] none [EBP] [ESI] [EDI]	00	000 001 010 011 100 101 110 111	00 08 10 18 20 28 30 38	01 09 11 19 21 29 31 39	02 0A 12 1A 22 2A 32 3A	03 0B 13 1B 23 2B 33 3B	04 0C 14 1C 24 2C 34 3C	05 0D 15 1D 25 2D 35 3D	06 0E 16 1E 26 2E 36 3E	07 0F 17 1F 27 2F 37 3F
[EAX*2] [ECX*2] [EDX*2] [EBX*2] none [EBP*2] [ESI*2] [EDI*2]	01	000 001 010 011 100 101 110 111	40 48 50 58 60 68 70 78	41 49 51 59 61 69 71 79	42 4A 52 5A 62 6A 72 7A	43 4B 53 5B 63 6B 73 7B	44 4C 54 5C 64 6C 74 7C	45 4D 55 5D 65 6D 75 7D	46 4E 56 5E 66 6E 76 7E	47 4F 57 5F 67 6F 77 7F
[EAX*4] [ECX*4] [EDX*4] [EBX*4] none [EBP*4] [ESI*4] [EDI*4]	10	000 001 010 011 100 101 110 111	80 88 90 98 A0 A8 B0 B8	81 89 91 99 A1 A9 B1 B9	82 8A 92 9A A2 AA B2 BA	83 8B 93 9B A3 AB B3 BB	84 8C 94 9C A4 AC B4 BC	85 8D 95 9D A5 AD B5 BD	86 8E 96 9E A6 AE B6 BE	87 8F 97 9F A7 AF B7 BF
[EAX*8] [ECX*8] [EDX*8] [EBX*8] none [EBP*8] [ESI*8] [EDI*8]	11	000 001 010 011 100 101 110 111	C0 C8 D0 D8 E0 E8 F0 F8	C1 C9 D1 D9 E1 E9 F1 F9	C2 CA D2 DA E2 EA F2 FA	C3 CB D3 DB E3 EB F3 FB	C4 CC D4 DC E4 EC F4 FC	C5 CD D5 DD E5 ED F5 FD	C6 CE D6 DE E6 EE F6 FE	C7 CF D7 DF E7 EF F7 FF

NOTES:

1. The [*] nomenclature means a disp32 with no base if the MOD is 00B. Otherwise, [*] means disp8 or disp32 + [EBP]. This provides the following address modes:

MOD bits	Effective Address
00	[scaled index] + disp32
01	[scaled index] + disp8 + [EBP]
10	[scaled index] + disp32 + [EBP]

2.2 IA-32E MODE

IA-32e mode has two sub-modes. These are:

- **Compatibility Mode.** Enables a 64-bit operating system to run most legacy protected mode software unmodified.
- **64-Bit Mode.** Enables a 64-bit operating system to run applications written to access 64-bit address space.

2.2.1 REX Prefixes

REX prefixes are instruction-prefix bytes used in 64-bit mode. They do the following:

- Specify GPRs and SSE registers.

- Specify 64-bit operand size.
- Specify extended control registers.

Not all instructions require a REX prefix in 64-bit mode. A REX prefix is necessary only if an instruction references one of the extended registers or one of the byte registers SPL, BPL, SIL, DIL; or uses a 64-bit operand. A REX prefix is ignored, as are its individual bits, when it is not needed for an instruction or when it does not immediately precede the opcode byte or the escape opcode byte (0FH) of an instruction for which it is needed. This has the implication that only one REX prefix, properly located, can affect an instruction.

When a REX prefix is used in conjunction with an instruction containing a mandatory prefix, the mandatory prefix must come before the REX so the REX prefix can immediately precede the opcode or the escape byte. For example, CVTDQ2PD with a REX prefix should have REX placed between F3 and 0F E6. Other placements are ignored. The instruction-size limit of 15 bytes still applies to instructions with a REX prefix. See Figure 2-3.

Legacy Prefixes	REX Prefix	Opcode	ModR/M	SIB	Displacement	Immediate
Grp 1, Grp 2, Grp 3, Grp 4 (optional)	(optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes	Immediate data of 1, 2, or 4 bytes or none

Figure 2-3. Prefix Ordering in 64-bit Mode

2.2.1.1 Encoding

Intel 64 and IA-32 instruction formats specify up to three registers by using 3-bit fields in the encoding, depending on the format:

- ModR/M: the reg and r/m fields of the ModR/M byte.
- ModR/M with SIB: the reg field of the ModR/M byte, the base and index fields of the SIB (scale, index, base) byte.
- Instructions without ModR/M: the reg field of the opcode.

In 64-bit mode, these formats do not change. Bits needed to define fields in the 64-bit context are provided by the addition of REX prefixes.

2.2.1.2 More on REX Prefix Fields

REX prefixes are a set of 16 opcodes that span one row of the opcode map and occupy entries 40H to 4FH. These opcodes represent valid instructions (INC or DEC) in IA-32 operating modes and in compatibility mode. In 64-bit mode, the same opcodes represent the instruction prefix REX and are not treated as individual instructions.

The single-byte-opcode forms of the INC/DEC instructions are not available in 64-bit mode. INC/DEC functionality is still available using ModR/M forms of the same instructions (opcodes FF/0 and FF/1).

See Table 2-4 for a summary of the REX prefix format. Figure 2-4 though Figure 2-7 show examples of REX prefix fields in use. Some combinations of REX prefix fields are invalid. In such cases, the prefix is ignored. Some additional information follows:

- Setting REX.W can be used to determine the operand size but does not solely determine operand width. Like the 66H size prefix, 64-bit operand size override has no effect on byte-specific operations.
- For non-byte operations: if a 66H prefix is used with prefix (REX.W = 1), 66H is ignored.
- If a 66H override is used with REX and REX.W = 0, the operand size is 16 bits.
- REX.R modifies the ModR/M reg field when that field encodes a GPR, SSE, control or debug register. REX.R is ignored when ModR/M specifies other registers or defines an extended opcode.
- REX.X bit modifies the SIB index field.

- REX.B either modifies the base in the ModR/M r/m field or SIB base field; or it modifies the opcode reg field used for accessing GPRs.

Table 2-4. REX Prefix Fields [BITS: 0100WRXB]

Field Name	Bit Position	Definition
-	7:4	0100
W	3	0 = Operand size determined by CS.D
		1 = 64 Bit Operand Size
R	2	Extension of the ModR/M reg field
X	1	Extension of the SIB index field
B	0	Extension of the ModR/M r/m field, SIB base field, or Opcode reg field

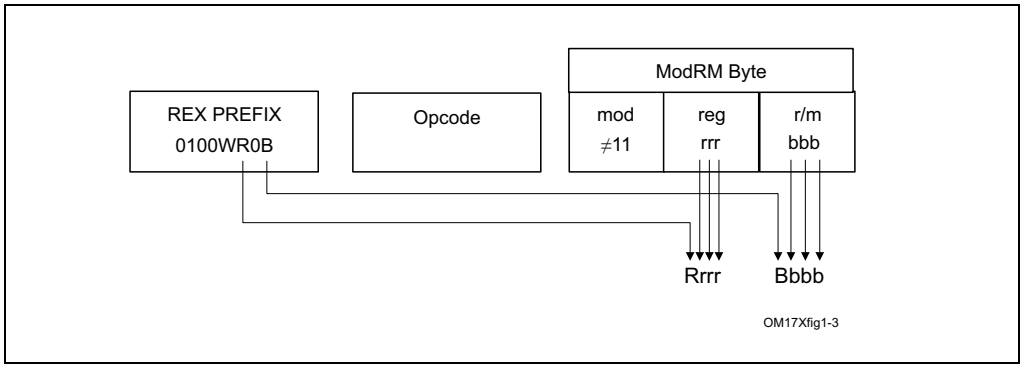


Figure 2-4. Memory Addressing Without an SIB Byte; REX.X Not Used

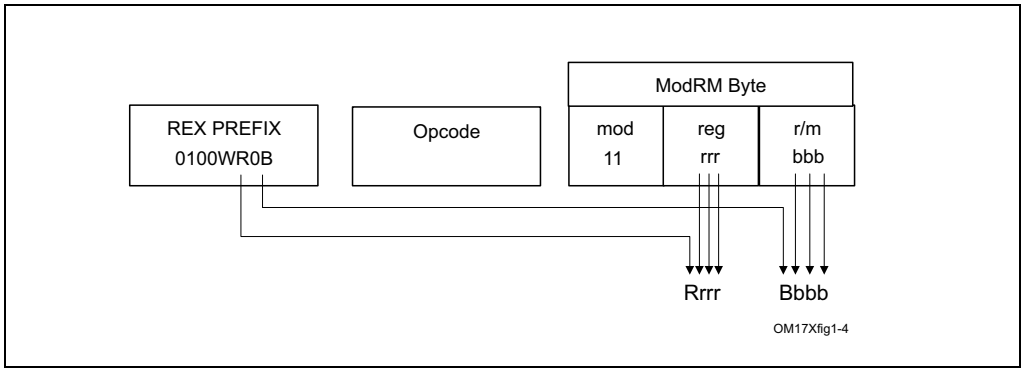


Figure 2-5. Register-Register Addressing (No Memory Operand); REX.X Not Used

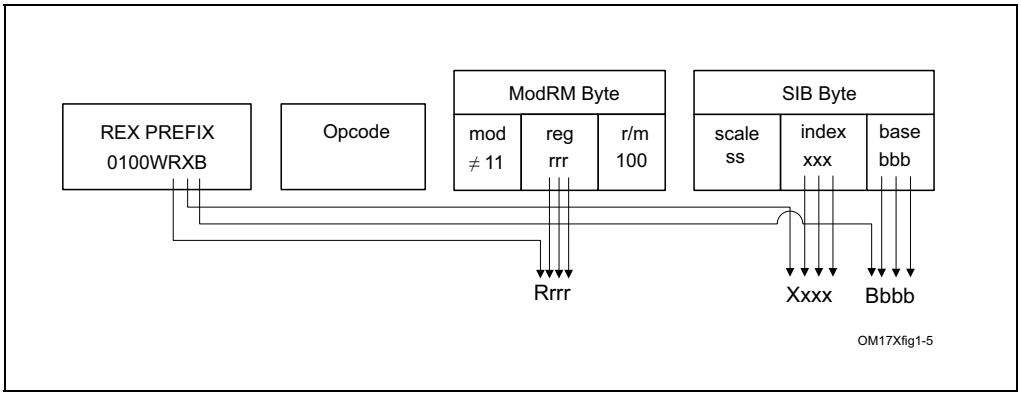


Figure 2-6. Memory Addressing With a SIB Byte

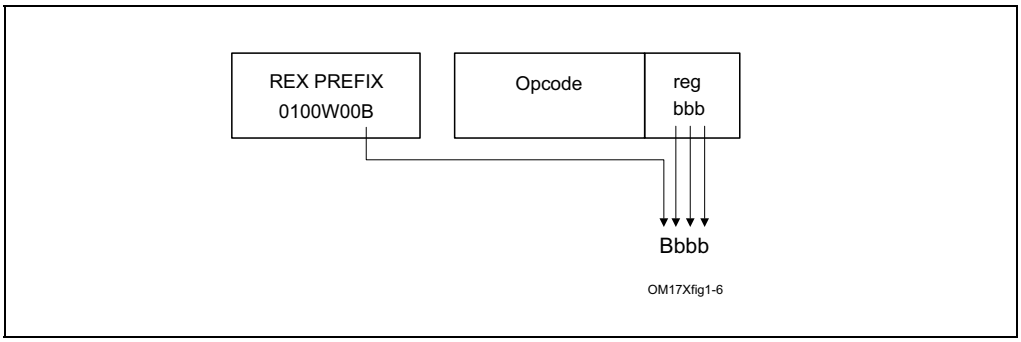


Figure 2-7. Register Operand Coded in Opcode Byte; REX.X & REX.R Not Used

In the IA-32 architecture, byte registers (AH, AL, BH, BL, CH, CL, DH, and DL) are encoded in the ModR/M byte's reg field, the r/m field or the opcode reg field as registers 0 through 7. REX prefixes provide an additional addressing capability for byte-registers that makes the least-significant byte of GPRs available for byte operations.

Certain combinations of the fields of the ModR/M byte and the SIB byte have special meaning for register encodings. For some combinations, fields expanded by the REX prefix are not decoded. Table 2-5 describes how each case behaves.

Table 2-5. Special Cases of REX Encodings

ModR/M or SIB	Sub-field Encodings	Compatibility Mode Operation	Compatibility Mode Implications	Additional Implications
ModR/M Byte	mod ? 11 r/m = b*100(ESP)	SIB byte present.	SIB byte required for ESP-based addressing.	REX prefix adds a fourth bit (b) which is not decoded (don't care). SIB byte also required for R12-based addressing.
ModR/M Byte	mod = 0 r/m = b*101(EBP)	Base register not used.	EBP without a displacement must be done using mod = 01 with displacement of 0.	REX prefix adds a fourth bit (b) which is not decoded (don't care). Using RBP or R13 without displacement must be done using mod = 01 with a displacement of 0.
SIB Byte	index = 0100(ESP)	Index register not used.	ESP cannot be used as an index register.	REX prefix adds a fourth bit (b) which is decoded. There are no additional implications. The expanded index field allows distinguishing RSP from R12, therefore R12 can be used as an index.

Table 2-5. Special Cases of REX Encodings (Contd.)

ModR/M or SIB	Sub-field Encodings	Compatibility Mode Operation	Compatibility Mode Implications	Additional Implications
SIB Byte	base = 0101(EBP)	Base register is unused if mod = 0.	Base register depends on mod encoding.	REX prefix adds a fourth bit (b) which is not decoded. This requires explicit displacement to be used with EBP/RBP or R13.

NOTES:

* Don't care about value of REX.B

2.2.1.3 Displacement

Addressing in 64-bit mode uses existing 32-bit ModR/M and SIB encodings. The ModR/M and SIB displacement sizes do not change. They remain 8 bits or 32 bits and are sign-extended to 64 bits.

2.2.1.4 Direct Memory-Offset MOVs

In 64-bit mode, direct memory-offset forms of the MOV instruction are extended to specify a 64-bit immediate absolute address. This address is called a moffset. No prefix is needed to specify this 64-bit memory offset. For these MOV instructions, the size of the memory offset follows the address-size default (64 bits in 64-bit mode). See Table 2-6.

Table 2-6. Direct Memory Offset Form of MOV

Opcode	Instruction
A0	MOV AL, moffset
A1	MOV EAX, moffset
A2	MOV moffset, AL
A3	MOV moffset, EAX

2.2.1.5 Immediates

In 64-bit mode, the typical size of immediate operands remains 32 bits. When the operand size is 64 bits, the processor sign-extends all immediates to 64 bits prior to their use.

Support for 64-bit immediate operands is accomplished by expanding the semantics of the existing move (MOV reg, imm16/32) instructions. These instructions (opcodes B8H – BFH) move 16-bits or 32-bits of immediate data (depending on the effective operand size) into a GPR. When the effective operand size is 64 bits, these instructions can be used to load an immediate into a GPR. A REX prefix is needed to override the 32-bit default operand size to a 64-bit operand size.

For example:

```
48 B8  8877665544332211  MOV RAX,1122334455667788H
```

2.2.1.6 RIP-Relative Addressing

A new addressing form, RIP-relative (relative instruction-pointer) addressing, is implemented in 64-bit mode. An effective address is formed by adding displacement to the 64-bit RIP of the next instruction.

In IA-32 architecture and compatibility mode, addressing relative to the instruction pointer is available only with control-transfer instructions. In 64-bit mode, instructions that use ModR/M addressing can use RIP-relative addressing. Without RIP-relative addressing, all ModR/M modes address memory relative to zero.

RIP-relative addressing allows specific ModR/M modes to address memory relative to the 64-bit RIP using a signed 32-bit displacement. This provides an offset range of ±2GB from the RIP. Table 2-7 shows the ModR/M and SIB encodings for RIP-relative addressing. Redundant forms of 32-bit displacement-addressing exist in the current ModR/M and SIB encodings. There is one ModR/M encoding and there are several SIB encodings. RIP-relative addressing is encoded using a redundant form.

In 64-bit mode, the ModR/M Disp32 (32-bit displacement) encoding is re-defined to be RIP+Disp32 rather than displacement-only. See Table 2-7.

Table 2-7. RIP-Relative Addressing

ModR/M and SIB Sub-field Encodings		Compatibility Mode Operation	64-bit Mode Operation	Additional Implications in 64-bit mode
ModR/M Byte	mod = 00	Disp32	RIP + Disp32	In 64-bit mode, if one wants to use a Disp32 without specifying a base register, one can use a SIB byte encoding (indicated by ModR/M.r/m=100) as described in the next row.
	r/m = 101 (none)			
SIB Byte	base = 101 (none)	If mod = 00, Disp32	Same as legacy	None
	index = 100 (none)			
	scale = 0, 1, 2, 4			

The ModR/M encoding for RIP-relative addressing does not depend on using a prefix. Specifically, the r/m bit field encoding of 101B (used to select RIP-relative addressing) is not affected by the REX prefix. For example, selecting R13 (REX.B = 1, r/m = 101B) with mod = 00B still results in RIP-relative addressing. The 4-bit r/m field of REX.B combined with ModR/M is not fully decoded. In order to address R13 with no displacement, software must encode R13 + 0 using a 1-byte displacement of zero.

RIP-relative addressing is enabled by 64-bit mode, not by a 64-bit address-size. The use of the address-size prefix does not disable RIP-relative addressing. The effect of the address-size prefix is to truncate and zero-extend the computed effective address to 32 bits.

2.2.1.7 Default 64-Bit Operand Size

In 64-bit mode, two groups of instructions have a default operand size of 64 bits (do not need a REX prefix for this operand size). These are:

- Near branches.
- All instructions, except far branches, that implicitly reference the RSP.

2.2.2 Additional Encodings for Control and Debug Registers

In 64-bit mode, more encodings for control and debug registers are available. The REX.R bit is used to modify the ModR/M reg field when that field encodes a control or debug register (see Table 2-4). These encodings enable the processor to address CR8-CR15 and DR8- DR15. An additional control register (CR8) is defined in 64-bit mode. CR8 becomes the Task Priority Register (TPR).

In the first implementation of IA-32e mode, CR9-CR15 and DR8-DR15 are not implemented. Any attempt to access unimplemented registers results in an invalid-opcode exception (#UD).

2.3 INTEL® ADVANCED VECTOR EXTENSIONS (INTEL® AVX)

Intel AVX instructions are encoded using an encoding scheme that combines prefix bytes, opcode extension field, operand encoding fields, and vector length encoding capability into a new prefix, referred to as VEX. In the VEX encoding scheme, the VEX prefix may be two or three bytes long, depending on the instruction semantics. Despite the two-byte or three-byte length of the VEX prefix, the VEX encoding format provides a more compact representation/packing of the components of encoding an instruction in Intel 64 architecture. The VEX encoding scheme also allows more headroom for future growth of Intel 64 architecture.

2.3.1 Instruction Format

Instruction encoding using VEX prefix provides several advantages:

- Instruction syntax support for three operands and up-to four operands when necessary. For example, the third source register used by VBLENDVPD is encoded using bits 7:4 of the immediate byte.
- Encoding support for vector length of 128 bits (using XMM registers) and 256 bits (using YMM registers).
- Encoding support for instruction syntax of non-destructive source operands.
- Elimination of escape opcode byte (0FH), SIMD prefix byte (66H, F2H, F3H) via a compact bit field representation within the VEX prefix.
- Elimination of the need to use REX prefix to encode the extended half of general-purpose register sets (R8-R15) for direct register access, memory addressing, or accessing XMM8-XMM15 (including YMM8-YMM15).
- Flexible and more compact bit fields are provided in the VEX prefix to retain the full functionality provided by REX prefix. REX.W, REX.X, REX.B functionalities are provided in the three-byte VEX prefix only because only a subset of SIMD instructions need them.
- Extensibility for future instruction extensions without significant instruction length increase.

Figure 2-8 shows the Intel 64 instruction encoding format with VEX prefix support. Legacy instruction without a VEX prefix is fully supported and unchanged. The use of VEX prefix in an Intel 64 instruction is optional, but a VEX prefix is required for Intel 64 instructions that operate on YMM registers or support three and four operand syntax. VEX prefix is not a constant-valued, "single-purpose" byte like 0FH, 66H, F2H, F3H in legacy SSE instructions. VEX prefix provides substantially richer capability than the REX prefix.

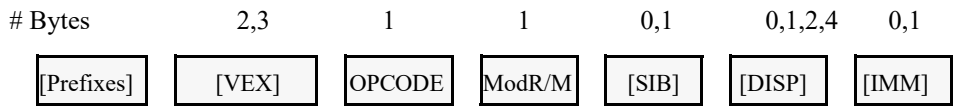


Figure 2-8. Instruction Encoding Format with VEX Prefix

2.3.2 VEX and the LOCK prefix

Any VEX-encoded instruction with a LOCK prefix preceding VEX will #UD.

2.3.3 VEX and the 66H, F2H, and F3H prefixes

Any VEX-encoded instruction with a 66H, F2H, or F3H prefix preceding VEX will #UD.

2.3.4 VEX and the REX prefix

Any VEX-encoded instruction with a REX prefix proceeding VEX will #UD.

2.3.5 The VEX Prefix

The VEX prefix is encoded in either the two-byte form (the first byte must be C5H) or in the three-byte form (the first byte must be C4H). The two-byte VEX is used mainly for 128-bit, scalar, and the most common 256-bit AVX instructions; while the three-byte VEX provides a compact replacement of REX and 3-byte opcode instructions (including AVX and FMA instructions). Beyond the first byte of the VEX prefix, it consists of a number of bit fields providing specific capability, they are shown in Figure 2-9.

The bit fields of the VEX prefix can be summarized by its functional purposes:

- Non-destructive source register encoding (applicable to three and four operand syntax): This is the first source operand in the instruction syntax. It is represented by the notation, VEX.vvvv. This field is encoded using 1's complement form (inverted form), i.e., XMM0/YMM0/R0 is encoded as 1111B, XMM15/YMM15/R15 is encoded as 0000B.
- Vector length encoding: This 1-bit field represented by the notation VEX.L. L= 0 means vector length is 128 bits wide, L=1 means 256 bit vector. The value of this field is written as VEX.128 or VEX.256 in this document to distinguish encoded values of other VEX bit fields.
- REX prefix functionality: Full REX prefix functionality is provided in the three-byte form of VEX prefix. However the VEX bit fields providing REX functionality are encoded using 1's complement form, i.e., XMM0/YMM0/R0 is encoded as 1111B, XMM15/YMM15/R15 is encoded as 0000B.
  - — Two-byte form of the VEX prefix only provides the equivalent functionality of REX.R, using 1's complement encoding. This is represented as VEX.R.
  - — Three-byte form of the VEX prefix provides REX.R, REX.X, REX.B functionality using 1's complement encoding and three dedicated bit fields represented as VEX.R, VEX.X, VEX.B.
  - — Three-byte form of the VEX prefix provides the functionality of REX.W only to specific instructions that need to override default 32-bit operand size for a general purpose register to 64-bit size in 64-bit mode. For those applicable instructions, VEX.W field provides the same functionality as REX.W. VEX.W field can provide completely different functionality for other instructions.

  Consequently, the use of REX prefix with VEX encoded instructions is not allowed. However, the intent of the REX prefix for expanding register set is reserved for future instruction set extensions using VEX prefix encoding format.
- Compaction of SIMD prefix: Legacy SSE instructions effectively use SIMD prefixes (66H, F2H, F3H) as an opcode extension field. VEX prefix encoding allows the functional capability of such legacy SSE instructions (operating on XMM registers, bits 255:128 of corresponding YMM unmodified) to be encoded using the VEX.pp field without the presence of any SIMD prefix. The VEX-encoded 128-bit instruction will zero-out bits 255:128 of the destination register. VEX-encoded instruction may have 128 bit vector length or 256 bits length.
- Compaction of two-byte and three-byte opcode: More recently introduced legacy SSE instructions employ two and three-byte opcode. The one or two leading bytes are: 0FH, and 0FH 3AH/0FH 38H. The one-byte escape (0FH) and two-byte escape (0FH 3AH, 0FH 38H) can also be interpreted as an opcode extension field. The VEX.mmmmm field provides compaction to allow many legacy instruction to be encoded without the constant byte sequence, 0FH, 0FH 3AH, 0FH 38H. These VEX-encoded instruction may have 128 bit vector length or 256 bits length.

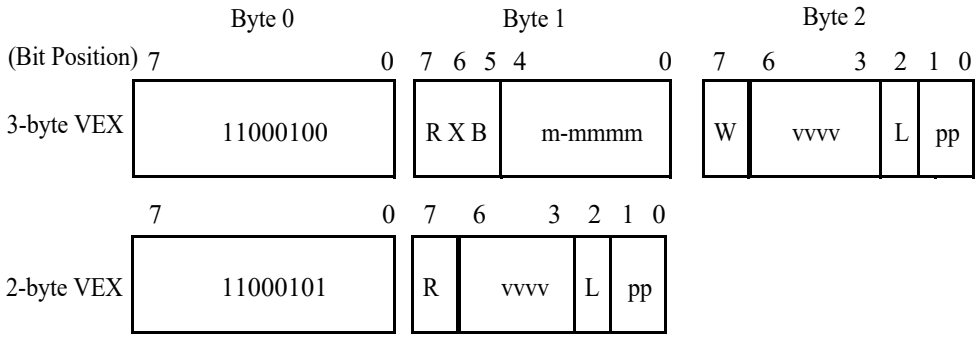
The VEX prefix is required to be the last prefix and immediately precedes the opcode bytes. It must follow any other prefixes. If VEX prefix is present a REX prefix is not supported.

The 3-byte VEX leaves room for future expansion with 3 reserved bits. REX and the 66h/F2h/F3h prefixes are reclaimed for future use.

VEX prefix has a two-byte form and a three byte form. If an instruction syntax can be encoded using the two-byte form, it can also be encoded using the three byte form of VEX. The latter increases the length of the instruction by one byte. This may be helpful in some situations for code alignment.

The VEX prefix supports 256-bit versions of floating-point SSE, SSE2, SSE3, and SSE4 instructions. Note, certain new instruction functionality can only be encoded with the VEX prefix.

The VEX prefix will #UD on any instruction containing MMX register sources or destinations.



R: REX.R in 1's complement (inverted) form
1: Same as REX.R=0 (must be 1 in 32-bit mode)
0: Same as REX.R=1 (64-bit mode only)

X: REX.X in 1's complement (inverted) form
1: Same as REX.X=0 (must be 1 in 32-bit mode)
0: Same as REX.X=1 (64-bit mode only)

B: REX.B in 1's complement (inverted) form
1: Same as REX.B=0 (Ignored in 32-bit mode).
0: Same as REX.B=1 (64-bit mode only)

W: opcode specific (use like REX.W, or used for opcode extension, or ignored, depending on the opcode byte)

m-mmmm:
00000: Reserved for future use (will #UD)
00001: implied 0F leading opcode byte
00010: implied 0F 38 leading opcode bytes
00011: implied 0F 3A leading opcode bytes
00100-11111: Reserved for future use (will #UD)

vvvv: a register specifier (in 1's complement form) or 1111 if unused.

L: Vector Length
0: scalar or 128-bit vector
1: 256-bit vector

pp: opcode extension providing equivalent functionality of a SIMD prefix
00: None
01: 66
10: F3
11: F2

Figure 2-9. VEX bit fields

The following subsections describe the various fields in two or three-byte VEX prefix.

2.3.5.1 VEX Byte 0, bits[7:0]

VEX Byte 0, bits [7:0] must contain the value 11000101b (C5h) or 11000100b (C4h). The 3-byte VEX uses the C4h first byte, while the 2-byte VEX uses the C5h first byte.

2.3.5.2 VEX Byte 1, bit [7] - 'R'

VEX Byte 1, bit [7] contains a bit analogous to a bit inverted REX.R. In protected and compatibility modes the bit must be set to '1' otherwise the instruction is LES or LDS.

This bit is present in both 2- and 3-byte VEX prefixes.

The usage of WRXB bits for legacy instructions is explained in detail section 2.2.1.2 of Intel 64 and IA-32 Architectures Software developer's manual, Volume 2A.

This bit is stored in bit inverted format.

2.3.5.3 3-byte VEX byte 1, bit[6] - 'X'

Bit[6] of the 3-byte VEX byte 1 encodes a bit analogous to a bit inverted REX.X. It is an extension of the SIB Index field in 64-bit modes. In 32-bit modes, this bit must be set to '1' otherwise the instruction is LES or LDS.

This bit is available only in the 3-byte VEX prefix.

This bit is stored in bit inverted format.

2.3.5.4 3-byte VEX byte 1, bit[5] - 'B'

Bit[5] of the 3-byte VEX byte 1 encodes a bit analogous to a bit inverted REX.B. In 64-bit modes, it is an extension of the ModR/M r/m field, or the SIB base field. In 32-bit modes, this bit is ignored.

This bit is available only in the 3-byte VEX prefix.

This bit is stored in bit inverted format.

2.3.5.5 3-byte VEX byte 2, bit[7] - 'W'

Bit[7] of the 3-byte VEX byte 2 is represented by the notation VEX.W. It can provide following functions, depending on the specific opcode.

- For AVX instructions that have equivalent legacy SSE instructions (typically these SSE instructions have a general-purpose register operand with its operand size attribute promotable by REX.W), if REX.W promotes the operand size attribute of the general-purpose register operand in legacy SSE instruction, VEX.W has same meaning in the corresponding AVX equivalent form. In 32-bit modes for these instructions, VEX.W is silently ignored.
- For AVX instructions that have equivalent legacy SSE instructions (typically these SSE instructions have operands with their operand size attribute fixed and not promotable by REX.W), if REX.W is don't care in legacy SSE instruction, VEX.W is ignored in the corresponding AVX equivalent form irrespective of mode.
- For new AVX instructions where VEX.W has no defined function (typically these meant the combination of the opcode byte and VEX.mmmmm did not have any equivalent SSE functions), VEX.W is reserved as zero and setting to other than zero will cause instruction to #UD.

2.3.5.6 2-byte VEX Byte 1, bits[6:3] and 3-byte VEX Byte 2, bits [6:3]- 'vvvv' the Source or Dest Register Specifier

In 32-bit mode the VEX first byte C4 and C5 alias onto the LES and LDS instructions. To maintain compatibility with existing programs the VEX 2nd byte, bits [7:6] must be 11b. To achieve this, the VEX payload bits are selected to place only inverted, 64-bit valid fields (extended register selectors) in these upper bits.

The 2-byte VEX Byte 1, bits [6:3] and the 3-byte VEX, Byte 2, bits [6:3] encode a field (shorthand VEX.vvvv) that for instructions with 2 or more source registers and an XMM or YMM or memory destination encodes the first source register specifier stored in inverted (1's complement) form.

VEX.vvvv is not used by the instructions with one source (except certain shifts, see below) or on instructions with no XMM or YMM or memory destination. If an instruction does not use VEX.vvvv then it should be set to 1111b otherwise instruction will #UD.

In 64-bit mode all 4 bits may be used. See Table for the encoding of the XMM or YMM registers. In 32-bit and 16-bit modes bit 6 must be 1 (if bit 6 is not 1, the 2-byte VEX version will generate LDS instruction and the 3-byte VEX version will ignore this bit).

Table 2-8. VEX.vvvv to Register Name Mapping

VEX.vvvv	Dest Register	General-Purpose Register (If Applicable)[1]	Valid in Legacy/Compatibility 32-bit modes?[2]
1111B	XMM0/YMM0	RAX/EAX	Valid
1110B	XMM1/YMM1	RCX/ECX	Valid
1101B	XMM2/YMM2	RDX/EDX	Valid
1100B	XMM3/YMM3	RBX/EBX	Valid
1011B	XMM4/YMM4	RSP/ESP	Valid
1010B	XMM5/YMM5	RBP/EBP	Valid
1001B	XMM6/YMM6	RSI/ESI	Valid
1000B	XMM7/YMM7	RDI/EDI	Valid
0111B	XMM8/YMM8	R8/R8D	Invalid
0110B	XMM9/YMM9	R9/R9D	Invalid
0101B	XMM10/YMM10	R10/R10D	Invalid
0100B	XMM11/YMM11	R11/R11D	Invalid
0011B	XMM12/YMM12	R12/R12D	Invalid
0010B	XMM13/YMM13	R13/R13D	Invalid
0001B	XMM14/YMM14	R14/R14D	Invalid
0000B	XMM15/YMM15	R15/R15D	Invalid

NOTES:

1. See Section 2.6, "VEX Encoding Support for GPR Instructions" for additional details.
2. Only the first eight General-Purpose Registers are accessible/encodable in 16/32b modes.

The VEX.vvvv field is encoded in bit inverted format for accessing a register operand.

2.3.6 Instruction Operand Encoding and VEX.vvvv, ModR/M

VEX-encoded instructions support three-operand and four-operand instruction syntax. Some VEX-encoded instructions have syntax with less than three operands, e.g., VEX-encoded pack shift instructions support one source operand and one destination operand).

The roles of VEX.vvvv, reg field of ModR/M byte (ModR/M.reg), r/m field of ModR/M byte (ModR/M.r/m) with respect to encoding destination and source operands vary with different type of instruction syntax.

The role of VEX.vvvv can be summarized to three situations:

- VEX.vvvv encodes the first source register operand, specified in inverted (1's complement) form and is valid for instructions with 2 or more source operands.
- VEX.vvvv encodes the destination register operand, specified in 1's complement form for certain vector shifts. The instructions where VEX.vvvv is used as a destination are listed in Table 2-9. The notation in the "Opcode" column in Table 2-9 is described in detail in section 3.1.1.
- VEX.vvvv does not encode any operand, the field is reserved and should contain 1111b.

Table 2-9. Instructions with a VEX.vvvv Destination

Opcode	Instruction mnemonic
VEX.128.66.0F 73 /7 ib	VPSLLDQ xmm1, xmm2, imm8
VEX.128.66.0F 73 /3 ib	VPSRLDQ xmm1, xmm2, imm8

Table 2-9. Instructions with a VEX.vvvv Destination (Contd.)

Opcode	Instruction mnemonic
VEX.128.66.0F 71 /2 ib	VPSRLW xmm1, xmm2, imm8
VEX.128.66.0F 72 /2 ib	VPSRLD xmm1, xmm2, imm8
VEX.128.66.0F 73 /2 ib	VPSRLQ xmm1, xmm2, imm8
VEX.128.66.0F 71 /4 ib	VPSRAW xmm1, xmm2, imm8
VEX.128.66.0F 72 /4 ib	VPSRAD xmm1, xmm2, imm8
VEX.128.66.0F 71 /6 ib	VPSLLW xmm1, xmm2, imm8
VEX.128.66.0F 72 /6 ib	VPSLLD xmm1, xmm2, imm8
VEX.128.66.0F 73 /6 ib	VPSLLQ xmm1, xmm2, imm8

The role of ModR/M.r/m field can be summarized to two situations:

- ModR/M.r/m encodes the instruction operand that references a memory address.
- For some instructions that do not support memory addressing semantics, ModR/M.r/m encodes either the destination register operand or a source register operand.

The role of ModR/M.reg field can be summarized to two situations:

- ModR/M.reg encodes either the destination register operand or a source register operand.
- For some instructions, ModR/M.reg is treated as an opcode extension and not used to encode any instruction operand.

For instruction syntax that support four operands, VEX.vvvv, ModR/M.r/m, ModR/M.reg encodes three of the four operands. The role of bits 7:4 of the immediate byte serves the following situation:

- Imm8[7:4] encodes the third source register operand.

2.3.6.1 3-byte VEX byte 1, bits[4:0] - "m-mmmm"

Bits[4:0] of the 3-byte VEX byte 1 encode an implied leading opcode byte (0F, 0F 38, or 0F 3A). Several bits are reserved for future use and will #UD unless 0.

Table 2-10. VEX.m-mmmm Interpretation

VEX.m-mmmm	Implied Leading Opcode Bytes
00000B	Reserved
00001B	0F
00010B	0F 38
00011B	0F 3A
00100-11111B	Reserved
(2-byte VEX)	0F

VEX.m-mmmm is only available on the 3-byte VEX. The 2-byte VEX implies a leading 0Fh opcode byte.

2.3.6.2 2-byte VEX byte 1, bit[2], and 3-byte VEX byte 2, bit [2]- "L"

The vector length field, VEX.L, is encoded in bit[2] of either the second byte of 2-byte VEX, or the third byte of 3-byte VEX. If "VEX.L = 1", it indicates 256-bit vector operation. "VEX.L = 0" indicates scalar and 128-bit vector operations.

The instruction VZEROUPPER is a special case that is encoded with VEX.L = 0, although its operation zero's bits 255:128 of all YMM registers accessible in the current operating mode. See Table 2-11.

Table 2-11. VEX.L Interpretation

VEX.L	Vector Length
0	128-bit (or 32/64-bit scalar)
1	256-bit

2.3.6.3 2-byte VEX byte 1, bits[1:0], and 3-byte VEX byte 2, bits [1:0]- "pp"

Up to one implied prefix is encoded by bits[1:0] of either the 2-byte VEX byte 1 or the 3-byte VEX byte 2. The prefix behaves as if it was encoded prior to VEX, but after all other encoded prefixes. See Table 2-12.

Table 2-12. VEX.pp Interpretation

pp	Implies this prefix after other prefixes but before VEX
00B	None
01B	66
10B	F3
11B	F2

2.3.7 The Opcode Byte

One (and only one) opcode byte follows the 2 or 3 byte VEX. Legal opcodes are specified in Appendix B, in color. Any instruction that uses illegal opcode will #UD.

2.3.8 The ModR/M, SIB, and Displacement Bytes

The encodings are unchanged but the interpretation of reg_field or rm_field differs (see above).

2.3.9 The Third Source Operand (Immediate Byte)

VEX-encoded instructions can support instruction with a four operand syntax. VBLENDVPD, VBLENDVPS, and PBLENDVB use imm8[7:4] to encode one of the source registers.

2.3.10 Intel® AVX Instructions and the Upper 128-bits of YMM registers

If an instruction with a destination XMM register is encoded with a VEX prefix, the processor zeroes the upper bits (above bit 128) of the equivalent YMM register. Legacy SSE instructions without VEX preserve the upper bits.

2.3.10.1 Vector Length Transition and Programming Considerations

An instruction encoded with a VEX.128 prefix that loads a YMM register operand operates as follows:

- Data is loaded into bits 127:0 of the register
- Bits above bit 127 in the register are cleared.

Thus, such an instruction clears bits 255:128 of a destination YMM register on processors with a maximum vector-register width of 256 bits. In the event that future processors extend the vector registers to greater widths, an instruction encoded with a VEX.128 or VEX.256 prefix will also clear any bits beyond bit 255. (This is in contrast with legacy SSE instructions, which have no VEX prefix; these modify only bits 127:0 of any destination register operand.)

Programmers should bear in mind that instructions encoded with VEX.128 and VEX.256 prefixes will clear any future extensions to the vector registers. A calling function that uses such extensions should save their state before calling legacy functions. This is not possible for involuntary calls (e.g., into an interrupt-service routine). It is

recommended that software handling involuntary calls accommodate this by not executing instructions encoded with VEX.128 and VEX.256 prefixes. In the event that it is not possible or desirable to restrict these instructions, then software must take special care to avoid actions that would, on future processors, zero the upper bits of vector registers.

Processors that support further vector-register extensions (defining bits beyond bit 255) will also extend the XSAVE and XRSTOR instructions to save and restore these extensions. To ensure forward compatibility, software that handles involuntary calls and that uses instructions encoded with VEX.128 and VEX.256 prefixes should first save and then restore the vector registers (with any extensions) using the XSAVE and XRSTOR instructions with save/restore masks that set bits that correspond to all vector-register extensions. Ideally, software should rely on a mechanism that is cognizant of which bits to set. (E.g., an OS mechanism that sets the save/restore mask bits for all vector-register extensions that are enabled in XCR0.) Saving and restoring state with instructions other than XSAVE and XRSTOR will, on future processors with wider vector registers, corrupt the extended state of the vector registers - even if doing so functions correctly on processors supporting 256-bit vector registers. (The same is true if XSAVE and XRSTOR are used with a save/restore mask that does not set bits corresponding to all supported extensions to the vector registers.)

2.3.11 Intel® AVX Instruction Length

The Intel AVX instructions described in this document (including VEX and ignoring other prefixes) do not exceed 11 bytes in length, but may increase in the future. The maximum length of an Intel 64 and IA-32 instruction remains 15 bytes.

2.3.12 Vector SIB (VSIB) Memory Addressing

In Intel® Advanced Vector Extensions 2 (Intel® AVX2), an SIB byte that follows the ModR/M byte can support VSIB memory addressing to an array of linear addresses. VSIB addressing is only supported in a subset of Intel AVX2 instructions. VSIB memory addressing requires 32-bit or 64-bit effective address. In 32-bit mode, VSIB addressing is not supported when address size attribute is overridden to 16 bits. In 16-bit protected mode, VSIB memory addressing is permitted if address size attribute is overridden to 32 bits. Additionally, VSIB memory addressing is supported only with VEX prefix.

In VSIB memory addressing, the SIB byte consists of:

- The scale field (bit 7:6) specifies the scale factor.
- The index field (bits 5:3) specifies the register number of the vector index register, each element in the vector register specifies an index.
- The base field (bits 2:0) specifies the register number of the base register.

Table 2-13 shows the 32-bit VSIB addressing form. It is organized to give 256 possible values of the SIB byte (in hexadecimal). General purpose registers used as a base are indicated across the top of the table, along with corresponding values for the SIB byte's base field. The register names also include R8D-R15D applicable only in 64-bit mode (when address size override prefix is used, but the value of VEX.B is not shown in Table 2-13). In 32-bit mode, R8D-R15D does not apply.

Table rows in the body of the table indicate the vector index register used as the index field and each supported scaling factor shown separately. Vector registers used in the index field can be XMM or YMM registers. The left-most column includes vector registers VR8-VR15 (i.e., XMM8/YMM8-XMM15/YMM15), which are only available in 64-bit mode and does not apply if encoding in 32-bit mode.

Table 2-13. 32-Bit VSIB Addressing Forms of the SIB Byte

r32 (In decimal) Base = (In binary) Base =				EAX/ R8D 0 000	ECX/ R9D 1 001	EDX/ R10D 2 010	EBX/ R11D 3 011	ESP/ R12D 4 100	EBP/ R13D[1] 5 101	ESI/ R14D 6 110	EDI/ R15D 7 111
Scaled Index		**SS**	**Index**	**Value of SIB Byte (in Hexadecimal)**							
VR0/VR8 VR1/VR9 VR2/VR10 VR3/VR11 VR4/VR12 VR5/VR13 VR6/VR14 VR7/VR15	*1	00	000 001 010 011 100 101 110 111	00 08 10 18 20 28 30 38	01 09 11 19 21 29 31 39	02 0A 12 1A 22 2A 32 3A	03 0B 13 1B 23 2B 33 3B	04 0C 14 1C 24 2C 34 3C	05 0D 15 1D 25 2D 35 3D	06 0E 16 1E 26 2E 36 3E	07 0F 17 1F 27 2F 37 3F
VR0/VR8 VR1/VR9 VR2/VR10 VR3/VR11 VR4/VR12 VR5/VR13 VR6/VR14 VR7/VR15	*2	01	000 001 010 011 100 101 110 111	40 48 50 58 60 68 70 78	41 49 51 59 61 69 71 79	42 4A 52 5A 62 6A 72 7A	43 4B 53 5B 63 6B 73 7B	44 4C 54 5C 64 6C 74 7C	45 4D 55 5D 65 6D 75 7D	46 4E 56 5E 66 6E 76 7E	47 4F 57 5F 67 6F 77 7F
VR0/VR8 VR1/VR9 VR2/VR10 VR3/VR11 VR4/VR12 VR5/VR13 VR6/VR14 VR7/VR15	*4	10	000 001 010 011 100 101 110 111	80 88 90 98 A0 A8 B0 B8	81 89 91 89 A1 A9 B1 B9	82 8A 92 9A A2 AA B2 BA	83 8B 93 9B A3 AB B3 BB	84 8C 94 9C A4 AC B4 BC	85 8D 95 9D A5 AD B5 BD	86 8E 96 9E A6 AE B6 BE	87 8F 97 9F A7 AF B7 BF
VR0/VR8 VR1/VR9 VR2/VR10 VR3/VR11 VR4/VR12 VR5/VR13 VR6/VR14 VR7/VR15	*8	11	000 001 010 011 100 101 110 111	C0 C8 D0 D8 E0 E8 F0 F8	C1 C9 D1 D9 E1 E9 F1 F9	C2 CA D2 DA E2 EA F2 FA	C3 CB D3 DB E3 EB F3 FB	C4 CC D4 DC E4 EC F4 FC	C5 CD D5 DD E5 ED F5 FD	C6 CE D6 DE E6 EE F6 FE	C7 CF D7 DF E7 EF F7 FF

NOTES:

1. If ModR/M.mod = 00b, the base address is zero, then effective address is computed as [scaled vector index] + disp32. Otherwise the base address is computed as [EBP/R13]+ disp, the displacement is either 8 bit or 32 bit depending on the value of ModR/M.mod:

MOD	Effective Address
00b	[Scaled Vector Register] + Disp32
01b	[Scaled Vector Register] + Disp8 + [EBP/R13]
10b	[Scaled Vector Register] + Disp32 + [EBP/R13]

2.3.12.1 64-bit Mode VSIB Memory Addressing

In 64-bit mode VSIB memory addressing uses the VEX.B field and the base field of the SIB byte to encode one of the 16 general-purpose register as the base register. The VEX.X field and the index field of the SIB byte encode one of the 16 vector registers as the vector index register.

In 64-bit mode the top row of Table 2-13 base register should be interpreted as the full 64-bit of each register.

2.4 INTEL® ADVANCED MATRIX EXTENSIONS (INTEL® AMX)

Intel® AMX instructions follow the general documentation convention established in previous sections. Additionally, Intel® Advanced Matrix Extensions use notation conventions as described below.

In the instruction encoding boxes, **sibmem** is used to denote an encoding where a ModR/M byte and SIB byte are used to indicate a memory operation where the base and displacement are used to point to memory, and the index

register (if present) is used to denote a stride between memory rows. The index register is scaled by the sib.scale field as usual. The base register is added to the displacement, if present.

In the instruction encoding, the ModR/M byte is represented several ways depending on the role it plays. The ModR/M byte has 3 fields: 2-bit ModR/M.mod field, a 3-bit ModR/M.reg field and a 3-bit ModR/M.r/m field. When all bits of the ModR/M byte have fixed values for an instruction, the 2-hex nibble value of that byte is presented after the opcode in the encoding boxes on the instruction description pages. When only some fields of the ModR/M byte must contain fixed values, those values are specified as follows:

- If only the ModR/M.mod must be 0b11, and ModR/M.reg and ModR/M.r/m fields are unrestricted, this is denoted as **11:rrr:bbb**. The **rrr** correspond to the 3-bits of the ModR/M.reg field and the **bbb** correspond to the 3-bits of the ModR/M.r/m field.
- If the ModR/M.mod field is constrained to be a value other than 0b11, i.e., it must be one of 0b00, 0b01, or 0b10, then the notation !(11) is used.
- If the ModR/M.reg field had a specific required value, e.g., 0b101, that would be denoted as mm:101:bbb.

NOTE

Historically this document only specified the ModR/M.reg field restrictions with the notation /0 ... /7 and did not specify restrictions on the ModR/M.mod and ModR/M.r/m fields in the encoding boxes.

2.5 INTEL® AVX AND INTEL® SSE INSTRUCTION EXCEPTION CLASSIFICATION

To look up the exceptions of legacy 128-bit SIMD instruction, 128-bit VEX-encoded instructions, and 256-bit VEX-encoded instruction, Table summarizes the exception behavior into separate classes, with detailed exception conditions defined in sub-sections 2.5.1 through 2.6.1. For example, ADDPS contains the entry:

"See Exceptions Type 2"

In this entry, "Type2" can be looked up in Table .

The instruction's corresponding CPUID feature flag can be identified in the fourth column of the Instruction summary table.

Note: #UD on CPUID feature flags=0 is not guaranteed in a virtualized environment if the hardware supports the feature flag.

NOTE

Instructions that operate only with MMX, X87, or general-purpose registers are not covered by the exception classes defined in this section. For instructions that operate on MMX registers, see Section 23.25.3, "Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

Table 2-14. Exception Class Description

Exception Class	Instruction Set	Mem Arg	Floating-Point Exceptions (#XM)
Type 1	AVX, Legacy SSE	16/32 byte explicitly aligned	None
Type 2	AVX, Legacy SSE	16/32 byte not explicitly aligned	Yes
Type 3	AVX, Legacy SSE	< 16 byte	Yes
Type 4	AVX, Legacy SSE	16/32 byte not explicitly aligned	No
Type 5	AVX, Legacy SSE	< 16 byte	No
Type 6	AVX (no Legacy SSE)	Varies	(At present, none do)

Table 2-14. Exception Class Description (Contd.)

Exception Class	Instruction Set	Mem Arg	Floating-Point Exceptions (#XM)
Type 7	AVX, Legacy SSE	None	None
Type 8	AVX	None	None
Type 11	F16C	8 or 16 byte, Not explicitly aligned, no AC#	Yes
Type 12	AVX2 Gathers	Not explicitly aligned, no AC#	No

See Table 2-15 for lists of instructions in each exception class.

Table 2-15. Instructions in Each Exception Class

Exception Class	Instruction
Type 1	(V)MOVAPD, (V)MOVAPS, (V)MOVDQA, (V)MOVNTDQ, (V)MOVNTDQA, (V)MOVNTPD, (V)MOVNTPS
Type 2	(V)ADDPD, (V)ADDPS, (V)ADDSUBPD, (V)ADDSUBPS, (V)CMPPD, (V)CMPPS, (V)CVTDQ2PS, (V)CVTPD2DQ, (V)CVTPD2PS, (V)CVTPS2DQ, (V)CVTTPD2DQ, (V)CVTTPS2DQ, (V)DIVPD, (V)DIVPS, (V)DPPD*, (V)DPPS*, VFMADD132PD, VFMADD213PD, VFMADD231PD, VFMADD132PS, VFMADD213PS, VFMADD231PS, VFMADDSUB132PD, VFMADDSUB213PD, VFMADDSUB231PD, VFMADDSUB132PS, VFMADDSUB213PS, VFMADDSUB231PS, VFMSUBADD132PD, VFMSUBADD213PD, VFMSUBADD231PD, VFMSUBADD132PS, VFMSUBADD213PS, VFMSUBADD231PS, VFMSUB132PD, VFMSUB213PD, VFMSUB231PD, VFMSUB132PS, VFMSUB213PS, VFMSUB231PS, VFNMADD132PD, VFNMADD213PD, VFNMADD231PD, VFNMADD132PS, VFNMADD213PS, VFNMADD231PS, VFNMSUB132PD, VFNMSUB213PD, VFNMSUB231PD, VFNMSUB132PS, VFNMSUB213PS, VFNMSUB231PS, (V)HADDPD, (V)HADDPS, (V)HSUBPD, (V)HSUBPS, (V)MAXPD, (V)MAXPS, (V)MINPD, (V)MINPS, (V)MULPD, (V)MULPS, (V)ROUNDPD, (V)ROUNDPS, (V)SQRTPD, (V)SQRTPS, (V)SUBPD, (V)SUBPS
Type 3	(V)ADDSD, (V)ADDSS, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)COMISS, (V)CVTPS2PD, (V)CVTSD2SI, (V)CVTSD2SS, (V)CVTSI2SD, (V)CVTSI2SS, (V)CVTSS2SD, (V)CVTSS2SI, (V)CVTTSD2SI, (V)CVTTSS2SI, (V)DIVSD, (V)DIVSS, VFMADD132SD, VFMADD213SD, VFMADD231SD, VFMADD132SS, VFMADD213SS, VFMADD231SS, VFMSUB132SD, VFMSUB213SD, VFMSUB231SD, VFMSUB132SS, VFMSUB213SS, VFMSUB231SS, VFNMADD132SD, VFNMADD213SD, VFNMADD231SD, VFNMADD132SS, VFNMADD213SS, VFNMADD231SS, VFNMSUB132SD, VFNMSUB213SD, VFNMSUB231SD, VFNMSUB132SS, VFNMSUB213SS, VFNMSUB231SS, (V)MAXSD, (V)MAXSS, (V)MINSD, (V)MINSS, (V)MULSD, (V)MULSS, (V)ROUNDSD, (V)ROUNDSS, (V)SQRTSD, (V)SQRTSS, (V)SUBSD, (V)SUBSS, (V)UCOMISD, (V)UCOMISS
Type 4	(V)AESDEC, (V)AESDECLAST, (V)AESENC, (V)AESENCLAST, (V)AESIMC, (V)AESKEYGENASSIST, (V)ANDPD, (V)ANDPS, (V)ANDNPD, (V)ANDNPS, (V)BLENDPD, (V)BLENDPS, VBLENDVPD, VBLENDVPS, (V)LDDQU***, (V)MASKMOVDQU, (V)PTEST, VTESTPS, VTESTPD, (V)MOVDQU*, (V)MOVSHDUP, (V)MOVSLDUP, (V)MOVUPD*, (V)MOVUPS*, (V)MPSADBW, (V)ORPD, (V)ORPS, (V)PABSB, (V)PABSW, (V)PABSD, (V)PACKSSWB, (V)PACKSSDW, (V)PACKUSWB, (V)PACKUSDW, (V)PADDB, (V)PADDW, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PALIGNR, (V)PAND, (V)PANDN, (V)PAVGB, (V)PAVGW, (V)PBLENDVB, (V)PBLENDW, (V)PCMP(E/I)STRI/M***, (V)PCMPEQB, (V)PCMPEQW, (V)PCMPEQD, (V)PCMPEQQ, (V)PCMPGTB, (V)PCMPGTW, (V)PCMPGTD, (V)PCMPGTQ, (V)PCLMULQDQ, (V)PHADDW, (V)PHADDD, (V)PHADDSW, (V)PHMINPOSUW, (V)PHSUBD, (V)PHSUBW, (V)PHSUBSW, (V)PMADDWD, (V)PMADDUBSW, (V)PMAXSB, (V)PMAXSW, (V)PMAXSD, (V)PMAXUB, (V)PMAXUW, (V)PMAXUD, (V)PMINSB, (V)PMINSW, (V)PMINSD, (V)PMINUB, (V)PMINUW, (V)PMINUD, (V)PMULHUW, (V)PMULHRSW, (V)PMULHW, (V)PMULLW, (V)PMULLD, (V)PMULUDQ, (V)PMULDQ, (V)POR, (V)PSADBW, (V)PSHUFB, (V)PSHUFD, (V)PSHUFHW, (V)PSHUFLW, (V)PSIGNB, (V)PSIGNW, (V)PSIGND, (V)PSLLW, (V)PSLLD, (V)PSLLQ, (V)PSRAW, (V)PSRAD, (V)PSRLW, (V)PSRLD, (V)PSRLQ, (V)PSUBB, (V)PSUBW, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PUNPCKHBW, (V)PUNPCKHWD, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKLBW, (V)PUNPCKLWD, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PXOR, (V)RCPPS, (V)RSQRTPS, (V)SHUFPD, (V)SHUFPS, (V)UNPCKHPD, (V)UNPCKHPS, (V)UNPCKLPD, (V)UNPCKLPS, (V)XORPD, (V)XORPS, VPBLENDD, VPERMD, VPERMPS, VPERMPD, VPERMQ, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ, VPERMILPD, VPERMILPS, VPERM2F128

Table 2-15. Instructions in Each Exception Class (Contd.)

Exception Class	Instruction
Type 5	(V)CVTDQ2PD, (V)EXTRACTPS, (V)INSERTPS, (V)MOVD, (V)MOVQ, (V)MOVDDUP, (V)MOVLPD, (V)MOVLPS, (V)MOVHPD, (V)MOVHPS, (V)MOVSD, (V)MOVSS, (V)PEXTRB, (V)PEXTRD, (V)PEXTRW, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ, PMOVSXBW, (V)RCPSS, (V)RSQRTSS, (V)PMOVSX/ZX, VLDMXCSR*, VSTMXCSR
Type 6	VEXTRACTF128/VEXTRACTFxxxx, VBROADCASTSS, VBROADCASTSD, VBROADCASTF128, VINSERTF128, VMASKMOVPS**, VMASKMOVPD**, VPMASKMOVD, VPMASKMOVQ, VBROADCASTI128, VPBROADCASTB, VPBROADCASTD, VPBROADCASTW, VPBROADCASTQ, VEXTRACTI128, VINSERTI128, VPERM2I128
Type 7	(V)MOVLHPS, (V)MOVHLPS, (V)MOVMSKPD, (V)MOVMSKPS, (V)PMOVMSKB, (V)PSLLDQ, (V)PSRLDQ, (V)PSLLW, (V)PSLLD, (V)PSLLQ, (V)PSRAW, (V)PSRAD, (V)PSRLW, (V)PSRLD, (V)PSRLQ
Type 8	VZEROALL, VZEROUPPER
Type 11	VCVTPH2PS, VCVTPS2PH
Type 12	VGATHERDPS, VGATHERDPD, VGATHERQPS, VGATHERQPD, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

(*) - Additional exception restrictions are present - see the Instruction description for details

(**) - Instruction behavior on alignment check reporting with mask bits of less than all 1s are the same as with mask bits of all 1s, i.e., no alignment checks are performed.

(***) - PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM, and LDDQU instructions do not cause #GP if the memory operand is not aligned to 16-Byte boundary.

Table 2-15 classifies exception behaviors for Intel AVX instructions. Within each class of exception conditions that are listed in Table 2-18 through Table 2-27, certain subsets of Intel AVX instructions may be subject to #UD exception depending on the encoded value of the VEX.L field. Table 2-16 and Table 2-17 provide supplemental information of Intel AVX instructions that may be subject to #UD exception if encoded with incorrect values in the VEX.W or VEX.L field.

Table 2-16. #UD Exception and VEX.W=1 Encoding

Exception Class	#UD If VEX.W = 1 in All Modes	#UD If VEX.W = 1 in Non-64-Bit Modes
Type 1		
Type 2		
Type 3		
Type 4	VBLENDVPD, VBLENDVPS, VPBLENDVB, VTESTPD, VTESTPS, VPBLENDD, VPERMD, VPERMPS, VPERM2I128, VPSRAVD, VPERMILPD, VPERMILPS, VPERM2F128	
Type 5		
Type 6	VEXTRACTF128, VBROADCASTSS, VBROADCASTSD, VBROADCASTF128, VINSERTF128, VMASKMOVPS, VMASKMOVPD, VBROADCASTI128, VPBROADCASTB/W/D, VEXTRACTI128, VINSERTI128	
Type 7		
Type 8		
Type 11	VCVTPH2PS, VCVTPS2PH	
Type 12		

Table 2-17. #UD Exception and VEX.L Field Encoding

Exception Class	#UD If VEX.L = 0	#UD If (VEX.L = 1 && AVX2 not present && AVX present)	#UD If (VEX.L = 1 && AVX2 present)
Type 1		VMOVNTDQA	
Type 2		VDPPD	VDPPD
Type 3			
Type 4		VMASKMOVDQU, VMPSADBW, VPABSB/W/D, VPACKSSWB/DW, VPACKUSWB/DW, VPADDB/W/D, VPADDQ, VPADDSB/W, VPADDUSB/W, VPALIGNR, VPAND, VPANDN, VPAVGB/W, VPBLENDVB, VPBLENDW, VPCMP(E/I)STRI/M, VPCMPEQB/W/D/Q, VPCMPGTB/W/D/Q, VPHADDW/D, VPHADDSW, VPHMINPOSUW, VPHSUBD/W, VPHSUBSW, VPMADDWD, VPMADDUBSW, VPMAXSB/W/D, VPMAXUB/W/D, VPMINSB/W/D, VPMINUB/W/D, VPMULHUW, VPMULHRSW, VPMULHW/LW, VPMULLD, VPMULUDQ, VPMULDQ, VPOR, VPSADBW, VPSHUFB/D, VPSHUFHW/LW, VPSIGNB/W/D, VPSLLW/D/Q, VPSRAW/D, VPSRLW/D/Q, VPSUBB/W/D/Q, VPSUBSB/W, VPUNPCKHBW/WD/DQ, VPUNPCKHQDQ, VPUNPCKLBW/WD/DQ, VPUNPCKLQDQ, VPXOR	VPCMP(E/I)STRI/M, PHMINPOSUW
Type 5		VEXTRACTPS, VINSERTPS, VMOVD, VMOVQ, VMOVLPD, VMOVLPS, VMOVHPD, VMOVHPS, VPEXTRB, VPEXTRD, VPEXTRW, VPEXTRQ, VPINSRB, VPINSRD, VPINSRW, VPINSRQ, VPMOVSX/ZX, VLDMXCSR, VSTMXCSR	Same as column 3
Type 6	VEXTRACTF128, VPERM2F128, VBROADCASTSD, VBROADCASTF128, VINSERTF128,		
Type 7		VMOVLHPS, VMOVHLPS, VPMOVMSKB, VPSLLDQ, VPSRLDQ, VPSLLW, VPSLLD, VPSLLQ, VPSRAW, VPSRAD, VPSRLW, VPSRLD, VPSRLQ	VMOVLHPS, VMOVHLPS
Type 8			
Type 11			
Type 12			

2.5.1 Exceptions Type 1 (Aligned Memory Reference)

Table 2-18. Type 1 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X	X	VEX.256: Memory operand is not 32-byte aligned. VEX.128: Memory operand is not 16-byte aligned.
	X	X	X	X	Legacy SSE: Memory operand is not 16-byte aligned.
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.

2.5.2 Exceptions Type 2 (>=16 Byte Memory Reference, Unaligned)

Table 2-19. Type 2 Class Exception Conditions

Exception	Real	Virtual 8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)	X	X	X	X	Legacy SSE: Memory operand is not 16-byte aligned.
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1.

2.5.3 Exceptions Type 3 (<16 Byte Memory Argument)

Table 2-20. Type 3 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1.

2.5.4 Exceptions Type 4 (>=16 Byte Mem Arg, No Alignment, No Floating-point Exceptions)

Table 2-21. Type 4 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)	X	X	X	X	Legacy SSE: Memory operand is not 16-byte aligned.[1]
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.

NOTES:

1. LDDQU, MOVUPD, MOVUPS, PCMPESTRI, PCMPESTRM, PCMPISTRI, and PCMPISTRM instructions do not cause #GP if the memory operand is not aligned to 16-Byte boundary.

2.5.5 Exceptions Type 5 (<16 Byte Mem Arg and No FP Exceptions)

Table 2-22. Type 5 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.5.6 Exceptions Type 6 (VEX-Encoded Instructions without Legacy SSE Analogues)

Note: At present, the AVX instructions in this category do not generate floating-point exceptions.

Table 2-23. Type 6 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
			X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
			X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
Page Fault #PF(fault-code)			X	X	For a page fault.
Alignment Check #AC(0)			X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.5.7 Exceptions Type 7 (No FP Exceptions, No Memory Arg)

Table 2-24. Type 7 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.

2.5.8 Exceptions Type 8 (AVX and No Memory Argument)

Table 2-25. Type 8 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			Always in Real or Virtual-8086 mode.
			X	X	If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0. If CPUID.01H.ECX.AVX[bit 28]=0. If VEX.vvvv ? 1111B.
	X	X	X	X	If proceeded by a LOCK prefix (F0H).
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.

2.5.9 Exceptions Type 11 (VEX-only, Mem Arg, No AC, Floating-point Exceptions)

Table 2-26. Type 11 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF (fault-code)		X	X	X	For a page fault.
SIMD Floating-Point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1.

2.5.10 Exceptions Type 12 (VEX-only, VSIB Mem Arg, No AC, No Floating-point Exceptions)

Table 2-27. Type 12 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			VEX prefix.
			X	X	VEX prefix: If XCR0[2:1] ? '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	NA	If address size attribute is 16 bit.
	X	X	X	X	If ModR/M.mod = '11b'.
	X	X	X	X	If ModR/M.rm ? '100b'.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
	X	X	X	X	If any vector register is used more than once between the destination register, mask register and the index register in VSIB addressing.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF (fault-code)		X	X	X	For a page fault.

2.6 VEX ENCODING SUPPORT FOR GPR INSTRUCTIONS

VEX prefix may be used to encode instructions that operate on neither YMM nor XMM registers. VEX-encoded general-purpose-register instructions have the following properties:

- Instruction syntax support for three encodable operands.
- Encoding support for instruction syntax of non-destructive source operand, destination operand encoded via VEX.vvvv, and destructive three-operand syntax.
- Elimination of escape opcode byte (0FH), two-byte escape via a compact bit field representation within the VEX prefix.
- Elimination of the need to use REX prefix to encode the extended half of general-purpose register sets (R8-R15) for direct register access or memory addressing.
- Flexible and more compact bit fields are provided in the VEX prefix to retain the full functionality provided by REX prefix. REX.W, REX.X, REX.B functionalities are provided in the three-byte VEX prefix only.
- VEX-encoded GPR instructions are encoded with VEX.L=0.

Any VEX-encoded GPR instruction with a 66H, F2H, or F3H prefix preceding VEX will #UD.
Any VEX-encoded GPR instruction with a REX prefix proceeding VEX will #UD.
VEX-encoded GPR instructions are not supported in real and virtual 8086 modes.

2.6.1 Exceptions Type 13 (VEX-Encoded GPR Instructions)

The exception conditions applicable to VEX-encoded GPR instruction differs from those of legacy GPR instructions. Table 2-28 lists VEX-encoded GPR instructions. The exception conditions for VEX-encoded GRP instructions are found in Table 2-29 for those instructions which have a default operand size of 32 bits and 16-bit operand size is not encodable.

Table 2-28. VEX-Encoded GPR Instructions

Exception Class	Instruction
Type 13	ANDN, BEXTR, BLSI, BLSMSK, BLSR, BZHI, MULX, PDEP, PEXT, RORX, SARX, SHLX, SHRX

(*) - Additional exception restrictions are present - see the Instruction description for details.

Table 2-29. Type 13 Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If BMI1/BMI2 CPUID feature flag is '0'.
	X	X			If a VEX prefix is present.
	X	X	X	X	If VEX.L = 1.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
Stack, #SS(0)	X	X	X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.7 INTEL® AVX-512 ENCODING

The majority of the Intel AVX-512 family of instructions (operating on 512/256/128-bit vector register operands) are encoded using a new prefix (called EVEX). Opmask instructions (operating on opmask register operands) are encoded using the VEX prefix. The EVEX prefix has some parts resembling the instruction encoding scheme using the VEX prefix, and many other capabilities not available with the VEX prefix.

The significant feature differences between EVEX and VEX are summarized below.

- EVEX is a 4-Byte prefix (the first byte must be 62H); VEX is either a 2-Byte (C5H is the first byte) or 3-Byte (C4H is the first byte) prefix.
- EVEX prefix can encode 32 vector registers (XMM/YMM/ZMM) in 64-bit mode.
- EVEX prefix can encode an opmask register for conditional processing or selection control in EVEX-encoded vector instructions. Opmask instructions, whose source/destination operands are opmask registers and treat the content of an opmask register as a single value, are encoded using the VEX prefix.
- EVEX memory addressing with disp8 form uses a compressed disp8 encoding scheme to improve the encoding density of the instruction byte stream.
- EVEX prefix can encode functionality that are specific to instruction classes (e.g., packed instruction with "load+op" semantic can support embedded broadcast functionality, floating-point instruction with rounding semantic can support static rounding functionality, floating-point instruction with non-rounding arithmetic semantic can support "suppress all exceptions" functionality).

2.7.1 Instruction Format and EVEX

The placement of the EVEX prefix in an IA instruction is represented in Figure 2-10. Note that the values contained within brackets are optional.

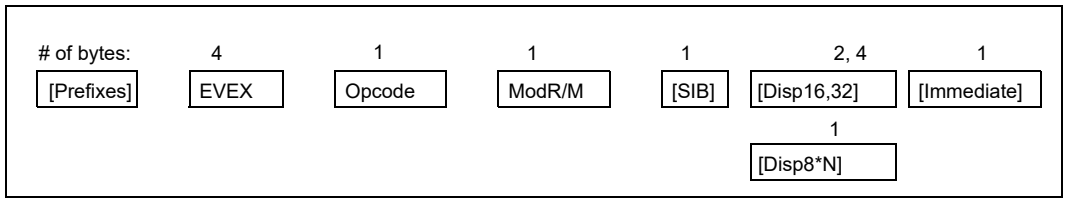


Figure 2-10. Intel® AVX-512 Instruction Format and the EVEX Prefix

The EVEX prefix is a 4-byte prefix, with the first two bytes derived from unused encoding form of the 32-bit-mode-only BOUND instruction. The layout of the EVEX prefix is shown in Figure 2-11. The first byte must be 62H, followed by three payload bytes, denoted as P0, P1, and P2 individually or collectively as P[23:0] (see Figure 2-11).

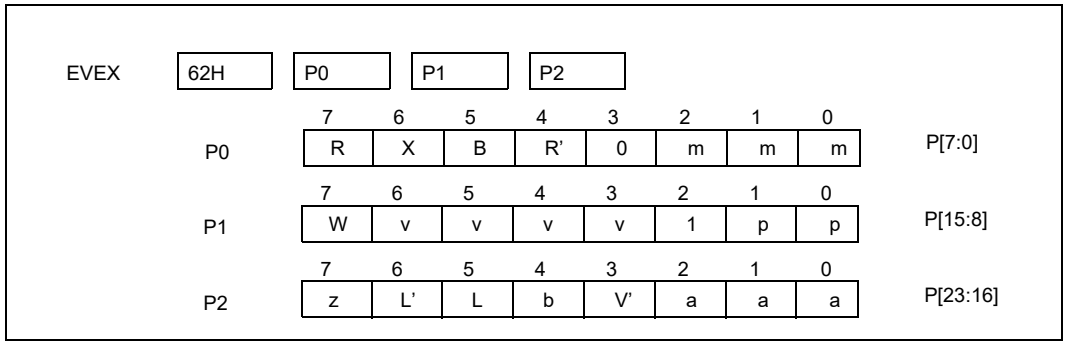


Figure 2-11. Bit Field Layout of the EVEX Prefix[1]

NOTES:

1. See Table 2-30 for additional details on bit fields.

Table 2-30. EVEX Prefix Bit Field Functional Grouping

Notation	Bit field Group	Position	Comment
EVEX.mmm	Access to up to eight decoding maps	P[2:0]	Currently, only the following decoding maps are supported: 1, 2, 3, 5, and 6.
--	Reserved	P[3]	Must be 0.
EVEX.R'	High-16 register specifier modifier	P[4]	Combine with EVEX.R and ModR/M.reg. This bit is stored in inverted format.
EVEX.RXB	Next-8 register specifier modifier	P[7:5]	Combine with ModR/M.reg, ModR/M.rm (base, index/vidx). This field is encoded in bit inverted format.
EVEX.X	High-16 register specifier modifier	P[6]	Combine with EVEX.B and ModR/M.rm, when SIB/VSIB absent.
EVEX.pp	Compressed legacy prefix	P[9:8]	Identical to VEX.pp.
--	Fixed Value	P[10]	Must be 1.
EVEX.vvvv	VVVV register specifier	P[14:11]	Same as VEX.vvvv. This field is encoded in bit inverted format.
EVEX.W	Operand size promotion/Opcode extension	P[15]	
EVEX.aaa	Embedded opmask register specifier	P[18:16]	
EVEX.V'	High-16 VVVV/VIDX register specifier	P[19]	Combine with EVEX.vvvv or when VSIB present. This bit is stored in inverted format.
EVEX.b	Broadcast/RC/SAE Context	P[20]	
EVEX.L'L	Vector length/RC	P[22:21]	
EVEX.z	Zeroing/Merging	P[23]	

The bit fields in P[23:0] are divided into the following functional groups (Table 2-30 provides a tabular summary):

- Reserved bits: P[3] must be 0, otherwise #UD.
- Fixed-value bit: P[10] must be 1, otherwise #UD.
- Compressed legacy prefix/escape bytes: P[1:0] is identical to the lowest 2 bits of VEX.mmmmm; P[9:8] is identical to VEX.pp.
- EVEX.mmm: P[2:0] provides access to up to eight decoding maps. Currently, only the following decoding maps are supported: 1, 2, 3, 5, and 6. Map ids 1, 2, and 3 are denoted by 0F, 0F38, and 0F3A, respectively, in the instruction encoding descriptions.
- Operand specifier modifier bits for vector register, general purpose register, memory addressing: P[7:5] allows access to the next set of 8 registers beyond the low 8 registers when combined with ModR/M register specifiers.
- Operand specifier modifier bit for vector register: P[4] (or EVEX.R') allows access to the high 16 vector register set when combined with P[7] and ModR/M.reg specifier; P[6] can also provide access to a high 16 vector register when SIB or VSIB addressing are not needed.
- Non-destructive source /vector index operand specifier: P[19] and P[14:11] encode the second source vector register operand in a non-destructive source syntax, vector index register operand can access an upper 16 vector register using P[19].
- Op-mask register specifiers: P[18:16] encodes op-mask register set k0-k7 in instructions operating on vector registers.
- EVEX.W: P[15] is similar to VEX.W which serves either as opcode extension bit or operand size promotion to 64-bit in 64-bit mode.
- Vector destination merging/zeroing: P[23] encodes the destination result behavior which either zeroes the masked elements or leave masked element unchanged.
- Broadcast/Static-rounding/SAE context bit: P[20] encodes multiple functionality, which differs across different classes of instructions and can affect the meaning of the remaining field (EVEX.L'L). The functionality for the following instruction classes are:

  - Broadcasting a single element across the destination vector register: this applies to the instruction class with Load+Op semantic where one of the source operand is from memory.
  - Redirect L'L field (P[22:21]) as static rounding control for floating-point instructions with rounding semantic. Static rounding control overrides MXCSR.RC field and implies "Suppress all exceptions" (SAE).
  - Enable SAE for floating -point instructions with arithmetic semantic that is not rounding.
  - For instruction classes outside of the afore-mentioned three classes, setting EVEX.b will cause #UD.
- Vector length/rounding control specifier: P[22:21] can serve one of three options.
  - Vector length information for packed vector instructions.
  - Ignored for instructions operating on vector register content as a single data element.
  - Rounding control for floating-point instructions that have a rounding semantic and whose source and destination operands are all vector registers.

2.7.2 Register Specifier Encoding and EVEX

EVEX-encoded instruction can access 8 opmask registers, 16 general-purpose registers and 32 vector registers in 64-bit mode (8 general-purpose registers and 8 vector registers in non-64-bit modes). EVEX-encoding can support instruction syntax that access up to 4 instruction operands. Normal memory addressing modes and VSIB memory addressing are supported with EVEX prefix encoding. The mapping of register operands used by various instruction syntax and memory addressing in 64-bit mode are shown in Table 2-31. Opmask register encoding is described in Section 2.7.3.

Table 2-31. 32-Register Support in 64-bit Mode Using EVEX with Embedded REX Bits

	4[1]	3	[2:0]	Reg. Type	Common Usages
REG	EVEX.R'	REX.R	modrm.reg	GPR, Vector	Destination or Source
VVVV	EVEX.V'	EVEX.vvvv		GPR, Vector	2ndSource or Destination
RM	EVEX.X	EVEX.B	modrm.r/m	GPR, Vector	1st Source or Destination
BASE	0	EVEX.B	modrm.r/m	GPR	memory addressing
INDEX	0	EVEX.X	sib.index	GPR	memory addressing
VIDX	EVEX.V'	EVEX.X	sib.index	Vector	VSIB memory addressing

NOTES:

1. Not applicable for accessing general purpose registers.

The mapping of register operands used by various instruction syntax and memory addressing in 32-bit modes are shown in Table 2-32.

Table 2-32. EVEX Encoding Register Specifiers in 32-bit Mode

	[2:0]	Reg. Type	Common Usages
REG	modrm.reg	GPR, Vector	Destination or Source
VVVV	EVEX.vvv	GPR, Vector	2nd Source or Destination
RM	modrm.r/m	GPR, Vector	1st Source or Destination
BASE	modrm.r/m	GPR	Memory Addressing
INDEX	sib.index	GPR	Memory Addressing
VIDX	sib.index	Vector	VSIB Memory Addressing

2.7.3 Opmask Register Encoding

There are eight opmask registers, k0-k7. Opmask register encoding falls into two categories:

- Opmask registers that are the source or destination operands of an instruction treating the content of opmask register as a scalar value, are encoded using the VEX prefix scheme. It can support up to three operands using standard modR/M byte's reg field and rm field and VEX.vvvv. Such a scalar opmask instruction does not support conditional update of the destination operand.
- An opmask register providing conditional processing and/or conditional update of the destination register of a vector instruction is encoded using EVEX.aaa field (see Section 2.7.4).
- An opmask register serving as the destination or source operand of a vector instruction is encoded using standard modR/M byte's reg field and rm fields.

Table 2-33. Opmask Register Specifier Encoding

	[2:0]	Register Access	Common Usages
REG	modrm.reg	k0-k7	Source
VVVV	VEX.vvvv	k0-k7	2nd Source
RM	modrm.r/m	k0-7	1st Source
{k1}	EVEX.aaa	k0[1]-k7	Opmask

NOTES:

1. Instructions that overwrite the conditional mask in opmask do not permit using k0 as the embedded mask.

2.7.4 Masking Support in EVEX

EVEX can encode an opmask register to conditionally control per-element computational operation and updating of result of an instruction to the destination operand. The predicate operand is known as the opmask register. The EVEX.aaa field, P[18:16] of the EVEX prefix, is used to encode one out of a set of eight 64-bit architectural registers. Note that from this set of 8 architectural registers, only k1 through k7 can be addressed as predicate operands. k0 can be used as a regular source or destination but cannot be encoded as a predicate operand.

AVX-512 instructions support two types of masking with EVEX.z bit (P[23]) controlling the type of masking:

- Merging-masking, which is the default type of masking for EVEX-encoded vector instructions, preserves the old value of each element of the destination where the corresponding mask bit has a 0. It corresponds to the case of EVEX.z = 0.
- Zeroing-masking, is enabled by having the EVEX.z bit set to 1. In this case, an element of the destination is set to 0 when the corresponding mask bit has a 0 value.

AVX-512 Foundation instructions can be divided into the following groups:

- Instructions which support "zeroing-masking".
  - Also allow merging-masking.
- Instructions which require aaa = 000.
  - Do not allow any form of masking.
- Instructions which allow merging-masking but do not allow zeroing-masking.
  - Require EVEX.z to be set to 0.
  - This group is mostly composed of instructions that write to memory.
- Instructions which require aaa <> 000 do not allow EVEX.z to be set to 1.
  - Allow merging-masking and do not allow zeroing-masking, e.g., gather instructions.

2.7.5 Compressed Displacement (disp8*N) Support in EVEX

For memory addressing using disp8 form, EVEX-encoded instructions always use a compressed displacement scheme by multiplying disp8 in conjunction with a scaling factor N that is determined based on the vector length, the value of EVEX.b bit (embedded broadcast) and the input element size of the instruction. In general, the factor N corresponds to the number of bytes characterizing the internal memory operation of the input operand (e.g., 64 when the accessing a full 512-bit memory vector). The scale factor N is listed in Table 2-34 and Table 2-35 below, where EVEX encoded instructions are classified using the **tupletype** attribute. The scale factor N of each tupletype is listed based on the vector length (VL) and other factors affecting it.

Table 2-34 covers EVEX-encoded instructions which has a load semantic in conjunction with additional computational or data element movement operation, operating either on the full vector or half vector (due to conversion of numerical precision from a wider format to narrower format). EVEX.b is supported for such instructions for data element sizes which are either dword or qword (see Section 2.7.11).

EVEX-encoded instruction that are pure load/store, and "Load+op" instruction semantic that operate on data element size less then dword do not support broadcasting using EVEX.b. These are listed in Table 2-35. Table 2-35 also includes many broadcast instructions which perform broadcast using a subset of data elements without using EVEX.b. These instructions and a few data element size conversion instruction are covered in Table 2-35. Instruction classified in Table 2-35 do not use EVEX.b and EVEX.b must be 0, otherwise #UD will occur.

The tupletype will be referenced in the instruction operand encoding table in the reference page of each instruction, providing the cross reference for the scaling factor N to encoding memory addressing operand.

Note that the disp8*N rules still apply when using 16b addressing.

Table 2-34. Compressed Displacement (DISP8*N) Affected by Embedded Broadcast

TupleType	EVEX.b	InputSize	EVEX.W	Broadcast	N (VL=128)	N (VL=256)	N (VL= 512)	Comment
Full	0	32bit	0	none	16	32	64	Load+Op (Full Vector Dword/Qword)
	1	32bit	0	{1tox}	4	4	4	
	0	64bit	1	none	16	32	64	
	1	64bit	1	{1tox}	8	8	8	
Half	0	32bit	0	none	8	16	32	Load+Op (Half Vector)
	1	32bit	0	{1tox}	4	4	4	

Table 2-35. EVEX DISP8*N for Instructions Not Affected by Embedded Broadcast

TupleType	InputSize	EVEX.W	N (VL= 128)	N (VL= 256)	N (VL= 512)	Comment
Full Mem	N/A	N/A	16	32	64	Load/store or subDword full vector
Tuple1 Scalar	8bit	N/A	1	1	1	1Tuple
	16bit	N/A	2	2	2	
	32bit	0	4	4	4	
	64bit	1	8	8	8	
Tuple1 Fixed	32bit	N/A	4	4	4	1 Tuple, memsize not affected by EVEX.W
	64bit	N/A	8	8	8	
Tuple2	32bit	0	8	8	8	Broadcast (2 elements)
	64bit	1	NA	16	16	
Tuple4	32bit	0	NA	16	16	Broadcast (4 elements)
	64bit	1	NA	NA	32	
Tuple8	32bit	0	NA	NA	32	Broadcast (8 elements)
Half Mem	N/A	N/A	8	16	32	SubQword Conversion
Quarter Mem	N/A	N/A	4	8	16	SubDword Conversion

Table 2-35. EVEX DISP8*N for Instructions Not Affected by Embedded Broadcast (Contd.)

TupleType	InputSize	EVEX.W	N (VL= 128)	N (VL= 256)	N (VL= 512)	Comment
Eighth Mem	N/A	N/A	2	4	8	SubWord Conversion
Mem128	N/A	N/A	16	16	16	Shift count from memory
MOVDDUP	N/A	N/A	8	32	64	VMOVDDUP

2.7.6 EVEX Encoding of Broadcast/Rounding/SAE Support

EVEX.b can provide three types of encoding context, depending on the instruction classes:

- Embedded broadcasting of one data element from a source memory operand to the destination for vector instructions with "load+op" semantic.
- Static rounding control overriding MXCSR.RC for floating-point instructions with rounding semantic.
- "Suppress All exceptions" (SAE) overriding MXCSR mask control for floating-point arithmetic instructions that do not have rounding semantic.

2.7.7 Embedded Broadcast Support in EVEX

EVEX encodes an embedded broadcast functionality that is supported on many vector instructions with 32-bit (double word or single precision floating-point) and 64-bit data elements, and when the source operand is from memory. EVEX.b (P[20]) bit is used to enable broadcast on load-op instructions. When enabled, only one element is loaded from memory and broadcasted to all other elements instead of loading the full memory size.

The following instruction classes do not support embedded broadcasting:

- Instructions with only one scalar result is written to the vector destination.
- Instructions with explicit broadcast functionality provided by its opcode.
- Instruction semantic is a pure load or a pure store operation.

2.7.8 Static Rounding Support in EVEX

Static rounding control embedded in the EVEX encoding system applies only to register-to-register flavor of floating-point instructions with rounding semantic at two distinct vector lengths: (i) scalar, (ii) 512-bit. In both cases, the field EVEX.L'L expresses rounding mode control overriding MXCSR.RC if EVEX.b is set. When EVEX.b is set, "suppress all exceptions" is implied. The processor behaves as if all MXCSR masking controls are set.

2.7.9 SAE Support in EVEX

The EVEX encoding system allows arithmetic floating-point instructions without rounding semantic to be encoded with the SAE attribute. This capability applies to scalar and 512-bit vector lengths, register-to-register only, by setting EVEX.b. When EVEX.b is set, "suppress all exceptions" is implied. The processor behaves as if all MXCSR masking controls are set.

2.7.10 Vector Length Orthogonality

The architecture of EVEX encoding scheme can support SIMD instructions operating at multiple vector lengths. Many AVX-512 Foundation instructions operate at 512-bit vector length. The vector length of EVEX encoded vector instructions are generally determined using the L'L field in EVEX prefix, except for 512-bit floating-point, reg-reg instructions with rounding semantic. The table below shows the vector length corresponding to various values of the L'L bits. When EVEX is used to encode scalar instructions, L'L is generally ignored.

When EVEX.b bit is set for a register-register instructions with floating-point rounding semantic, the same two bits P2[6:5] specifies rounding mode for the instruction, with implied SAE behavior. The mapping of different instruction classes relative to the embedded broadcast/rounding/SAE control and the EVEX.L'L fields are summarized in Table 2-36.

Table 2-36. EVEX Embedded Broadcast/Rounding/SAE and Vector Length on Vector Instructions

Position	P2[4]	P2[6:5]	P2[6:5]
Broadcast/Rounding/SAE Context	**EVEX.b**	**EVEX.L'L**	**EVEX.RC**
Reg-reg, FP Instructions w/ rounding semantic or SAE	Enable static rounding control (SAE implied)	Vector length Implied (512 bit or scalar)	00b: SAE + RNE 01b: SAE + RD 10b: SAE + RU 11b: SAE + RZ
Load+op Instructions w/ memory source	Broadcast Control	00b: 128-bit 01b: 256-bit 10b: 512-bit 11b: Reserved (#UD)	NA
Other Instructions (Explicit Load/Store/Broadcast/Gather/Scatter)	Must be 0 (otherwise #UD)		NA

2.7.11 #UD Equations for EVEX

Instructions encoded using EVEX can face three types of UD conditions: state dependent, opcode independent and opcode dependent.

2.7.11.1 State Dependent #UD

In general, attempts of execute an instruction, which required OS support for incremental extended state component, will #UD if required state components were not enabled by OS. Table 2-37 lists instruction categories with respect to required processor state components. Attempts to execute a given category of instructions while enabled states were less than the required bit vector in XCR0 shown in Table 2-37 will cause #UD.

Table 2-37. OS XSAVE Enabling Requirements of Instruction Categories

Instruction Categories	Vector Register State Access	Required XCR0 Bit Vector [7:0]
Legacy SIMD prefix encoded Instructions (e.g SSE)	XMM	xxxxxx11b
VEX-encoded instructions operating on YMM	YMM	xxxxx111b
EVEX-encoded 128-bit instructions	ZMM	111xx111b
EVEX-encoded 256-bit instructions	ZMM	111xx111b
EVEX-encoded 512-bit instructions	ZMM	111xx111b
VEX-encoded instructions operating on opmask	k-reg	111xxx11b

2.7.11.2 Opcode Independent #UD

A number of bit fields in EVEX encoded instruction must obey mode-specific but opcode-independent patterns listed in Table 2-38.

Table 2-38. Opcode Independent, State Dependent EVEX Bit Fields

Position	Notation	64-bit #UD	Non-64-bit #UD
P[3]	--	if > 0	if > 0
P[10]	--	if 0	if 0
P[2:0]	EVEX.mmm	if 000b, 100b, or 111b	if 000b, 100b, or 111b
P[7 : 6]	EVEX.RX	None (valid)	None (BOUND if EVEX.RX != 11b)

2.7.11.3 Opcode Dependent #UD

This section describes legal values for the rest of the EVEX bit fields. Table 2-39 lists the #UD conditions of EVEX prefix bit fields which encodes or modifies register operands.

Table 2-39. #UD Conditions of Operand-Encoding EVEX Prefix Bit Fields

Notation	Position	Operand Encoding	64-bit #UD	Non-64-bit #UD
EVEX.R	P[7]	ModRM.reg encodes k-reg	If EVEX.R = 0	None (BOUND if EVEX.RX != 11b)
		ModRM.reg is opcode extension	None (ignored)	
		ModRM.reg encodes all other registers	None (valid)	
EVEX.X	P[6]	ModRM.r/m encodes ZMM/YMM/XMM	None (valid)	
		ModRM.r/m encodes k-reg or GPR	None (ignored)	
		ModRM.r/m without SIB/VSIB	None (ignored)	
		ModRM.r/m with SIB/VSIB	None (valid)	
EVEX.B	P[5]	ModRM.r/m encodes k-reg	None (ignored)	None (ignored)
		ModRM.r/m encodes other registers	None (valid)	
		ModRM.r/m base present	None (valid)	
		ModRM.r/m base not present	None (ignored)	
EVEX.R'	P[4]	ModRM.reg encodes k-reg or GPR	If 0	None (ignored)
		ModRM.reg is opcode extension	None (ignored)	
		ModRM.reg encodes ZMM/YMM/XMM	None (valid)	
EVEX.vvvv	P[14:11]	vvvv encodes ZMM/YMM/XMM	None (valid)	None (valid) P[14] ignored
		Otherwise	If != 1111b	If != 1111b
EVEX.V'	P[19]	Encodes ZMM/YMM/XMM	None (valid)	If 0
		Otherwise	If 0	If 0

Table 2-40 lists the #UD conditions of instruction encoding of opmask register using EVEX.aaa and EVEX.z

Table 2-40. #UD Conditions of Opmask Related Encoding Field

Notation	Position	Operand Encoding	64-bit #UD	Non-64-bit #UD
EVEX.aaa	P[18:16]	Instructions do not use opmask for conditional processing[1].	If aaa != 000b	If aaa != 000b
		Opmask used as conditional processing mask and updated at completion[2].	If aaa = 000b	If aaa = 000b;
		Opmask used as conditional processing.	None (valid[3])	None (valid[1])
EVEX.z	P[23]	Vector instruction using opmask as source or destination[4].	If EVEX.z != 0	If EVEX.z != 0
		Store instructions or gather/scatter instructions.	If EVEX.z != 0	If EVEX.z != 0
		Instructions with EVEX.aaa = 000b.	If EVEX.z != 0	If EVEX.z != 0
VEX.vvvv	Varies	K-regs are instruction operands not mask control.	If vvvv = 0xxxb	None

NOTES:

1. E.g., VPBROADCASTMxxx, VPMOVM2x, VPMOVx2M.
2. E.g., Gather/Scatter family.
3. aaa can take any value. A value of 000 indicates that there is no masking on the instruction; in this case, all elements will be processed as if there was a mask of 'all ones' regardless of the actual value in K0.
4. E.g., VFPCLASSPD/PS, VCMPB/D/Q/W family, VPMOVM2x, VPMOVx2M.

Table 2-41 lists the #UD conditions of EVEX bit fields that depends on the context of EVEX.b.

Table 2-41. #UD Conditions Dependent on EVEX.b Context

Notation	Position	Operand Encoding	64-bit #UD	Non-64-bit #UD
EVEX.L'Lb	P[22 : 20]	Reg-reg, FP instructions with rounding semantic.	None (valid[1])	None (valid[1])
		Other reg-reg, FP instructions that can cause #XM.	None (valid[2])	None (valid[2])
		Other reg-mem instructions in Table 2-34.	None (valid[3])	None (valid[3])
		Other instruction classes[4] in Table 2-35.	If EVEX.b = 1	If EVEX.b = 1

NOTES:

1. L'L specifies rounding control, see Table 2-36, supports {er} syntax.
2. L'L is ignored.
3. L'L specifies vector length, see Table 2-36, supports embedded broadcast syntax
4. L'L specifies either vector length or ignored.

2.7.12 Device Not Available

EVEX-encoded instructions follow the same rules when it comes to generating #NM (Device Not Available) exception. In particular, it is generated when CR0.TS[bit 3]= 1.

2.7.13 Scalar Instructions

EVEX-encoded scalar SIMD instructions can access up to 32 registers in 64-bit mode. Scalar instructions support masking (using the least significant bit of the opmask register), but broadcasting is not supported.

2.8 EXCEPTION CLASSIFICATIONS OF EVEX-ENCODED INSTRUCTIONS

The exception behavior of EVEX-encoded instructions can be classified into the classes shown in the rest of this section. The classification of EVEX-encoded instructions follow a similar framework as those of AVX and AVX2 instructions using the VEX prefix. Exception types for EVEX-encoded instructions are named in the style of "E##" or with a suffix "E##XX". The "##" designation generally follows that of AVX/AVX2 instructions. The majority of EVEX encoded instruction with "Load+op" semantic supports memory fault suppression, which is represented by E##. The instructions with "Load+op" semantic but do not support fault suppression are named "E##NF". A summary table of exception classes by class names are shown below.

Table 2-42. EVEX-Encoded Instruction Exception Class Summary

Exception Class	Instruction set	Mem arg	(#XM)
Type E1	Vector Moves/Load/Stores	Explicitly aligned, w/ fault suppression	None
Type E1NF	Vector Non-temporal Stores	Explicitly aligned, no fault suppression	None
Type E2	FP Vector Load+op	Support fault suppression	Yes
Type E2NF	FP Vector Load+op	No fault suppression	Yes
Type E3	FP Scalar/Partial Vector, Load+Op	Support fault suppression	Yes
Type E3NF	FP Scalar/Partial Vector, Load+Op	No fault suppression	Yes
Type E4	Integer Vector Load+op	Support fault suppression	No
Type E4NF	Integer Vector Load+op	No fault suppression	No
Type E5	Legacy-like Promotion	Varies, Support fault suppression	No
Type E5NF	Legacy-like Promotion	Varies, No fault suppression	No

Table 2-42. EVEX-Encoded Instruction Exception Class Summary (Contd.)

Exception Class	Instruction set	Mem arg	(#XM)
Type E6	Post AVX Promotion	Varies, w/ fault suppression	No
Type E6NF	Post AVX Promotion	Varies, no fault suppression	No
Type E7NM	Register-to-register op	None	None
Type E9NF	Miscellaneous 128-bit	Vector-length Specific, no fault suppression	None
Type E10	Non-XF Scalar	Vector Length ignored, w/ fault suppression	None
Type E10NF	Non-XF Scalar	Vector Length ignored, no fault suppression	None
Type E11	VCVTPH2PS, VCVTPS2PH	Half Vector Length, w/ fault suppression	Yes
Type E12	Gather and Scatter Family	VSIB addressing, w/ fault suppression	None
Type E12NP	Gather and Scatter Prefetch Family	VSIB addressing, w/o page fault	None

Table 2-43 lists EVEX-encoded instruction mnemonic by exception classes.

Table 2-43. EVEX Instructions in Each Exception Class

Exception Class	Instruction
Type E1	VMOVAPD, VMOVAPS, VMOVDQA32, VMOVDQA64
Type E1NF	VMOVNTDQ, VMOVNTDQA, VMOVNTPD, VMOVNTPS
Type E2	VADDPD, VADDPH, VADDPS, VCMPPD, VCMPPH, VCMPPS, VCVTDQ2PH, VCVTDQ2PS, VCVTPD2DQ, VCVTPD2PH, VCVTPD2PS, VCVTPD2QQ, VCVTPD2UQQ, VCVTPD2UDQ, VCVTPH2DQ, VCVTPH2PD, VCVTPH2QQ, VCVTPH2UDQ, VCVTPH2UQQ, VCVTPH2UW, VCVTPH2W, VCVTPS2DQ, VCVTPS2UDQS, VCVTQQ2PD, VCVTQQ2PH, VCVTQQ2PS, VCVTTPD2DQ, VCVTTPD2QQ, VCVTTPD2UDQ, VCVTTPD2UQQ, VCVTTPH2DQ, VCVTTPH2QQ, VCVTTPH2UDQ, VCVTTPH2UQQ, VCVTTPH2UW, VCVTTPH2W, VCVTTPS2DQ, VCVTTPS2UDQ, VCVTUDQ2PH, VCVTUDQ2PS, VCVTUQQ2PD, VCVTUQQ2PH, VCVTUQQ2PS, VCVTUW2PH, VCVTW2PH, VDIVPD, VDIVPH, VDIVPS, VEXP2PD, VEXP2PS, VFIXUPIMMPD, VFIXUPIMMPS, VFMADDxxxPD, VFMADDxxxPH, VFMADDxxxPS, VFMADDSUBxxxPD, VFMADDSUBxxxPH, VFMADDSUBxxxPS, VFMSUBADDxxxPD, VFMSUBADDxxxPH, VFMSUBADDxxxPS, VFMSUBxxxPD, VFMSUBxxxPH, VFMSUBxxxPS, VFNMADDxxxPD, VFNMADDxxxPH, VFNMADDxxxPS, VFNMSUBxxxPD, VFNMSUBxxxPH, VFNMSUBxxxPS, VGETEXPPD, VGETEXPPH, VGETEXPPS, VGETMANTPD, VGETMANTPH, VGETMANTPS, VGETMANTSH, VMAXPD, VMAXPH, VMAXPS, VMINPD, VMINPH, VMINPS, VMULPD, VMULPH, VMULPS, VRANGEPD, VRANGEPS, VREDUCEPD, VREDUCEPH, VREDUCEPS, VRNDSCALEPD, VRNDSCALEPH, VRNDSCALEPS, VRCP28PD, VRCP28PS, VRSQRT28PD, VRSQRT28PS, VSCALEFPD, VSCALEFPS, VSQRTPD, VSQRTPH, VSQRTPS, VSUBPD, VSUBPH, VSUBPS
Type E3	VADDSD, VADDSH, VADDSS, VCMPSD, VCMPSH, VCMPSS, VCVTPS2QQ, VCVTPS2UQQ, VCVTPS2PD, VCVTSD2SH, VCVTSD2SS, VCVTSH2SD, VCVTSH2SS, VCVTSS2SD, VCVTSS2SH, VCVTTPS2QQ, VCVTTPS2UQQ, VDIVSD, VDIVSH, VDIVSS, VFMADDxxxSD, VFMADDxxxSH, VFMADDxxxSS, VFMSUBxxxSD, VFMSUBxxxSH, VFMSUBxxxSS, VFNMADDxxxSD, VFNMADDxxxSH, VFNMADDxxxSS, VFNMSUBxxxSD, VFNMSUBxxxSH, VFNMSUBxxxSS, VFIXUPIMMSD, VFIXUPIMMSS, VGETEXPSD, VGETEXPSH, VGETEXPSS, VGETMANTSD, VGETMANTSH, VGETMANTSS, VMAXSD, VMAXSH, VMAXSS, VMINSD, VMINSH, VMINSS, VMULSD, VMULSH, VMULSS, VRANGESD, VRANGESS, VREDUCESD, VREDUCESH, VREDUCESS, VRNDSCALESD, VRNDSCALESH, VRNDSCALESS, VSCALEFSD, VSCALEFSH, VSCALEFSS, VRCP28SD, VRCP28SS, VRSQRT28SD, VRSQRT28SS, VSQRTSD, VSQRTSH, VSQRTSS, VSUBSD, VSUBSH, VSUBSS
Type E3NF	VCOMISD, VCOMISH, VCOMISS, VCVTSD2SI, VCVTSD2USI, VCVTSH2SI, VCVTSH2USI, VCVTSI2SD, VCVTSI2SH, VCVTSI2SS, VCVTSS2SI, VCVTSS2USI, VCVTTSD2SI, VCVTTSD2USI, VCVTTSH2SI, VCVTTSH2USI, VCVTTSS2SI, VCVTTSS2USI, VCVTUSI2SD, VCVTUSI2SH, VCVTUSI2SS, VUCOMISD, VUCOMISH, VUCOMISS

Table 2-43. EVEX Instructions in Each Exception Class (Contd.)

Exception Class	Instruction
Type E4	VANDPD, VANDPS, VANDNPD, VANDNPS, VBLENDMPD, VBLENDMPS, VFCMADDCPH, VFCMULCPH, VFMADDCPH, VFMULCPH, VFPCLASSPD, VFPCLASSPH, VFPCLASSPS, VORPD, VORPS, VPABSD, VPABSQ, VPADDD, VPADDQ, VPANDD, VPANDQ, VPANDND, VPANDNQ, VPBLENDMB, VPBLENDMD, VPBLENDMQ, VPBLENDMW, VPCMPD, VPCMPEQD, VPCMPEQQ, VPCMPGTD, VPCMPGTQ, VPCMPQ, VPCMPUD, VPCMPUQ, VPLZCNTD, VPLZCNTQ, VPMADD52LUQ, VPMADD52HUQ, VPMAXSD, VPMAXSQ, VPMAXUD, VPMAXUQ, VPMINSD, VPMINSQ, VPMINUD, VPMINUQ, VPMULLD, VPMULLQ, VPMULUDQ, VPMULDQ, VPORD, VPORQ, VPROLD, VPROLQ, VPROLVD, VPROLVQ, VPRORD, VPRORQ, VPRORVD, VPRORVQ, (VPSLLD, VPSLLQ, VPSRAD, VPSRAQ, VPSRAVW, VPSRAVD, VPSRAVW, VPSRAVQ, VPSRLD, VPSRLQ)[1], VPSUBD, VPSUBQ, VPSUBUSB, VPSUBUSW, VPTERNLOGD, VPTERNLOGQ, VPTESTMD, VPTESTMQ, VPTESTNMD, VPTESTNMQ, VPXORD, VPXORQ, VPSLLVD, VPSLLVQ, VRCP14PD, VRCP14PS, VRCPPH, VRSQRT14PD, VRSQRT14PS, VRSQRTPH, VXORPD, VXORPS
E4.nb[2]	VCOMPRESSPD, VCOMPRESSPS, VEXPANDPD, VEXPANDPS, VMOVDQU8, VMOVDQU16, VMOVDQU32, VMOVDQU64, VMOVUPD, VMOVUPS, VPABSB, VPABSW, VPADDB, VPADDW, VPADDSB, VPADDSW, VPADDUSB, VPADDUSW, VPAVGB, VPAVGW, VPCMPB, VPCMPEQB, VPCMPEQW, VPCMPGTB, VPCMPGTW, VPCMPW, VPCMPUB, VPCMPUW, VPCOMPRESSD, VPCOMPRESSQ, VPEXPANDD, VPEXPANDQ, VPMAXSB, VPMAXSW, VPMAXUB, VPMAXUW, VPMINSB, VPMINSW, VPMINUB, VPMINUW, VPMULHRSW, VPMULHUW, VPMULHW, VPMULLW, VPSLLVW, VPSLLW, VPSRAW, VPSRLVW, VPSRLW, VPSUBB, VPSUBW, VPSUBSB, VPSUBSW, VPTESTMB, VPTESTMW, VPTESTNMB, VPTESTNMW
Type E4NF	VALIGND, VALIGNQ, VPACKSSDW, VPACKUSDW, VPCONFLICTD, VPCONFLICTQ, VPERMD, VPERMI2D, VPERMI2PS, VPERMI2PD, VPERMI2Q, VPERMPD, VPERMPS, VPERMQ, VPERMT2D, VPERMT2PS, VPERMT2Q, VPERMT2PD, VPERMILPD, VPERMILPS, VPMULTISHIFTQB, VPSHUFD, VPUNPCKHDQ, VPUNPCKHQDQ, VPUNPCKLDQ, VPUNPCKLQDQ, VSHUFF32X4, VSHUFF64X2, VSHUFI32X4, VSHUFI64X2, VSHUFPD, VSHUFPS, VUNPCKHPD, VUNPCKHPS, VUNPCKLPD, VUNPCKLPS
E4NF.nb[2]	VDBPSADBW, VPACKSSWB, VPACKUSWB, VPALIGNR, VPMADDWD, VPMADDUBSW, VMOVSHDUP, VMOVSLDUP, VPSADBW, VPSHUFB, VPSHUFHW, VPSHUFLW, VPSLLDQ, VPSRLDQ, VPSLLW, VPSRAW, VPSRLW, (VPSLLD, VPSLLQ, VPSRAD, VPSRAQ, VPSRLD, VPSRLQ)[3], VPUNPCKHBW, VPUNPCKHWD, VPUNPCKLBW, VPUNPCKLWD, VPERMW, VPERMI2W, VPERMT2W
Type E5	PMOVSXBW, PMOVSXBW, PMOVSXBD, PMOVSXBQ, PMOVSXWD, PMOVSXWQ, PMOVSXDQ, PMOVZXBW, PMOVZXBD, PMOVZXBQ, PMOVZXWD, PMOVZXWQ, PMOVZXDQ, VCVTDQ2PD, VCVTUDQ2PD, VMOVSH, VPMOVSXxx, VPMOVZXxx,
Type E5NF	VMOVDDUP
Type E6	VBROADCASTF32X2, VBROADCASTF32X4, VBROADCASTF64X2, VBROADCASTF32X8, VBROADCASTF64X4, VBROADCASTI32X2, VBROADCASTI32X4, VBROADCASTI64X2, VBROADCASTI32X8, VBROADCASTI64X4, VBROADCASTSD, VBROADCASTSS, VFPCLASSSD, VFPCLASSSS, VPBROADCASTB, VPBROADCASTD, VPBROADCASTW, VPBROADCASTQ, VPMOVQB, VPMOVSQB, VPMOVUSQB, VPMOVQW, VPMOVSQW, VPMOVUSQW, VPMOVQD, VPMOVSQD, VPMOVUSQD, VPMOVDB, VPMOVSDB, VPMOVUSDB, VPMOVDW, VPMOVSDW, VPMOVUSDW, VPMOVWB, VPMOVSWB, VPMOVUSWB
Type E6NF	VEXTRACTF32X4, VEXTRACTF32X8, VEXTRACTF64X2, VEXTRACTF64X4, VEXTRACTI32X4, VEXTRACTI32X8, VEXTRACTI64X2, VEXTRACTI64X4, VINSERTF32X4, VINSERTF32X8, VINSERTF64X2, VINSERTF64X4, VINSERTI32X4, VINSERTI32X8, VINSERTI64X2, VINSERTI64X4, VPBROADCASTMB2Q, VPBROADCASTMW2D
Type E7NM.128[4]	VMOVHLPS, VMOVLHPS
Type E7NM.	(VPBROADCASTD, VPBROADCASTQ, VPBROADCASTB, VPBROADCASTW)[5], VPMOVB2M, VPMOVD2M, VPMOVM2B, VPMOVM2D, VPMOVM2Q, VPMOVM2W, VPMOVQ2M, VPMOVW2M
Type E9NF	VEXTRACTPS, VINSERTPS, VMOVHPD, VMOVHPS, VMOVLPD, VMOVLPS, VMOVD, VMOVQ, VMOVW, VPEXTRB, VPEXTRD, VPEXTRW, VPEXTRQ, VPINSRB, VPINSRD, VPINSRW, VPINSRQ
Type E10	VFCMADDCSH, VFMADDCSH, VFCMULCSH, VFMULCSH, VFPCLASSSH, VMOVSD, VMOVSS, VRCP14SD, VRCP14SS, VRCPSH, VRSQRT14SD, VRSQRT14SS, VRSQRTSH
Type E10NF	(VCVTSI2SD, VCVTUSI2SD)[6]
Type E11	VCVTPH2PS, VCVTPS2PH

Table 2-43. EVEX Instructions in Each Exception Class (Contd.)

Exception Class	Instruction
Type E12	VGATHERDPS, VGATHERDPD, VGATHERQPS, VGATHERQPD, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ, VPSCATTERDD, VPSCATTERDQ, VPSCATTERQD, VPSCATTERQQ, VSCATTERDPD, VSCATTERDPS, VSCATTERQPD, VSCATTERQPS
Type E12NP	VGATHERPF0DPD, VGATHERPF0DPS, VGATHERPF0QPD, VGATHERPF0QPS, VGATHERPF1DPD, VGATHERPF1DPS, VGATHERPF1QPD, VGATHERPF1QPS, VSCATTERPF0DPD, VSCATTERPF0DPS, VSCATTERPF0QPD, VSCATTERPF0QPS, VSCATTERPF1DPD, VSCATTERPF1DPS, VSCATTERPF1QPD, VSCATTERPF1QPS

NOTES:

1. Operand encoding Full tupletype with immediate.
2. Embedded broadcast is not supported with the ".nb" suffix.
3. Operand encoding Mem128 tupletype.
4. #UD raised if EVEX.L'L !=00b (VL=128).
5. The source operand is a general purpose register.
6. W0 encoding only.

2.8.1 Exceptions Type E1 and E1NF of EVEX-Encoded Instructions

EVEX-encoded instructions with memory alignment restrictions, and supporting memory fault suppression follow exception class E1.

Table 2-44. Type E1 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X	X	EVEX.512: Memory operand is not 64-byte aligned. EVEX.256: Memory operand is not 32-byte aligned. EVEX.128: Memory operand is not 16-byte aligned.
			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.

EVEX-encoded instructions with memory alignment restrictions, but do not support memory fault suppression follow exception class E1NF.

Table 2-45. Type E1NF Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X	X	EVEX.512: Memory operand is not 64-byte aligned. EVEX.256: Memory operand is not 32-byte aligned. EVEX.128: Memory operand is not 16-byte aligned.
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.

2.8.2 Exceptions Type E2 of EVEX-Encoded Instructions

EVEX-encoded vector instructions with arithmetic semantic follow exception class E2.

Table 2-46. Type E2 Class Exception Conditions

Exception	Real	Virtual 8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} or {er} not set, and CR4.OSXMMEXCPT[bit 10] = 1.

2.8.3 Exceptions Type E3 and E3NF of EVEX-Encoded Instructions

EVEX-encoded scalar instructions with arithmetic semantic that support memory fault suppression follow exception class E3.

Table 2-47. Type E3 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} or {er} not set, and CR4.OSX-MMEXCPT[bit 10] = 1.

EVEX-encoded scalar instructions with arithmetic semantic that do not support memory fault suppression follow exception class E3NF.

Table 2-48. Type E3NF Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			EVEX prefix.
	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} or {er} not set, and CR4.OSXMMEXCPT[bit 10] = 1.

2.8.4 Exceptions Type E4 and E4NF of EVEX-Encoded Instructions

EVEX-encoded vector instructions that cause no SIMD FP exception and support memory fault suppression follow exception class E4.

Table 2-49. Type E4 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41 and in E4.nb subclass (see E4.nb entries in Table 2-43). ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

EVEX-encoded vector instructions that do not cause SIMD FP exception nor support memory fault suppression follow exception class E4NF.

Table 2-50. Type E4NF Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41 and in E4NF.nb subclass (see E4NF.nb entries in Table 2-43). ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.

2.8.5 Exceptions Type E5 and E5NF

EVEX-encoded scalar/partial-vector instructions that cause no SIMD FP exception and support memory fault suppression follow exception class E5.

Table 2-51. Type E5 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

EVEX-encoded scalar/partial vector instructions that do not cause SIMD FP exception nor support memory fault suppression follow exception class E5NF.

Table 2-52. Type E5NF Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.8.6 Exceptions Type E6 and E6NF

Table 2-53. Type E6 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
			X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
			X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
Page Fault #PF(fault-code)			X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)			X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

EVEX-encoded instructions that do not cause SIMD FP exception nor support memory fault suppression follow exception class E6NF.

Table 2-54. Type E6NF Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
			X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
			X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
Page Fault #PF(fault-code)			X	X	For a page fault.
Alignment Check #AC(0)			X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.8.7 Exceptions Type E7NM

EVEX-encoded instructions that cause no SIMD FP exception and do not reference memory follow exception class E7NM.

Table 2-55. Type E7NM Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM			X	X	If CR0.TS[bit 3]=1.

2.8.8 Exceptions Type E9 and E9NF

EVEX-encoded vector or partial-vector instructions that do not cause no SIMD FP exception and support memory fault suppression follow exception class E9.

Table 2-56. Type E9 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

EVEX-encoded vector or partial-vector instructions that must be encoded with VEX.L'L = 0, do not cause SIMD FP exception nor support memory fault suppression follow exception class E9NF.

Table 2-57. Type E9NF Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.8.9 Exceptions Type E10 and E10NF

EVEX-encoded scalar instructions that ignore EVEX.L'L vector length encoding, do not cause a SIMD FP exception, and support memory fault suppression follow exception class E10.

Table 2-58. Type E10 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

EVEX-encoded scalar instructions that ignore EVEX.L'L vector length encoding, do not cause a SIMD FP exception, and do not support memory fault suppression follow exception class E10NF.

Table 2-59. Type E10NF Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	If fault suppression not set, and a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.8.10 Exceptions Type E11 (EVEX-only, Mem Arg, No AC, Floating-point Exceptions)

EVEX-encoded instructions that can cause SIMD FP exception, memory operand support fault suppression but do not cause #AC follow exception class E11.

Table 2-60. Type E11 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a EVEX prefix.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		If fault suppression not set, and an illegal address in the SS segment.
				X	If fault suppression not set, and a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		If fault suppression not set, and an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If fault suppression not set, and the memory address is in a non-canonical form.
	X	X			If fault suppression not set, and any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF (fault-code)		X	X	X	If fault suppression not set, and a page fault.
SIMD Floating-Point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception, {sae} not set, and CR4.OSXMMEX-CPT[bit 10] = 1.

2.8.11 Exceptions Type E12 and E12NP (VSIB Mem Arg, No AC, No Floating-point Exceptions)

Table 2-61. Type E12 Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met. ▪ If vvvv != 1111b.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	NA	If address size attribute is 16 bit.
	X	X	X	X	If ModR/M.mod = '11b'.
	X	X	X	X	If ModR/M.rm != '100b'.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
	X	X	X	X	If k0 is used (gather or scatter operation).
	X	X	X	X	If index = destination register (gather operation).
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
Stack, #SS(0)			X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF (fault-code)		X	X	X	For a page fault.

EVEX-encoded prefetch instructions that do not cause #PF follow exception class E12NP.

Table 2-62. Type E12NP Class Exception Conditions

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X			If EVEX prefix present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39. ▪ Opmask encoding #UD condition of Table 2-40. ▪ EVEX.b encoding #UD condition of Table 2-41. ▪ Instruction specific EVEX.L'L restriction not met.
	X	X	X	X	If preceded by a LOCK prefix (F0H).
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
	X	X	X	NA	If address size attribute is 16 bit.
	X	X	X	X	If ModR/M.mod = '11b'.
	X	X	X	X	If ModR/M.rm != '100b'.
	X	X	X	X	If any corresponding CPUID feature flag is '0'.
	X	X	X	X	If k0 is used (gather or scatter operation).
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.

2.9 EXCEPTION CLASSIFICATIONS OF OPMASK INSTRUCTIONS, TYPE K20 AND TYPE K21

The exception behavior of VEX-encoded opmask instructions are listed below.

2.9.1 Exceptions Type K20

Exception conditions of Opmask instructions that do not address memory are listed as Type K20.

Table 2-63. TYPE K20 Exception Definition (VEX-Encoded OpMask Instructions w/o Memory Arg)

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If relevant CPUID feature flag is '0'.
	X	X			If a VEX prefix is present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39.
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
			X	X	If ModRM:[7:6] != 11b.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.

2.9.2 Exceptions Type K21

Exception conditions of Opmask instructions that address memory are listed as Type K21.

Table 2-64. TYPE K21 Exception Definition (VEX-Encoded OpMask Instructions Addressing Memory)

Exception	Real	Virtual 80x86	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If relevant CPUID feature flag is '0'.
	X	X			If a VEX prefix is present.
			X	X	If CR4.OSXSAVE[bit 18]=0. If any one of following conditions applies: ▪ State requirement, Table 2-37 not met. ▪ Opcode independent #UD condition in Table 2-38. ▪ Operand encoding #UD conditions in Table 2-39.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
			X	X	If any REX, F2, F3, or 66 prefixes precede a VEX prefix.
Stack, #SS(0)	X	X	X		For an illegal address in the SS segment.
				X	If a memory address referencing the SS segment is in a non-canonical form.
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH.
Page Fault #PF(fault-code)		X	X	X	For a page fault.
Alignment Check #AC(0)		X	X	X	For 2, 4, or 8 byte memory access if alignment checking is enabled and an unaligned memory access is made while the current privilege level is 3.

2.10 INTEL® AMX INSTRUCTION EXCEPTION CLASSES

Alignment exceptions: The Intel AMX instructions that access memory will never generate #AC exceptions.

Table 2-65. Intel® AMX Exception Classes

Class	Description
AMX-E1	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111.
	• #GP based on palette and configuration checks (see pseudocode). • #GP if the memory address is in a non-canonical form.
	• #SS(0) if the memory address referencing the SS segment is in a non-canonical form.
	• #PF if a page fault occurs.
AMX-E2	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111.
	• #GP if the memory address is in a non-canonical form.
	• #SS(0) if the memory address referencing the SS segment is in a non-canonical form.
	• #PF if a page fault occurs.
AMX-E3	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #UD if not using SIB addressing. • #UD if TILES_CONFIGURED == 0. • #UD if tsrc or tdest are not valid tiles. • #UD if tsrc/tdest are ≥ palette_table[tilecfg.palette_id].max_names. • #UD if tsrc.colbytes mod 4 ≠ 0 OR tdest.colbytes mod 4 ≠ 0. • #UD if tilecfg.start_row ≥ tsrc.rows OR tilecfg.start_row ≥ tdest.rows.
	• #GP if the memory address is in a non-canonical form.
	• #SS(0) if the memory address referencing the SS segment is in a non-canonical form.
	• #PF if any memory operand causes a page fault.
	• #NM if XFD[18] == 1.

Table 2-65. Intel® AMX Exception Classes (Contd.)

Class	Description
AMX-E4	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if srcdest == src1 OR src1 == src2 OR srcdest == src2. • #UD if TILES_CONFIGURED == 0. • #UD if srcdest.colbytes mod 4 ≠ 0. • #UD if src1.colbytes mod 4 ≠ 0. • #UD if src2.colbytes mod 4 ≠ 0. • #UD if srcdest/src1/src2 are not valid tiles. • #UD if srcdest/src1/src2 are ≥ palette_table[tilecfg.palette_id].max_names. • #UD if srcdest.colbytes ≠ src2.colbytes. • #UD if srcdest.rows ≠ src1.rows. • #UD if src1.colbytes / 4 ≠ src2.rows. • #UD if srcdest.colbytes > tmul_maxn. • #UD if src2.colbytes > tmul_maxn. • #UD if src1.colbytes/4 > tmul_maxk. • #UD if src2.rows > tmul_maxk.
	• #NM if XFD[18] == 1.
AMX-E5	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #UD if TILES_CONFIGURED == 0. • #UD if tdest is not a valid tile. • #UD if tdest is ≥ palette_table[tilecfg.palette_id].max_names.
	• #NM if XFD[18] == 1.
AMX-E6	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111.

6. Updates to Chapter 3, Volume 2A

Change bars and violet 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 Section 3.1.1.3, "Instruction Column in the Opcode Summary Table," to add a definition for "/ib" and updated the imm8 definition to include details when individual bits are used.
- Updated the BSWAP and BTC instructions to correct a typo. Previously, these instructions indicated that the use of REX.R permits access to additional registers (R8-R15), but it is REX.B that permits this. In BSWAP, REX.R has no effect, and it is ignored. In BTC, REX.B extends the ModR/M register field, and REX.R extends the register in the reg field.
- Updated the CPUID instruction:
  - — CPUID Subleaf 12H with ECX = 2 or higher was updated to expand the encoding of the Intel SGX memory types; these changes are marked by change bars and violet text.
  - — Updated all instances of "COS" to "CLOS" in this instruction; these changes are not marked with change bars or violet text.
- Updated the CVTTSS2SI instruction to correct the heading title.
- Updated the ENDBR32 and ENDBR64 instructions:
  - — Added the following statement to the descriptions: "This opcode is a NOP when CET indirect branch tracking is not enabled and on processors that do not support CET."
  - — An exception was added for both instructions to clarify that the instruction will #UD if the LOCK prefix is used.
- Updated the IMUL instruction to add the REX + <opcode> form.

CHAPTER 3
INSTRUCTION SET REFERENCE, A-L

This chapter describes the instruction set for the Intel 64 and IA-32 architectures (A-L) in IA-32e, protected, virtual-8086, and real-address modes of operation. The set includes general-purpose, x87 FPU, MMX, SSE/SSE2/SSE3/SSSE3/SSE4, AESNI/PCLMULQDQ, AVX, and system instructions. See also Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B; Chapter 5, "Instruction Set Reference, V," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C; and Chapter 6, "Instruction Set Reference, W-Z," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

For each instruction, each operand combination is described. A description of the instruction and its operand, an operational description, a description of the effect of the instructions on flags in the EFLAGS register, and a summary of exceptions that can be generated are also provided.

3.1 INTERPRETING THE INSTRUCTION REFERENCE PAGES

This section describes the format of information contained in the instruction reference pages in this chapter. It explains notational conventions and abbreviations used in these sections.

3.1.1 Instruction Format

The following is an example of the format used for each instruction description in this chapter. The heading below introduces the example. The table below provides an example summary table.

CMC—Complement Carry Flag [this is an example]

Opcode	Instruction	Op/En	64/32-bit Mode	CPUID Feature Flag	Description
F5	CMC	ZO	V/V	N/A	Complement carry flag.

Instruction Operand Encoding

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

3.1.1.1 Opcode Column in the Instruction Summary Table (Instructions without VEX Prefix)

The "Opcode" column in the table above shows the object code produced for each form of the instruction. When possible, codes are given as hexadecimal bytes in the same order in which they appear in memory. Definitions of entries other than hexadecimal bytes are as follows:

- **NP** — Indicates the use of 66/F2/F3 prefixes (beyond those already part of the instructions opcode) are not allowed with the instruction. Such use will either cause an invalid-opcode exception (#UD) or result in the encoding for a different instruction.
- **NFx** — Indicates the use of F2/F3 prefixes (beyond those already part of the instructions opcode) are not allowed with the instruction. Such use will either cause an invalid-opcode exception (#UD) or result in the encoding for a different instruction.
- **REX.W** — Indicates the use of a REX prefix that affects operand size or instruction semantics. The ordering of the REX prefix and other optional/mandatory instruction prefixes are discussed Chapter 2. Note that REX prefixes that promote legacy instructions to 64-bit behavior are not listed explicitly in the opcode column.
- **/digit** — A digit between 0 and 7 indicates that the ModR/M byte of the instruction uses only the r/m (register or memory) operand. The reg field contains the digit that provides an extension to the instruction's opcode.
- **/r —** Indicates that the ModR/M byte of the instruction contains a register operand and an r/m operand.
- **cb, cw, cd, cp, co, ct —** A 1-byte (cb), 2-byte (cw), 4-byte (cd), 6-byte (cp), 8-byte (co) or 10-byte (ct) value following the opcode. This value is used to specify a code offset and possibly a new value for the code segment register.
- **ib, iw, id, io —** A 1-byte (ib), 2-byte (iw), 4-byte (id) or 8-byte (io) immediate operand to the instruction that follows the opcode, ModR/M bytes or scale-indexing bytes. The opcode determines if the operand is a signed value. All words, doublewords, and quadwords are given with the low-order byte first.
- **+rb, +rw, +rd, +ro —** Indicated the lower 3 bits of the opcode byte is used to encode the register operand without a modR/M byte. The instruction lists the corresponding hexadecimal value of the opcode byte with low 3 bits as 000b. In non-64-bit mode, a register code, from 0 through 7, is added to the hexadecimal value of the opcode byte. In 64-bit mode, indicates the four bit field of REX.b and opcode[2:0] field encodes the register operand of the instruction. "+ro" is applicable only in 64-bit mode. See Table 3-1 for the codes.
- +**i —** A number used in floating-point instructions when one of the operands is ST(i) from the FPU register stack. The number i (which can range from 0 to 7) is added to the hexadecimal byte given at the left of the plus sign to form a single opcode byte.

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
AL	None	0	AX	None	0	EAX	None	0	RAX	None	0
CL	None	1	CX	None	1	ECX	None	1	RCX	None	1
DL	None	2	DX	None	2	EDX	None	2	RDX	None	2
BL	None	3	BX	None	3	EBX	None	3	RBX	None	3
AH	Not encodable (N.E.)	4	SP	None	4	ESP	None	4	N/A	N/A	N/A
CH	N.E.	5	BP	None	5	EBP	None	5	N/A	N/A	N/A
DH	N.E.	6	SI	None	6	ESI	None	6	N/A	N/A	N/A
BH	N.E.	7	DI	None	7	EDI	None	7	N/A	N/A	N/A
SPL	Yes	4	SP	None	4	ESP	None	4	RSP	None	4
BPL	Yes	5	BP	None	5	EBP	None	5	RBP	None	5

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro (Contd.)

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
SIL	Yes	6	SI	None	6	ESI	None	6	RSI	None	6
DIL	Yes	7	DI	None	7	EDI	None	7	RDI	None	7
Registers R8 - R15 (see below): Available in 64-Bit Mode Only											
R8B	Yes	0	R8W	Yes	0	R8D	Yes	0	R8	Yes	0
R9B	Yes	1	R9W	Yes	1	R9D	Yes	1	R9	Yes	1
R10B	Yes	2	R10W	Yes	2	R10D	Yes	2	R10	Yes	2
R11B	Yes	3	R11W	Yes	3	R11D	Yes	3	R11	Yes	3
R12B	Yes	4	R12W	Yes	4	R12D	Yes	4	R12	Yes	4
R13B	Yes	5	R13W	Yes	5	R13D	Yes	5	R13	Yes	5
R14B	Yes	6	R14W	Yes	6	R14D	Yes	6	R14	Yes	6
R15B	Yes	7	R15W	Yes	7	R15D	Yes	7	R15	Yes	7

3.1.1.2 Opcode Column in the Instruction Summary Table (Instructions with VEX prefix)

In the Instruction Summary Table, the Opcode column presents each instruction encoded using the VEX prefix in following form (including the modR/M byte if applicable, the immediate byte if applicable):

VEX.[128,256].[66,F2,F3].0F/0F3A/0F38.[W0,W1] opcode [/r] [/ib,/is4]

- **VEX** — Indicates the presence of the VEX prefix is required. The VEX prefix can be encoded using the three-byte form (the first byte is C4H), or using the two-byte form (the first byte is C5H). The two-byte form of VEX only applies to those instructions that do not require the following fields to be encoded: VEX.mmmmm, VEX.W, VEX.X, VEX.B. Refer to Section 2.3 for more detail on the VEX prefix.

  The encoding of various sub-fields of the VEX prefix is described using the following notations:

  — **128,256:** VEX.L field can be 0 (denoted by VEX.128, VEX.L0, or VEX.LZ) or 1 (denoted by VEX.256 or VEX.L1). The VEX.L field can be encoded using either the 2-byte or 3-byte form of the VEX prefix. The presence of the notation VEX.256 or VEX.128 in the opcode column should be interpreted as follows:

  - If VEX.256 is present in the opcode column: The semantics of the instruction must be encoded with VEX.L = 1. An attempt to encode this instruction with VEX.L= 0 can result in one of two situations: (a) if VEX.128 version is defined, the processor will behave according to the defined VEX.128 behavior; (b) an #UD occurs if there is no VEX.128 version defined.
  - If VEX.128 is present in the opcode column but there is no VEX.256 version defined for the same opcode byte: Two situations apply: (a) For VEX-encoded, 128-bit SIMD integer instructions, software must encode the instruction with VEX.L = 0. The processor will treat the opcode byte encoded with VEX.L= 1 by causing an #UD exception; (b) For VEX-encoded, 128-bit packed floating-point instructions, software must encode the instruction with VEX.L = 0. The processor will treat the opcode byte encoded with VEX.L= 1 by causing an #UD exception (e.g., VMOVLPS).
  - If VEX.L0 or VEX.L1 is present in the opcode column: The specified VEX.L value is required for encoding this instruction but does not have the connotation of specifying vector length.
  - If VEX.LIG is present in the opcode column: The VEX.L value is ignored. This generally applies to VEX-encoded scalar SIMD floating-point instructions. Scalar SIMD floating-point instruction can be distinguished from the mnemonic of the instruction. Generally, the last two letters of the instruction mnemonic would be either "SS", "SD", or "SI" for SIMD floating-point conversion instructions.
  - If VEX.LZ is present in the opcode column: The VEX.L must be encoded to be 0B, an #UD occurs if VEX.L is not zero.

— **66,F2,F3:** The presence or absence of these values map to the VEX.pp field encodings. If absent, this corresponds to VEX.pp=00B. If present, the corresponding VEX.pp value affects the "opcode" byte in the same way as if a SIMD prefix (66H, F2H or F3H) does to the ensuing opcode byte. Thus a non-zero encoding of VEX.pp may be considered as an implied 66H/F2H/F3H prefix. The VEX.pp field may be encoded using either the 2-byte or 3-byte form of the VEX prefix.

— **0F,0F3A,0F38:** The presence maps to a valid encoding of the VEX.mmmmm field. Only three encoded values of VEX.mmmmm are defined as valid, corresponding to the escape byte sequence of 0FH, 0F3AH, and 0F38H. The effect of a valid VEX.mmmmm encoding on the ensuing opcode byte is same as if the corresponding escape byte sequence on the ensuing opcode byte for non-VEX encoded instructions. Thus a valid encoding of VEX.mmmmm may be consider as an implies escape byte sequence of either 0FH, 0F3AH or 0F38H. The VEX.mmmmm field must be encoded using the 3-byte form of VEX prefix.

— **0F,0F3A,0F38 and 2-byte/3-byte VEX:** The presence of 0F3A and 0F38 in the opcode column implies that opcode can only be encoded by the three-byte form of VEX. The presence of 0F in the opcode column does not preclude the opcode to be encoded by the two-byte of VEX if the semantics of the opcode does not require any subfield of VEX not present in the two-byte form of the VEX prefix.

— **W0:** VEX.W=0.

— **W1:** VEX.W=1.

— The presence of W0/W1 in the opcode column applies to two situations: (a) it is treated as an extended opcode bit, (b) the instruction semantics support an operand size promotion to 64-bit of a general-purpose register operand or a 32-bit memory operand. The presence of W1 in the opcode column implies the opcode must be encoded using the 3-byte form of the VEX prefix. The presence of W0 in the opcode column does not preclude the opcode to be encoded using the C5H form of the VEX prefix, if the semantics of the opcode does not require other VEX subfields not present in the two-byte form of the VEX prefix. Please see Section 2.3 on the subfield definitions within VEX.

— **WIG:** can use C5H form (if not requiring VEX.mmmmm) or VEX.W value is ignored in the C4H form of VEX prefix.

— If WIG is present, the instruction may be encoded using either the two-byte form or the three-byte form of VEX. When encoding the instruction using the three-byte form of VEX, the value of VEX.W is ignored.

- **opcode** — Instruction opcode.
- **/is4** — An 8-bit immediate byte is present containing a source register specifier in either imm8[7:4] (for 64-bit mode) or imm8[6:4] (for 32-bit mode), and instruction-specific payload in imm8[3:0].
- In general, the encoding o f VEX.R, VEX.X, VEX.B field are not shown explicitly in the opcode column. The encoding scheme of VEX.R, VEX.X, VEX.B fields must follow the rules defined in Section 2.3.

EVEX.[128,256,512,LLIG].[66,F2,F3].0F/0F3A/0F38.[W0,W1,WIG] opcode [/r] [/ib]

- **EVEX** — The EVEX prefix is encoded using the four-byte form (the first byte is 62H). Refer to Section 2.7.1 for more detail on the EVEX prefix.

  The encoding of various sub-fields of the EVEX prefix is described using the following notations:

  — **128, 256, 512, LLIG:** This corresponds to the vector length; three values are allowed by EVEX: 512-bit, 256-bit and 128-bit. Alternatively, vector length is ignored (LIG) for certain instructions; this typically applies to scalar instructions operating on one data element of a vector register.

  — **66,F2,F3:** The presence of these value maps to the EVEX.pp field encodings. The corresponding VEX.pp value affects the "opcode" byte in the same way as if a SIMD prefix (66H, F2H or F3H) does to the ensuing opcode byte. Thus a non-zero encoding of VEX.pp may be considered as an implied 66H/F2H/F3H prefix.

  — **0F,0F3A,0F38:** The presence maps to a valid encoding of the EVEX.mmm field. Only three encoded values of EVEX.mmm are defined as valid, corresponding to the escape byte sequence of 0FH, 0F3AH, and 0F38H. The effect of a valid EVEX.mmm encoding on the ensuing opcode byte is the same as if the corresponding escape byte sequence on the ensuing opcode byte for non-EVEX encoded instructions. Thus a valid encoding of EVEX.mmm may be considered as an implied escape byte sequence of either 0FH, 0F3AH or 0F38H.

  — **W0:** EVEX.W=0.

  - **W1:** EVEX.W=1.
  - **WIG:** EVEX.W bit ignored
- **opcode** — Instruction opcode.
- In general, the encoding of EVEX.R and R', EVEX.X and X', and EVEX.B and B' fields are not shown explicitly in the opcode column.

NOTE

Previously, the terms NDS, NDD, and DDS were used in instructions with an EVEX (or VEX) prefix. These terms indicated that the vvvv field was valid for encoding, and specified register usage. These terms are no longer necessary and are redundant with the instruction operand encoding tables provided with each instruction. The instruction operand encoding tables give explicit details on all operands, indicating where every operand is stored and if they are read or written. If vvvv is not listed as an operand in the instruction operand encoding table, then EVEX (or VEX) vvvv must be 0b1111.

3.1.1.3 Instruction Column in the Opcode Summary Table

The "Instruction" column gives the syntax of the instruction statement as it would appear in an ASM386 program. The following is a list of the symbols used to represent operands in the instruction statements:

- **rel8** — A relative address in the range from 128 bytes before the end of the instruction to 127 bytes after the end of the instruction.
- **rel16, rel32** — A relative address within the same code segment as the instruction assembled. The rel16 symbol applies to instructions with an operand-size attribute of 16 bits; the rel32 symbol applies to instructions with an operand-size attribute of 32 bits.
- **ptr16:16, ptr16:32** — A far pointer, typically to a code segment different from that of the instruction. The notation *16:16* indicates that the value of the pointer has two parts. The value to the left of the colon is a 16-bit selector or value destined for the code segment register. The value to the right corresponds to the offset within the destination segment. The ptr16:16 symbol is used when the instruction's operand-size attribute is 16 bits; the ptr16:32 symbol is used when the operand-size attribute is 32 bits.
- **r8** — One of the byte general-purpose registers: AL, CL, DL, BL, AH, CH, DH, BH, BPL, SPL, DIL, and SIL; or one of the byte registers (R8B - R15B) available when using REX.R and 64-bit mode.
- **r16** — One of the word general-purpose registers: AX, CX, DX, BX, SP, BP, SI, DI; or one of the word registers (R8-R15) available when using REX.R and 64-bit mode.
- **r32** — One of the doubleword general-purpose registers: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI; or one of the doubleword registers (R8D - R15D) available when using REX.R in 64-bit mode.
- **r64** — One of the quadword general-purpose registers: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8–R15. These are available when using REX.R and 64-bit mode.
- **imm8** — An immediate byte value. The imm8 symbol can be a signed number between –128 and +127 inclusive; an unsigned number between 0 and 255 inclusive; or a bitmap when an instruction uses its individual bits. For instructions in which imm8 is combined with a word or doubleword operand, the immediate value is sign-extended to form a word or doubleword. The upper byte of the word is filled with the topmost bit of the immediate value.
- **imm16** — An immediate word value used for instructions whose operand-size attribute is 16 bits. This is a number between –32,768 and +32,767 inclusive.
- **imm32** — An immediate doubleword value used for instructions whose operand-size attribute is 32 bits. It allows the use of a number between +2,147,483,647 and –2,147,483,648 inclusive.
- **imm64** — An immediate quadword value used for instructions whose operand-size attribute is 64 bits. The value allows the use of a number between +9,223,372,036,854,775,807 and –9,223,372,036,854,775,808 inclusive.
- **/ib** — A single-byte value.

- **r/m8** — A byte operand that is either the contents of a byte general-purpose register (AL, CL, DL, BL, AH, CH, DH, BH, BPL, SPL, DIL, and SIL) or a byte from memory. Byte registers R8B - R15B are available using REX.R in 64-bit mode.
- **r/m16** — A word general-purpose register or memory operand used for instructions whose operand-size attribute is 16 bits. The word general-purpose registers are: AX, CX, DX, BX, SP, BP, SI, DI. The contents of memory are found at the address provided by the effective address computation. Word registers R8W - R15W are available using REX.R in 64-bit mode.
- **r/m32** — A doubleword general-purpose register or memory operand used for instructions whose operand-size attribute is 32 bits. The doubleword general-purpose registers are: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. The contents of memory are found at the address provided by the effective address computation. Doubleword registers R8D - R15D are available when using REX.R in 64-bit mode.
- **r/m64** — A quadword general-purpose register or memory operand used for instructions whose operand-size attribute is 64 bits when using REX.W. Quadword general-purpose registers are: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8–R15; these are available only in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **reg** — A general-purpose register used for instructions when the width of the register does not matter to the semantics of the operation of the instruction. The register can be r16, r32, or r64.
- **m** — A 16-, 32- or 64-bit operand in memory.
- **m8** — A byte operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. In 64-bit mode, it is pointed to by the RSI or RDI registers.
- **m16** — A word operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. This nomenclature is used only with the string instructions.
- **m32** — A doubleword operand in memory. The contents of memory are found at the address provided by the effective address computation.
- **m64** — A memory quadword operand in memory.
- **m128** — A memory double quadword operand in memory.
- **m16:16, m16:32 & m16:64** — A memory operand containing a far pointer composed of two numbers. The number to the left of the colon corresponds to the pointer's segment selector. The number to the right corresponds to its offset.
- **m16&32, m16&16, m32&32, m16&64** — A memory operand consisting of data item pairs whose sizes are indicated on the left and the right side of the ampersand. All memory addressing modes are allowed. The m16&16 and m32&32 operands are used by the BOUND instruction to provide an operand containing an upper and lower bounds for array indices. The m16&32 operand is used by LIDT and LGDT to provide a word with which to load the limit field, and a doubleword with which to load the base field of the corresponding GDTR and IDTR registers. The m16&64 operand is used by LIDT and LGDT in 64-bit mode to provide a word with which to load the limit field, and a quadword with which to load the base field of the corresponding GDTR and IDTR registers.
- **m80bcd**— A Binary Coded Decimal (BCD) operand in memory, 80 bits.
- **moffs8, moffs16, moffs32, moffs64** — A simple memory variable (memory offset) of type byte, word, or doubleword used by some variants of the MOV instruction. The actual address is given by a simple offset relative to the segment base. No ModR/M byte is used in the instruction. The number shown with moffs indicates its size, which is determined by the address-size attribute of the instruction.
- **Sreg** — A segment register. The segment register bit assignments are ES = 0, CS = 1, SS = 2, DS = 3, FS = 4, and GS = 5.
- **m32fp, m64fp, m80fp** — A single precision, double precision, and double extended-precision (respectively) floating-point operand in memory. These symbols designate floating-point values that are used as operands for x87 FPU floating-point instructions.
- **m16int, m32int, m64int** — A word, doubleword, and quadword integer (respectively) operand in memory. These symbols designate integers that are used as operands for x87 FPU integer instructions.
- **ST or ST(0)** — The top element of the FPU register stack.
- **ST(i)** — The i^{th} element from the top of the FPU register stack (*i* := 0 through 7).
- **mm** — An MMX register. The 64-bit MMX registers are: MM0 through MM7.

- **mm/m32** — The low order 32 bits of an MMX register or a 32-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- **mm/m64** — An MMX register or a 64-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- **xmm** — An XMM register. The 128-bit XMM registers are: XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode.
- **xmm/m32**— An XMM register or a 32-bit memory operand. The 128-bit XMM registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **xmm/m64** — An XMM register or a 64-bit memory operand. The 128-bit SIMD floating-point registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **xmm/m128** — An XMM register or a 128-bit memory operand. The 128-bit XMM registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **<XMM0>**— Indicates implied use of the XMM0 register.

  When there is ambiguity, xmm1 indicates the first source operand using an XMM register and xmm2 the second source operand using an XMM register.

  Some instructions use the XMM0 register as the third source operand, indicated by <XMM0>. The use of the third XMM register operand is implicit in the instruction encoding and does not affect the ModR/M encoding.
- **ymm** — A YMM register. The 256-bit YMM registers are: YMM0 through YMM7**;** YMM8 through YMM15 are available in 64-bit mode.
- **m256** — A 32-byte operand in memory. This nomenclature is used only with AVX instructions.
- **ymm/m256** — A YMM register or 256-bit memory operand.
- **<YMM0>**— Indicates use of the YMM0 register as an implicit argument.
- **bnd** — A 128-bit bounds register. BND0 through BND3.
- **mib** — A memory operand using SIB addressing form, where the index register is not used in address calculation, Scale is ignored. Only the base and displacement are used in effective address calculation.
- **m512** — A 64-byte operand in memory.
- **zmm/m512** — A ZMM register or 512-bit memory operand.
- **{k1}{z}** — A mask register used as instruction writemask. The 64-bit k registers are: k1 through k7. Writemask specification is available exclusively via EVEX prefix. The masking can either be done as a merging-masking, where the old values are preserved for masked out elements or as a zeroing masking. The type of masking is determined by using the EVEX.z bit.
- **{k1}** — Without {z}: a mask register used as instruction writemask for instructions that do not allow zeroing-masking but support merging-masking. This corresponds to instructions that require the value of the aaa field to be different than 0 (e.g., gather) and store-type instructions which allow only merging-masking.
- **k1** — A mask register used as a regular operand (either destination or source). The 64-bit k registers are: k0 through k7.
- **mV** — A vector memory operand; the operand size is dependent on the instruction.
- **vm32{x,y, z}** — A vector array of memory operands specified using VSIB memory addressing. The array of memory addresses are specified using a common base register, a constant scale factor, and a vector index register with individual elements of 32-bit index value in an XMM register (vm32x), a YMM register (vm32y) or a ZMM register (vm32z).
- **vm64{x,y, z}** — A vector array of memory operands specified using VSIB memory addressing. The array of memory addresses are specified using a common base register, a constant scale factor, and a vector index register with individual elements of 64-bit index value in an XMM register (vm64x), a YMM register (vm64y) or a ZMM register (vm64z).

- **zmm/m512/m32bcst** — An operand that can be a ZMM register, a 512-bit memory location or a 512-bit vector loaded from a 32-bit memory location.
- **zmm/m512/m64bcst** — An operand that can be a ZMM register, a 512-bit memory location or a 512-bit vector loaded from a 64-bit memory location.
- **<ZMM0>** — Indicates use of the ZMM0 register as an implicit argument.
- **{er}** — Indicates support for embedded rounding control, which is only applicable to the register-register form of the instruction. This also implies support for SAE (Suppress All Exceptions).
- **{sae}** — Indicates support for SAE (Suppress All Exceptions). This is used for instructions that support SAE, but do not support embedded rounding control.
- **SRC1** — Denotes the first source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having two or more source operands.
- **SRC2** — Denotes the second source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having two or more source operands.
- **SRC3** — Denotes the third source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having three source operands.
- **SRC** — The source in a single-source instruction.
- **DST** — The destination in an instruction. This field is encoded by reg_field.

In the instruction encoding, the MODRM byte is represented several ways depending on the role it plays. The MODRM byte has 3 fields: 2-bit MODRM.MOD field, a 3-bit MODRM.REG field and a 3-bit MODRM.RM field. When all bits of the MODRM byte have fixed values for an instruction, the 2-hex nibble value of that byte is presented after the opcode in the encoding boxes on the instruction description pages. When only some fields of the MODRM byte must contain fixed values, those values are specified as follows:

- If only the MODRM.MOD must be 0b11, and MODRM.REG and MODRM.RM fields are unrestricted, this is denoted as **11:rrr:bbb**. The **rrr** correspond to the 3-bits of the MODRM.REG field and the **bbb** correspond to the 3-bits of the MODMR.RM field.
- If the MODRM.MOD field is constrained to be a value other than 0b11, i.e., it must be one of 0b00, 0b01, or 0b10, then we use the notation !(11).
- If the MODRM.REG field had a specific required value, e.g., 0b101, that would be denoted as mm:101:bbb.

3.1.1.4 Operand Encoding Column in the Instruction Summary Table

The "operand encoding" column is abbreviated as Op/En in the Instruction Summary table heading. Instruction operand encoding information is provided for each assembly instruction syntax using a letter to cross reference to a row entry in the operand encoding definition table that follows the instruction summary table. The operand encoding table in each instruction reference page lists each instruction operand (according to each instruction syntax and operand ordering shown in the instruction column) relative to the ModRM byte, VEX.vvvv field or additional operand encoding placement.

EVEX encoded instructions employ compressed disp8*N encoding of the displacement bytes, where N is defined in Table 2-34 and Table 2-35, according to tupletypes. The tupletype for an instruction is listed in the operand encoding definition table where applicable.

NOTES

- The letters in the Op/En column of an instruction apply ONLY to the encoding definition table immediately following the instruction summary table.
- In the encoding definition table, the letter 'r' within a pair of parenthesis denotes the content of the operand will be read by the processor. The letter 'w' within a pair of parenthesis denotes the content of the operand will be updated by the processor.

3.1.1.5 64/32-bit Mode Column in the Instruction Summary Table

The "64/32-bit Mode" column indicates whether the opcode sequence is supported in (a) 64-bit mode or (b) the Compatibility mode and other IA-32 modes that apply in conjunction with the CPUID feature flag associated specific instruction extensions.

The 64-bit mode support is to the left of the 'slash' and has the following notation:

- **V** — Supported.
- **I** — Not supported.
- **N.E.** — Indicates an instruction syntax is not encodable in 64-bit mode (it may represent part of a sequence of valid instructions in other modes).
- **N.P.** — Indicates the REX prefix does not affect the legacy instruction in 64-bit mode.
- **N.I.** — Indicates the opcode is treated as a new instruction in 64-bit mode.
- **N.S.** — Indicates an instruction syntax that requires an address override prefix in 64-bit mode and is not supported. Using an address override prefix in 64-bit mode may result in model-specific execution behavior.

The Compatibility/Legacy Mode support is to the right of the 'slash' and has the following notation:

- **V —** Supported.
- **I —** Not supported.
- **N.E. —** Indicates an Intel 64 instruction mnemonics/syntax that is not encodable; the opcode sequence is not applicable as an individual instruction in compatibility mode or IA-32 mode. The opcode may represent a valid sequence of legacy IA-32 instructions.

3.1.1.6 CPUID Support Column in the Instruction Summary Table

The fourth column holds abbreviated CPUID feature flags (e.g., appropriate bit in CPUID.01H.ECX, CPUID.01H.EDX for SSE/SSE2/SSE3/SSSE3/SSE4.1/SSE4.2/AESNI/PCLMULQDQ/AVX/RDRAND support) that indicate processor support for the instruction. If the corresponding flag is '0', the instruction will #UD.

3.1.1.7 Description Column in the Instruction Summary Table

The "Description" column briefly explains forms of the instruction.

3.1.1.8 Description Section

Each instruction is then described by number of information sections. The "Description" section describes the purpose of the instructions and required operands in more detail.

Summary of terms that may be used in the description section:

- **Legacy SSE** — Refers to SSE, SSE2, SSE3, SSSE3, SSE4, AESNI, PCLMULQDQ, and any future instruction sets referencing XMM registers and encoded without a VEX prefix.
- **VEX.vvvv** — The VEX bit field specifying a source or destination register (in 1's complement form).
- **rm_field** — shorthand for the ModR/M *r/m* field and any REX.B.
- **reg_field** — shorthand for the ModR/M *reg* field and any REX.R.

3.1.1.9 Operation Section

The "Operation" section contains an algorithm description (frequently written in pseudo-code) for the instruction. Algorithms are composed of the following elements:

- Comments are enclosed within the symbol pairs "(*" and "*)".
- Compound statements are enclosed in keywords, such as: IF, THEN, ELSE, and FI for an if statement; DO and OD for a do statement; or CASE... OF for a case statement.
- A register name implies the contents of the register. A register name enclosed in brackets implies the contents of the location whose address is contained in that register. For example, ES:[DI] indicates the contents of the location whose ES segment relative address is in register DI. [SI] indicates the contents of the address contained in register SI relative to the SI register's default segment (DS) or the overridden segment.
- Parentheses around the "E" in a general-purpose register name, such as (E)SI, indicates that the offset is read from the SI register if the address-size attribute is 16, from the ESI register if the address-size attribute is 32.

Parentheses around the "R" in a general-purpose register name, (R)SI, in the presence of a 64-bit register definition such as (R)SI, indicates that the offset is read from the 64-bit RSI register if the address-size attribute is 64.

- Brackets are used for memory operands where they mean that the contents of the memory location is a segment-relative offset. For example, [SRC] indicates that the content of the source operand is a segment-relative offset.
- A := B indicates that the value of B is assigned to A.
- The symbols =, ≠, >, <, ≥, and ≤ are relational operators used to compare two values: meaning equal, not equal, greater or equal, less or equal, respectively. A relational expression such as A = B is TRUE if the value of A is equal to B; otherwise it is FALSE.
- The expression "« COUNT" and "» COUNT" indicates that the destination operand should be shifted left or right by the number of bits indicated by the count operand.

The following identifiers are used in the algorithmic descriptions:

- **OperandSize and AddressSize** — The OperandSize identifier represents the operand-size attribute of the instruction, which is 16, 32 or 64-bits. The AddressSize identifier represents the address-size attribute, which is 16, 32 or 64-bits. For example, the following pseudo-code indicates that the operand-size attribute depends on the form of the MOV instruction used.

```
IF Instruction = MOVW
    THEN OperandSize := 16;
ELSE
    IF Instruction = MOVD
        THEN OperandSize := 32;
    ELSE
        IF Instruction = MOVQ
            THEN OperandSize := 64;
        FI;
    FI;
FI;
```

  See "Operand-Size and Address-Size Attributes" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for guidelines on how these attributes are determined.
- **StackAddrSize** — Represents the stack address-size attribute associated with the instruction, which has a value of 16, 32 or 64-bits. See "Address-Size Attribute for Stack" in Chapter 6, "Procedure Calls, Interrupts, and Exceptions," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.
- **SRC** — Represents the source operand.
- **DEST** — Represents the destination operand.
- **MAXVL** — The maximum vector register width pertaining to the instruction. This is not the vector-length encoding in the instruction's encoding but is instead determined by the current value of XCR0. For details, refer to the table below. Note that the value of MAXVL is the largest of the features enabled. Future processors may define new bits in XCR0 whose setting may imply other values for MAXVL.

MAXVL Definition

XCR0 Component	MAXVL
XCR0.SSE	128
XCR0.AVX	256
XCR0.{ZMM_Hi256, Hi16_ZMM, OPMASK}	512

The following functions are used in the algorithmic descriptions:

- **ZeroExtend(value)** — Returns a value zero-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, zero extending a byte value of –10 converts the byte from F6H to

a doubleword value of 000000F6H. If the value passed to the ZeroExtend function and the operand-size attribute are the same size, ZeroExtend returns the value unaltered.

- **SignExtend(value)** — Returns a value sign-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, sign extending a byte containing the value –10 converts the byte from F6H to a doubleword value of FFFFFFF6H. If the value passed to the SignExtend function and the operand-size attribute are the same size, SignExtend returns the value unaltered.
- **SaturateSignedWordToSignedByte** — Converts a signed 16-bit value to a signed 8-bit value. If the signed 16-bit value is less than –128, it is represented by the saturated value -128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateSignedDwordToSignedWord** — Converts a signed 32-bit value to a signed 16-bit value. If the signed 32-bit value is less than –32768, it is represented by the saturated value –32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateSignedWordToUnsignedByte** — Converts a signed 16-bit value to an unsigned 8-bit value. If the signed 16-bit value is less than zero, it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToSignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than –128, it is represented by the saturated value –128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateToSignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than –32768, it is represented by the saturated value –32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateToUnsignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToUnsignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 65535, it is represented by the saturated value 65535 (FFFFH).
- **LowOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the least significant word of the doubleword result in the destination operand.
- **HighOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the most significant word of the doubleword result in the destination operand.
- **Push(value)** — Pushes a value onto the stack. The number of bytes pushed is determined by the operand-size attribute of the instruction. See the "Operation" subsection of the "PUSH—Push Word, Doubleword, or Quadword Onto the Stack" section in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B.
- **Pop()** — removes the value from the top of the stack and returns it. The statement EAX := Pop(); assigns to EAX the 32-bit value from the top of the stack. Pop will return either a word, a doubleword or a quadword depending on the operand-size attribute. See the "Operation" subsection in the "POP—Pop a Value From the Stack" section of Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B.
- **PopRegisterStack** — Marks the FPU ST(0) register as empty and increments the FPU register stack pointer (TOP) by 1.
- **Switch-Tasks** — Performs a task switch.
- **Bit(BitBase, BitOffset)** — Returns the value of a bit within a bit string. The bit string is a sequence of bits in memory or a register. Bits are numbered from low-order to high-order within registers and within memory bytes. If the BitBase is a register, the BitOffset can be in the range 0 to [15, 31, 63] depending on the mode and register size. See Figure 3-1: the function Bit[RAX, 21] is illustrated.

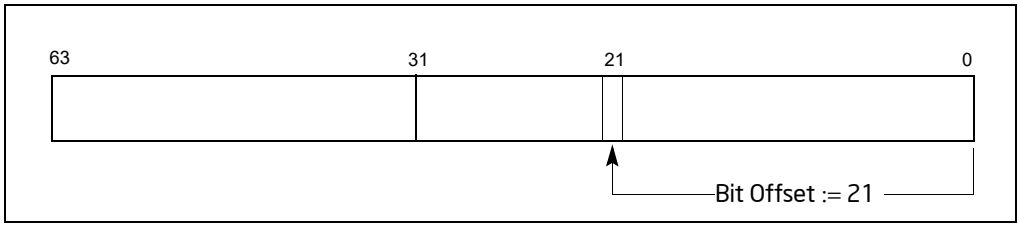


Figure 3-1. Bit Offset for BIT[RAX, 21]

If BitBase is a memory address, the BitOffset has different ranges depending on the operand size (see Table 3-2).

Table 3-2. Range of Bit Positions Specified by Bit Offset Operands

Operand Size	Immediate BitOffset	Register BitOffset
16	0 to 15	-2^{15} to $2^{15}-1$
32	0 to 31	-2^{31} to $2^{31}-1$
64	0 to 63	-2^{63} to $2^{63}-1$

The addressed bit is numbered (Offset MOD 8) within the byte at address (BitBase + (BitOffset DIV 8)) where DIV is signed division with rounding towards negative infinity and MOD returns a positive number (see Figure 3-2).

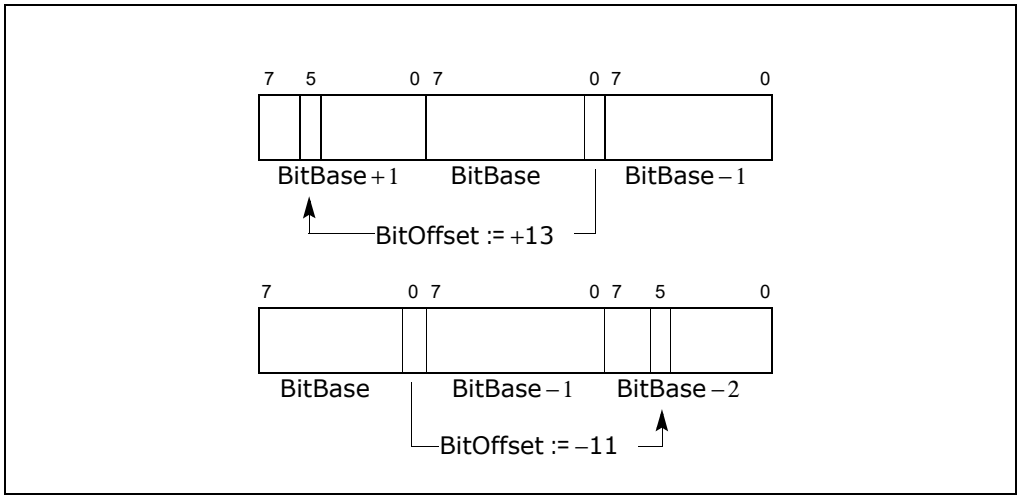


Figure 3-2. Memory Bit Indexing

3.1.1.10 Intel® C/C++ Compiler Intrinsics Equivalents Section

The Intel C/C++ compiler intrinsic functions give access to the full power of the Intel Architecture Instruction Set, while allowing the compiler to optimize register allocation and instruction scheduling for faster execution. Most of these functions are associated with a single IA instruction, although some may generate multiple instructions or different instructions depending upon how they are used. In particular, these functions are used to invoke instructions that perform operations on vector registers that can hold multiple data elements. These SIMD instructions use the following data types.

- __m128, __m256, and __m512 can represent 4, 8, or 16 packed single precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The __m128 data type is also used with various single precision floating-point scalar instructions that perform calculations using only the lowest 32 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.
- __m128d, __m256d, and __m512d can represent 2, 4, or 8 packed double precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The __m128d data type is also used with various double precision floating-point scalar instructions that perform calculations using only the lowest 64 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.
- __m128i, __m256i, and __m512i can represent integer data in bytes, words, doublewords, quadwords, and occasionally larger data types.

Each of these data types incorporates in its name the number of bits it can hold. For example, the __m128 type holds 128 bits, and because each single precision floating-point value is 32 bits long the __m128 type holds (128/32) or four values. Normally the compiler will allocate memory for these data types on an even multiple of the size of the type. Such aligned memory locations may be faster to read and write than locations at other addresses.

These SIMD data types are not basic Standard C data types or C++ objects, so they may be used only with the assignment operator, passed as function arguments, and returned from a function call. If you access the internal members of these types directly, or indirectly by using them in a union, there may be side effects affecting optimization, so it is recommended to use them only with the SIMD instruction intrinsic functions described in this manual or the Intel C/C++ compiler documentation.

Many intrinsic functions names are prefixed with an indicator of the vector length and suffixed by an indicator of the vector element data type, although some functions do not follow the rules below. The prefixes are:

- _mm_ indicates that the function operates on 128-bit (or sometimes 64-bit) vectors.
- _mm256_ indicates the function operates on 256-bit vectors.
- _mm512_ indicates that the function operates on 512-bit vectors.

The suffixes include:

- _ps, which indicates a function that operates on packed single precision floating-point data. Packed single precision floating-point data corresponds to arrays of the C/C++ type *float* with either 4, 8 or 16 elements. Values of this type can be loaded from an array using the *_mm_loadu_ps, _mm256_loadu_ps,* or *_mm512_loadu_ps* functions, or created from individual values using *_mm_set_ps, _mm256_set_ps,* or *_mm512_set_ps* functions, and they can be stored in an array using *_mm_storeu_ps, _mm256_storeu_ps,* or *_mm512_storeu_ps*.
- _ss, which indicates a function that operates on scalar single precision floating-point data. Single precision floating-point data corresponds to the C/C++ type *float*, and values of type float can be converted to type __m128 for use with these functions using the *_mm_set_ss* function, and converted back using the *_mm_cvtss_f32* function. When used with functions that operate on packed single precision floating-point data the scalar element corresponds with the first packed value.
- _pd, which indicates a function that operates on packed double precision floating-point data. Packed double precision floating-point data corresponds to arrays of the C/C++ type *double* with either 2, 4, or 8 elements. Values of this type can be loaded from an array using the *_mm_loadu_pd, _mm256_loadu_pd,* or *_mm512_loadu_pd* functions, or created from individual values using *_mm_set_pd, _mm2566_set_pd,* or *_mm512_set_pd* functions, and they can be stored in an array using *_mm_storeu_pd, _mm256_storeu_pd,* or *_mm512_storeu_pd*.
- _sd, which indicates a function that operates on scalar double precision floating-point data. Double-precision floating-point data corresponds to the C/C++ type *double*, and values of type double can be converted to type __m128d for use with these functions using the *_mm_set_sd* function, and converted back using the *_mm_cvtsd_f64* function. When used with functions that operate on packed double precision floating-point data the scalar element corresponds with the first packed value.
- _epi8, which indicates a function that operates on packed 8-bit signed integer values. Packed 8-bit signed integers correspond to an array of *signed char* with 16, 32 or 64 elements. Values of this type can be created from individual elements using *_mm_set_epi8, _mm256_set_epi8,* or *_mm512_set_epi8* functions.
- _epi16, which indicates a function that operates on packed 16-bit signed integer values. Packed 16-bit signed integers correspond to an array of *short* with 8, 16 or 32 elements. Values of this type can be created from individual elements using *_mm_set_epi16, _mm256_set_epi16,* or *_mm512_set_epi16* functions.
- _epi32, which indicates a function that operates on packed 32-bit signed integer values. Packed 32-bit signed integers correspond to an array of *int* with 4, 8 or 16 elements. Values of this type can be created from individual elements using *_mm_set_epi32, _mm256_set_epi32,* or *_mm512_set_epi32* functions.
- _epi64, which indicates a function that operates on packed 64-bit signed integer values. Packed 64-bit signed integers correspond to an array of *long long* (or *long* if it is a 64-bit data type) with 2, 4 or 8 elements. Values of this type can be created from individual elements using *_mm_set_epi32, _mm256_set_epi32,* or *_mm512_set_epi32* functions.
- _epu8, which indicates a function that operates on packed 8-bit unsigned integer values. Packed 8-bit unsigned integers correspond to an array of *unsigned char* with 16, 32 or 64 elements.

- _epu16, which indicates a function that operates on packed 16-bit unsigned integer values. Packed 16-bit unsigned integers correspond to an array of *unsigned short* with 8, 16 or 32 elements.
- _epu32, which indicates a function that operates on packed 32-bit unsigned integer values. Packed 32-bit unsigned integers correspond to an array of *unsigned* with 4, 8 or 16 elements.
- _epu64, which indicates a function that operates on packed 64-bit unsigned integer values. Packed 64-bit unsigned integers correspond to an array of *unsigned long long* (or *unsigned long* if it is a 64-bit data type) with 2, 4 or 8 elements.
- _si128, which indicates a function that operates on a single 128-bit value of type __m128i.
- _si256, which indicates a function that operates on a single a 256-bit value of type __m256i.
- _si512, which indicates a function that operates on a single a 512-bit value of type __m512i.

Values of any packed integer type can be loaded from an array using the *_mm_loadu_si128, _mm256_loadu_-si256,* or *_mm512_loadu_si512* functions, and they can be stored in an array using *_mm_storeu_si128, _mm256_storeu_si256,* or *_mm512_storeu_si512*.

These functions and data types are used with the SSE, AVX, and AVX-512 instruction set extension families. In addition there are similar functions that correspond to MMX instructions. These are less frequently used because they require additional state management, and only operate on 64-bit packed integer values.

The declarations of Intel C/C++ compiler intrinsic functions may reference some non-standard data types, such as __int64. The C Standard header *stdint.h* defines similar platform-independent types, and the documentation for that header gives characteristics that apply to corresponding non-standard types according to the following table.

Table 3-3. Standard and Non-Standard Data Types

Non-standard Type	Standard Type (from stdint.h)
__int64	int64_t
unsigned __int64	uint64_t
__int32	int32_t
unsigned __int32	uint32_t
__int16	int16_t
unsigned __int16	uint16_t

For a more detailed description of each intrinsic function and additional information related to its usage, refer to the online Intel Intrinsics Guide, https://software.intel.com/sites/landingpage/IntrinsicsGuide.

3.1.1.11 Flags Affected Section

The "Flags Affected" section lists the flags in the EFLAGS register that are affected by the instruction. When a flag is cleared, it is equal to 0; when it is set, it is equal to 1. The arithmetic and logical instructions usually assign values to the status flags in a uniform manner (see Appendix A, "EFLAGS Cross-Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). Non-conventional assignments are described in the "Operation" section. The values of flags listed as **undefined** may be changed by the instruction in an indeterminate manner. Flags that are not listed are unchanged by the instruction.

3.1.1.12 FPU Flags Affected Section

The floating-point instructions have an "FPU Flags Affected" section that describes how each instruction can affect the four condition code flags of the FPU status word.

3.1.1.13 Protected Mode Exceptions Section

The "Protected Mode Exceptions" section lists the exceptions that can occur when the instruction is executed in protected mode and the reasons for the exceptions. Each exception is given a mnemonic that consists of a pound

sign (#) followed by two letters and an optional error code in parentheses. For example, #GP(0) denotes a general protection exception with an error code of 0. Table 3-4 associates each two-letter mnemonic with the corresponding exception vector and name. See Chapter 6, "Procedure Calls, Interrupts, and Exceptions," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for a detailed description of the exceptions.

Application programmers should consult the documentation provided with their operating systems to determine the actions taken when exceptions occur.

Table 3-4. Intel 64 and IA-32 General Exceptions

Vector	Name	Source	Protected Mode[1]	Real Address Mode	Virtual 8086 Mode
0	#DE—Divide Error	DIV and IDIV instructions.	Yes	Yes	Yes
1	#DB—Debug	Any code or data reference.	Yes	Yes	Yes
3	#BP—Breakpoint	INT3 instruction.	Yes	Yes	Yes
4	#OF—Overflow	INTO instruction.	Yes	Yes	Yes
5	#BR—BOUND Range Exceeded	BOUND instruction.	Yes	Yes	Yes
6	#UD—Invalid Opcode (Undefined Opcode)	UD instruction or reserved opcode.	Yes	Yes	Yes
7	#NM—Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes
8	#DF—Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.	Yes	Yes	Yes
10	#TS—Invalid TSS	Task switch or TSS access.	Yes	Reserved	Yes
11	#NP—Segment Not Present	Loading segment registers or accessing system segments.	Yes	Reserved	Yes
12	#SS—Stack Segment Fault	Stack operations and SS register loads.	Yes	Yes	Yes
13	#GP—General Protection[2]	Any memory reference and other protection checks.	Yes	Yes	Yes
14	#PF—Page Fault	Any memory reference.	Yes	Reserved	Yes
16	#MF—Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes
17	#AC—Alignment Check	Any data reference in memory.	Yes	Reserved	Yes
18	#MC—Machine Check	Model dependent machine check errors.	Yes	Yes	Yes
19	#XM—SIMD Floating-Point Numeric Error	SSE/SSE2/SSE3 floating-point instructions.	Yes	Yes	Yes

NOTES:

1. Apply to protected mode, compatibility mode, and 64-bit mode.
2. In the real-address mode, vector 13 is the segment overrun exception.

3.1.1.14 Real-Address Mode Exceptions Section

The "Real-Address Mode Exceptions" section lists the exceptions that can occur when the instruction is executed in real-address mode (see Table 3-4).

3.1.1.15 Virtual-8086 Mode Exceptions Section

The "Virtual-8086 Mode Exceptions" section lists the exceptions that can occur when the instruction is executed in virtual-8086 mode (see Table 3-4).

3.1.1.16 Floating-Point Exceptions Section

The "Floating-Point Exceptions" section lists exceptions that can occur when an x87 FPU floating-point instruction is executed. All of these exception conditions result in a floating-point error exception (#MF, exception 16) being generated. Table 3-5 associates a one- or two-letter mnemonic with the corresponding exception name. See "Floating-Point Exception Conditions" in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for a detailed description of these exceptions.

Table 3-5. x87 FPU Floating-Point Exceptions

Mnemonic	Name	Source
#IS #IA	Floating-point invalid operation: - Stack overflow or underflow - Invalid arithmetic operation	 - x87 FPU stack overflow or underflow - Invalid FPU arithmetic operation
#Z	Floating-point divide-by-zero	Divide-by-zero
#D	Floating-point denormal operand	Source operand that is a denormal number
#O	Floating-point numeric overflow	Overflow in result
#U	Floating-point numeric underflow	Underflow in result
#P	Floating-point inexact result (precision)	Inexact result (precision)

3.1.1.17 SIMD Floating-Point Exceptions Section

The "SIMD Floating-Point Exceptions" section lists exceptions that can occur when an SSE/SSE2/SSE3 floating-point instruction is executed. All of these exception conditions result in a SIMD floating-point error exception (#XM, exception 19) being generated. Table 3-6 associates a one-letter mnemonic with the corresponding exception name. For a detailed description of these exceptions, refer to "SSE and SSE2 Exceptions", in Chapter 11 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

Table 3-6. SIMD Floating-Point Exceptions

Mnemonic	Name	Source
#I	Floating-point invalid operation	Invalid arithmetic operation or source operand
#Z	Floating-point divide-by-zero	Divide-by-zero
#D	Floating-point denormal operand	Source operand that is a denormal number
#O	Floating-point numeric overflow	Overflow in result
#U	Floating-point numeric underflow	Underflow in result
#P	Floating-point inexact result	Inexact result (precision)

3.1.1.18 Compatibility Mode Exceptions Section

This section lists exceptions that occur within compatibility mode.

3.1.1.19 64-Bit Mode Exceptions Section

This section lists exceptions that occur within 64-bit mode.

3.2 INTEL® AMX CONSIDERATIONS

The following implementation parameters and helper functions are applicable to the Intel® AMX instructions.

3.2.1 Implementation Parameters

The parameters are reported via CPUID leaf 1DH. Index 0 reports all zeros for all fields.

```
define palette_table[id]:
    uint16_t total_tile_bytes
    uint16_t bytes_per_tile
    uint16_t bytes_per_row
    uint16_t max_names
    uint16_t max_rows
```

The tile parameters are set by LDTILECFG or XRSTOR* of TILECFG:

```
define tile[tid]:
    byte rows
    word colsb // bytes_per_row
    bool valid
```

3.2.2 Helper Functions

The helper functions used in Intel AMX instructions are defined below.

```
define write_row_and_zero(treg, r, data, nbytes):
   for j in 0 ...nbytes-1:
       treg.row[r].byte[j] := data.byte[j]

   // zero the rest of the row
   for j in nbytes ... palette_table[tilecfg.palette_id].bytes_per_row-1:
      treg.row[r].byte[j] := 0


define zero_upper_rows(treg, r):
   for i in r ... palette_table[tilecfg.palette_id].max_rows-1:
      for j in 0 ... palette_table[tilecfg.palette_id].bytes_per_row-1:
                   treg.row[i].byte[j] := 0



define zero_tilecfg_start():
   tilecfg.start_row :=0


define zero_all_tile_data():
  if XCR0[TILEDATA]:
      b := CPUID(0xD,TILEDATA).EAX // size of feature
      for j in 0 ...b:
             TILEDATA.byte[j] := 0
```

```
define xcr0_supports_palette(palette_id):
  if palette_id == 0:
     return 1
  elif palette_id == 1:
     if XCR0[TILECFG] and XCR0[TILEDATA]:
        return 1
  return 0
```

3.3 INSTRUCTIONS (A-L)

The remainder of this chapter provides descriptions of Intel 64 and IA-32 instructions (A-L). See also: Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B; Chapter 5, "Instruction Set Reference, V," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C; and Chapter 6, "Instruction Set Reference, W-Z," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

BSWAP—Byte Swap

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
0F C8+*rd*	BSWAP *r32*	O	Valid*	Valid	Reverses the byte order of a 32-bit register.
REX.W + 0F C8+*rd*	BSWAP *r64*	O	Valid	N.E.	Reverses the byte order of a 64-bit register.

NOTES:

* See IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
O	opcode + rd (r, w)	N/A	N/A	N/A

Description

Reverses the byte order of a 32-bit or 64-bit (destination) register. This instruction is provided for converting little-endian values to big-endian format and vice versa. To swap bytes in a word value (16-bit register), use the XCHG instruction. When the BSWAP instruction references a 16-bit register, the result is undefined.

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.B permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

IA-32 Architecture Legacy Compatibility

The BSWAP instruction is not supported on IA-32 processors earlier than the Intel486™ processor family. For compatibility with this instruction, software should include functionally equivalent code for execution on Intel processors earlier than the Intel486 processor family.

Operation

```
TEMP := DEST
IF 64-bit mode AND OperandSize = 64
    THEN
        DEST[7:0] := TEMP[63:56];
        DEST[15:8] := TEMP[55:48];
        DEST[23:16] := TEMP[47:40];
        DEST[31:24] := TEMP[39:32];
        DEST[39:32] := TEMP[31:24];
        DEST[47:40] := TEMP[23:16];
        DEST[55:48] := TEMP[15:8];
        DEST[63:56] := TEMP[7:0];
    ELSE
        DEST[7:0] := TEMP[31:24];
        DEST[15:8] := TEMP[23:16];
        DEST[23:16] := TEMP[15:8];
        DEST[31:24] := TEMP[7:0];
FI;
```

Flags Affected

None.

Exceptions (All Operating Modes)

#UD	If the LOCK prefix is used.

BTC—Bit Test and Complement

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
0F BB /r	BTC r/m16, r16	MR	Valid	Valid	Store selected bit in CF flag and complement.
0F BB /r	BTC r/m32, r32	MR	Valid	Valid	Store selected bit in CF flag and complement.
REX.W + 0F BB /r	BTC r/m64, r64	MR	Valid	N.E.	Store selected bit in CF flag and complement.
0F BA /7 ib	BTC r/m16, imm8	MI	Valid	Valid	Store selected bit in CF flag and complement.
0F BA /7 ib	BTC r/m32, imm8	MI	Valid	Valid	Store selected bit in CF flag and complement.
REX.W + 0F BA /7 ib	BTC r/m64, imm8	MI	Valid	N.E.	Store selected bit in CF flag and complement.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (r, w)	ModRM:reg (r)	N/A	N/A
MI	ModRM:r/m (r, w)	imm8	N/A	N/A

Description

Selects the bit in a bit string (specified with the first operand, called the bit base) at the bit-position designated by the bit offset operand (second operand), stores the value of the bit in the CF flag, and complements the selected bit in the bit string. The bit base operand can be a register or a memory location; the bit offset operand can be a register or an immediate value:

- If the bit base operand specifies a register, the instruction takes the modulo 16, 32, or 64 of the bit offset operand (modulo size depends on the mode and register size; 64-bit operands are available only in 64-bit mode). This allows any bit position to be selected.
- If the bit base operand specifies a memory location, the operand represents the address of the byte in memory that contains the bit base (bit 0 of the specified byte) of the bit string. The range of the bit position that can be referenced by the offset operand depends on the operand size.

See also: **Bit(BitBase, BitOffset)** on page 3-81.

Some assemblers support immediate bit offsets larger than 31 by using the immediate bit offset field in combination with the displacement field of the memory operand. See "BT—Bit Test" in this chapter for more information on this addressing mechanism.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.B permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
CF := Bit(BitBase, BitOffset);
Bit(BitBase, BitOffset) := NOT Bit(BitBase, BitOffset);
```

Flags Affected

The CF flag contains the value of the selected bit before it is complemented. The ZF flag is unaffected. The OF, SF, AF, and PF flags are undefined.

Protected Mode Exceptions

#GP(0)	If the destination operand points to a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

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 a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

CPUID—CPU Identification

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F A2	CPUID	ZO	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
ZO	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	
	Basic CPUID Information	
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.		
	Deterministic Cache Parameters Leaf (Initial EAX Value = 04H)	
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 321.
	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	
		Bits 07-05: Cache Level (starts at 1). Bit 08: Self Initializing cache level (does not need SW initialization). Bit 09: Fully Associative cache. 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**, ****, *****.
	EBX	Bits 11-00: L = System Coherency Line Size**. Bits 21-12: P = Physical Line partitions**. Bits 31-22: W = Ways of associativity**.
	ECX	Bits 31-00: S = Number of Sets**.
	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. **NOTES:** * 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. ** Add one to the return value to get the result. ***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. **** 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. ***** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.
	MONITOR/MWAIT Leaf (Initial EAX Value = 05H)	
05H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity). Bits 31-16: Reserved = 0.
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity). Bits 31-16: Reserved = 0.
	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.
	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.

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

Initial EAX Value	Information Provided about the Processor	
	NOTE: * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.	
Thermal and Power Management Leaf (Initial EAX Value = 06H)		
06H	EAX	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 PECI 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.
	EBX	Bits 03-00: Number of Interrupt Thresholds in Digital Thermal Sensor. Bits 31-04: Reserved.
	ECX	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 (1B0H). 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.

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

Initial EAX Value	Information Provided about the Processor	
	EDX	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 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. 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. **NOTE:** Bits 0 and 1 will always be set together.
Structured Extended Feature Flags Enumeration Leaf (Initial EAX Value = 07H, ECX = 0)		
07H	EAX	Bits 31-00: Reports the maximum input value for supported leaf 7 sub-leaves.
	EBX	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.

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

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: PREFETCHWT1. (Intel® Xeon Phi™ only.) Bit 01: AVX512_VBMI. Bit 02: UMIP. Supports user-mode instruction prevention if 1. Bit 03: PKU. Supports protection keys for user-mode pages if 1. Bit 04: OSPKE. If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions). Bit 05: WAITPKG. Bit 06: AVX512_VBMI2. 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. Bit 08: GFNI. Bit 09: VAES. Bit 10: VPCLMULQDQ. Bit 11: AVX512_VNNI. Bit 12: AVX512_BITALG. 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. Bit 14: AVX512_VPOPCNTDQ. Bit 15: Reserved. Bit 16: LA57. Supports 57-bit linear addresses and five-level paging if 1. Bits 21-17: The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode. Bit 22: RDPID and IA32_TSC_AUX are available if 1. Bit 23: KL. Supports Key Locker if 1. Bit 24: BUS_LOCK_DETECT. If 1, indicates support for OS bus-lock detection. Bit 25: CLDEMOTE. Supports cache line demote if 1. Bit 26: Reserved. Bit 27: MOVDIRI. Supports MOVDIRI if 1. Bit 28: MOVDIR64B. Supports MOVDIR64B if 1. Bit 29: ENQCMD. Supports Enqueue Stores if 1. Bit 30: SGX_LC. Supports SGX Launch Configuration if 1. Bit 31: PKS. Supports protection keys for supervisor-mode pages if 1.
	EDX	Bit 00: Reserved. Bit 01: SGX-KEYS. If 1, Attestation Services for Intel® SGX is supported. Bit 02: AVX512_4VNNIW. (Intel® Xeon Phi™ only.) Bit 03: AVX512_4FMAPS. (Intel® Xeon Phi™ only.) Bit 04: Fast Short REP MOV. Bit 05: UINTR. If 1, the processor supports user interrupts. Bits 07-06: Reserved. Bit 08: AVX512_VP2INTERSECT. 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. Bit 10: MD_CLEAR supported. Bit 11: RTM_ALWAYS_ABORT. If set, any execution of XBEGIN immediately aborts and transitions to the specified fallback address. Bit 12: Reserved. 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). Bit 14: SERIALIZE. 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. Bit 16: TSXLDTRK. If 1, the processor supports Intel TSX suspend/resume of load address tracking.

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

Initial EAX Value	Information Provided about the Processor	
		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. 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. 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. 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). **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.
Structured Extended Feature Enumeration Sub-leaf (Initial EAX Value = 07H, ECX = 1)		
07H		**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.
	EAX	This field reports 0 if the sub-leaf index, *1*, is invalid. 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 29-23: Reserved.

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

Initial EAX Value	Information Provided about the Processor	
		Bit 30: INVD_DISABLE_POST_BIOS_DONE. If 1, supports INVD execution prevention after BIOS Done. Bit 31: Reserved.
	EBX	This field reports 0 if the sub-leaf index, *1*, is invalid. Bit 00: Enumerates the presence of the IA32_PPIN and IA32_PPIN_CTL MSRs. If 1, these MSRs are supported. Bits 31-01: Reserved.
	ECX	This field reports 0 if the sub-leaf index, *1*, is invalid; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, *1*, is invalid. Bits 17-00: Reserved. Bit 18: CET_SSS. If 1, indicates that an operating system can enable supervisor shadow stacks as long as it ensures that a supervisor shadow stack cannot become prematurely busy due to 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 (VMM) should return this bit as 1 only if it ensures that VM exits cannot cause a guest supervisor shadow stack to appear to be prematurely busy. Such a VMM could set the "prematurely busy shadow stack" VM-exit control and use the additional information that it provides. Bits 31-19: Reserved.
Structured Extended Feature Enumeration Sub-leaf (Initial EAX Value = 07H, ECX = 2)		
07H		**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.
	EAX	This field reports 0 if the sub-leaf index, 2, is invalid; otherwise it is reserved.
	EBX	This field reports 0 if the sub-leaf index, 2, is invalid; otherwise it is reserved.
	ECX	This field reports 0 if the sub-leaf index, 2, is invalid; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, 2, is invalid. 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. 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. 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. 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. 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. 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. Bits 31-06: Reserved.
Direct Cache Access Information Leaf (Initial EAX Value = 09H)		
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.

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

Initial EAX Value	Information Provided about the Processor	
	Architectural Performance Monitoring Leaf (Initial EAX Value = 0AH)	
0AH	EAX	Bits 07-00: Version ID of architectural performance monitoring. Bits 15-08: Number of general-purpose performance monitoring counter per logical processor. Bits 23-16: Bit width of general-purpose, performance monitoring counter. 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.
	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.
	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);
	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.
	Extended Topology Enumeration Leaf (Initial EAX Value = 0BH)	
0BH		**NOTES:** *CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH.* The sub-leaves of CPUID leaf 0BH 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). 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. 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. 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.
	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.

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

Initial EAX Value	Information Provided about the Processor	
	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.
	ECX	Bits 07-00: The input ECX sub-leaf index. 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. Hierarchy / Domain / Domain Type Identification Value Lowest / Logical Processor / 1 Highest / Core / 2 (Note that enumeration values of 0 and 3-255 are reserved.) Bits 31-16: Reserved.
	EDX	Bits 31-00: x2APIC ID of the current logical processor.
		NOTES: * 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.
Processor Extended State Enumeration Main Leaf (Initial EAX Value = 0DH, ECX = 0)		
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	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. Bit 00: x87 state. Bit 01: SSE state. Bit 02: AVX state. Bits 04-03: MPX state. Bits 07-05: AVX-512 state. Bit 08: Used for IA32_XSS. Bit 09: PKRU state. Bits 16-10: Used for IA32_XSS. Bit 17: TILECFG state. Bit 18: TILEDATA state. Bits 31-19: Reserved.
	EBX	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.
	ECX	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.
	EDX	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. Bits 31-00: Reserved.

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

Initial EAX Value	Information Provided about the Processor	
	Processor Extended State Enumeration Sub-leaf (Initial EAX Value = 0DH, ECX = 1)	
0DH	EAX	Bit 00: XSAVEOPT is available. Bit 01: Supports XSAVEC and the compacted form of XRSTOR if set. Bit 02: Supports XGETBV with ECX = 1 if set. Bit 03: Supports XSAVES/XRSTORS and IA32_XSS if set. Bit 04: Supports extended feature disable (XFD) if set. Bits 31-05: Reserved.
	EBX	Bits 31-00: The size in bytes of the XSAVE area containing all states enabled by XCR0 | IA32_XSS. **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 XCR0. If EAX[1] and EAX[3] are both enumerated as 0, EBX enumerates zero.
	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 XCR0. Bit 08: PT state. Bit 09: Used for XCR0. 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 XCR0. Bits 31-19: Reserved.
	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.
	Processor Extended State Enumeration Sub-leaves (Initial EAX Value = 0DH, ECX = n, n > 1)	
0DH		**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 XCR0 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].
	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*.
	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 XCR0 register*.
	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 XCR0. 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*.

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

Initial EAX Value	Information Provided about the Processor	
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
Intel® Resource Director Technology (Intel® RDT) Monitoring Enumeration Sub-leaf (Initial EAX Value = 0FH, ECX = 0)		
0FH		**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.
	EAX	Reserved.
	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.
L3 Cache Intel® RDT Monitoring Capability Enumeration Sub-leaf (Initial EAX Value = 0FH, ECX = 1)		
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX.
	EAX	Bits 07-00:The counter width is encoded as an offset from 24b. A value of zero in this field indicates that 24-bit counters are supported. A value of 8 in this field indicates that 32-bit counters are supported. Bit 08: If 1, indicates the presence of an overflow bit in the IA32_QM_CTR MSR (bit 61). Bit 09: If 1, indicates the presence of non-CPU agent Intel RDT CMT support. Bit 10: If 1, indicates the presence of non-CPU agent Intel RDT MBM support. Bits 31-11: 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.
Intel® Resource Director Technology (Intel® RDT) Allocation Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = 0)		
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.
L3 Cache Allocation Technology Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID =1)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.

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

Initial EAX Value	Information Provided about the Processor	
	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	Bit 00: Reserved. Bit 01: If 1, indicates L3 CAT for non-CPU agents is supported. Bit 02: If 1, indicates L3 Code and Data Prioritization Technology is supported. Bit 03: If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous. Bits 31-04: Reserved.
	EDX	Bits 15-00: Highest Class of Service (CLOS) number supported for this ResID. Bits 31-16: Reserved.
L2 Cache Allocation Technology Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID =2)		
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: CDP. If 1, indicates L2 Code and Data Prioritization Technology is supported. Bit 03: If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous. Bits 31-04: Reserved.
	EDX	Bits 15-00: Highest CLOS number supported for this ResID. Bits 31-16: Reserved.
Memory Bandwidth Allocation Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID =3)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	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.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 01-00: Reserved. Bit 02: Reports whether the response of the delay values is linear. Bits 31-03: Reserved.
	EDX	Bits 15-00: Highest CLOS number supported for this ResID. Bits 31-16: Reserved.
Intel® SGX Capability Enumeration Leaf, Sub-leaf 0 (Initial EAX Value = 12H, ECX = 0)		
12H		**NOTES:** Leaf 12H sub-leaf 0 (ECX = 0) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.

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

Initial EAX Value	Information Provided about the Processor	
	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 EINCVIRTCHILD, EDECVIRTCHILD, 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.
	EBX	Bits 31-00: MISCSELECT. Bit vector of supported extended SGX features.
	ECX	Bits 31-00: Reserved.
	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.
Intel SGX Attributes Enumeration Leaf, Sub-leaf 1 (Initial EAX Value = 12H, ECX = 1)		
12H		**NOTES:** Leaf 12H sub-leaf 1 (ECX = 1) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	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.
Intel® SGX EPC Enumeration Leaf, Sub-leaves (Initial EAX Value = 12H, ECX = 2 or higher)		
12H		**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.
	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.
	Type	0000b. This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.

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

Initial EAX Value	Information Provided about the Processor	
	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, integrity, and replay protection. If ECX[3:0] = 0010b, then this section has confidentiality protection only. If EAX[3:0] = 0011b, then this section has confidentiality and integrity protection. 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.
	Intel® Processor Trace Enumeration Main Leaf (Initial EAX Value = 14H, ECX = 0)	
14H		**NOTES:** Leaf 14H main leaf (ECX = 0).
	EAX	Bits 31-00: Reports the maximum sub-leaf supported in leaf 14H.
	EBX	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. Bit 01: If 1, indicates support of Configurable PSB and Cycle-Accurate Mode. Bit 02: If 1, indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset. Bit 03: If 1, indicates support of MTC timing packet and suppression of COFI-based packets. 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. Bit 05: If 1, indicates support of Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation. 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. Bit 07: If 1, writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation. Bit 08: If 1, writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation. Bit 31-09: Reserved.
	ECX	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. Bit 01: If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. Bit 02: If 1, indicates support of Single-Range Output scheme. Bit 03: If 1, indicates support of output to Trace Transport subsystem. Bit 30-04: Reserved. Bit 31: If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.
	EDX	Bits 31-00: Reserved.

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

Initial EAX Value	Information Provided about the Processor	
	Intel® Processor Trace Enumeration Sub-leaf (Initial EAX Value = 14H, ECX = 1)	
14H	EAX	Bits 02-00: Number of configurable Address Ranges for filtering. Bits 15-03: Reserved. Bits 31-16: Bitmap of supported MTC period encodings.
	EBX	Bits 15-00: Bitmap of supported Cycle Threshold value encodings. Bit 31-16: Bitmap of supported Configurable PSB frequency encodings.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
	Time Stamp Counter and Nominal Core Crystal Clock Information Leaf (Initial EAX Value = 15H)	
15H		**NOTES:** If EBX[31:0] is 0, the TSC/"core crystal clock" ratio is not enumerated. EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency. If ECX is 0, the nominal core crystal clock frequency is not enumerated. "TSC frequency" = "core crystal clock frequency" * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	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.
	Processor Frequency Information Leaf (Initial EAX Value = 16H)	
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.
		NOTES: * 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. While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.
	System-On-Chip Vendor Attribute Enumeration Main Leaf (Initial EAX Value = 17H, ECX = 0)	
17H		**NOTES:** 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.

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

Initial EAX Value	Information Provided about the Processor	
	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.
System-On-Chip Vendor Attribute Enumeration Sub-leaf (Initial EAX Value = 17H, ECX = 1..3)		
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.
		NOTES: 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.
System-On-Chip Vendor Attribute Enumeration Sub-leaves (Initial EAX Value = 17H, ECX > MaxSOCID_Index)		
17H		**NOTES:** Leaf 17H output depends on the initial value in ECX.
	EAX	Bits 31-00: Reserved = 0.
	EBX	Bits 31-00: Reserved = 0.
	ECX	Bits 31-00: Reserved = 0.
	EDX	Bits 31-00: Reserved = 0.
Deterministic Address Translation Parameters Main Leaf (Initial EAX Value = 18H, ECX = 0)		
18H		**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). 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.
	EAX	Bits 31-00: Reports the maximum input value of supported sub-leaf in leaf 18H.

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

Initial EAX Value	Information Provided about the Processor	
	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.
Deterministic Address Translation Parameters Sub-leaf (Initial EAX Value = 18H, ECX ≥ 1)		
18H		**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. 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.
	EAX	Bits 31-00: Reserved.
	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.

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

Initial EAX Value	Information Provided about the Processor	
	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.
	Key Locker Leaf (Initial EAX Value = 19H)	
19H	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.
	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.
	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.
	EDX	Reserved.
	Native Model ID Enumeration Leaf (Initial EAX Value = 1AH, ECX = 0)	
1AH		**NOTES:** 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 ≥ 1AH and CPUID.1A.EAX ≠ 0, then the leaf exists.
	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.
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.

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

Initial EAX Value	Information Provided about the Processor	
	PCONFIG Information Sub-leaf (Initial EAX Value = 1BH, ECX ≥ 0)	
1BH		For details on this sub-leaf, see "INPUT EAX = 1BH: Returns PCONFIG Information" on page 3-323. **NOTE:** Leaf 1BH is supported if CPUID.(EAX=07H, ECX=0H):EDX[18] = 1.
	Last Branch Records Information Leaf (Initial EAX Value = 1CH)	
1CH		**NOTE:** This leaf pertains to the architectural feature.
	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.
	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.
	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.
	EDX	Bits 31-00: Reserved.
	Tile Information Main Leaf (Initial EAX Value = 1DH, ECX = 0)	
1DH		**NOTES:** For sub-leaves of 1DH, they are indexed by the palette id. Leaf 1DH sub-leaves 2 and above are reserved.
	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.
	Tile Palette 1 Sub-leaf (Initial EAX Value = 1DH, ECX = 1)	
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.

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

Initial EAX Value	Information Provided about the Processor
TMUL Information Main Leaf (Initial EAX Value = 1EH, ECX = 0)	
1EH	**NOTE:** Leaf 1EH sub-leaves 1 and above are reserved. 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.
V2 Extended Topology Enumeration Leaf (Initial EAX Value = 1FH)	
1FH	**NOTES:** *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.* 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). 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." 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. 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. 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. EBX: Bits 15-00: The number of logical processors across all instances of this domain within the next higher-scoped domain relative to this current logical processor. (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. In an asymmetric topology this would be the summation of the value across the lower domain level instances to create each upper domain level instance.) This number reflects configuration as shipped by Intel. Note, software must not use this field to enumerate processor topology*. Bits 31-16: Reserved.

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

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 07-00: The input ECX sub-leaf index. 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.) Hierarchy / Domain / Domain Type Identification Value Lowest / Logical Processor / 1 ... / Core / 2 ... / Module / 3 ... / Tile / 4 ... / Die / 5 ... / DieGrp / 6 Highest / Package/Socket / (implied) (Note that enumeration values of 0 and 7-255 are reserved.) Bits 31-16: Reserved.
	EDX	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. **NOTES:** * 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.
	Processor History Reset Sub-leaf (Initial EAX Value = 20H, ECX = 0)	
20H	EAX	Reports the maximum number of sub-leaves that are supported in leaf 20H.
	EBX	Indicates which bits may be set in the IA32_HRESET_ENABLE MSR to enable reset of different components of hardware-maintained history. 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. Bits 31-01: Reserved = 0.
	ECX	Reserved.
	EDX	Reserved.
	Unimplemented CPUID Leaf Functions	
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.
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.
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information.
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.

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

Initial EAX Value	Information Provided about the Processor	
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved.
	ECX	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.
	EDX	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.
80000006H	EAX	Reserved = 0.
	EBX	Reserved = 0.
	ECX	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.
	EDX	Reserved = 0.

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

Initial EAX Value	Information Provided about the Processor	
		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. If TME-MK is enabled, the number of bits that can be used to address physical memory is CPUID.80000008H:EAX[7:0] - IA32_TME_ACTIVATE[35:32].

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.

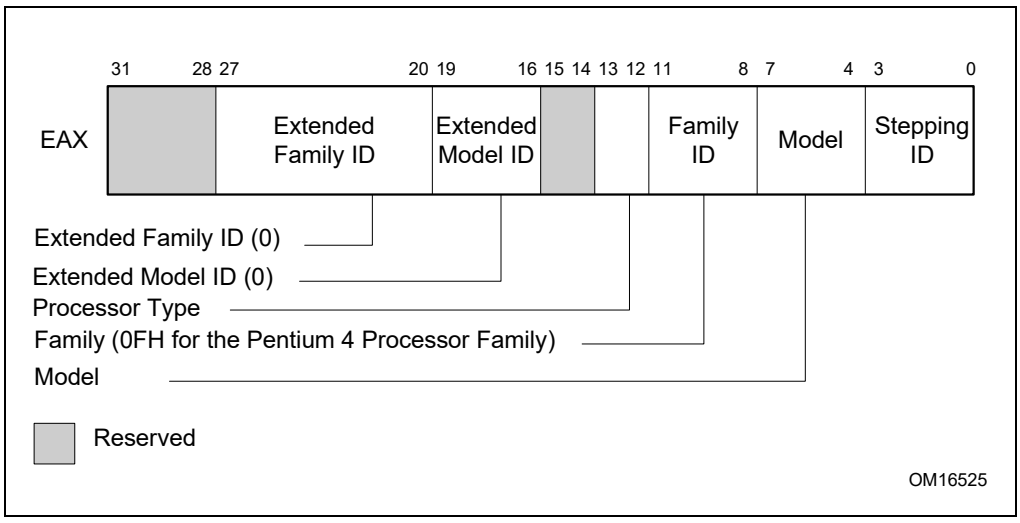


Figure 3-6. Version Information Returned by CPUID in EAX

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.

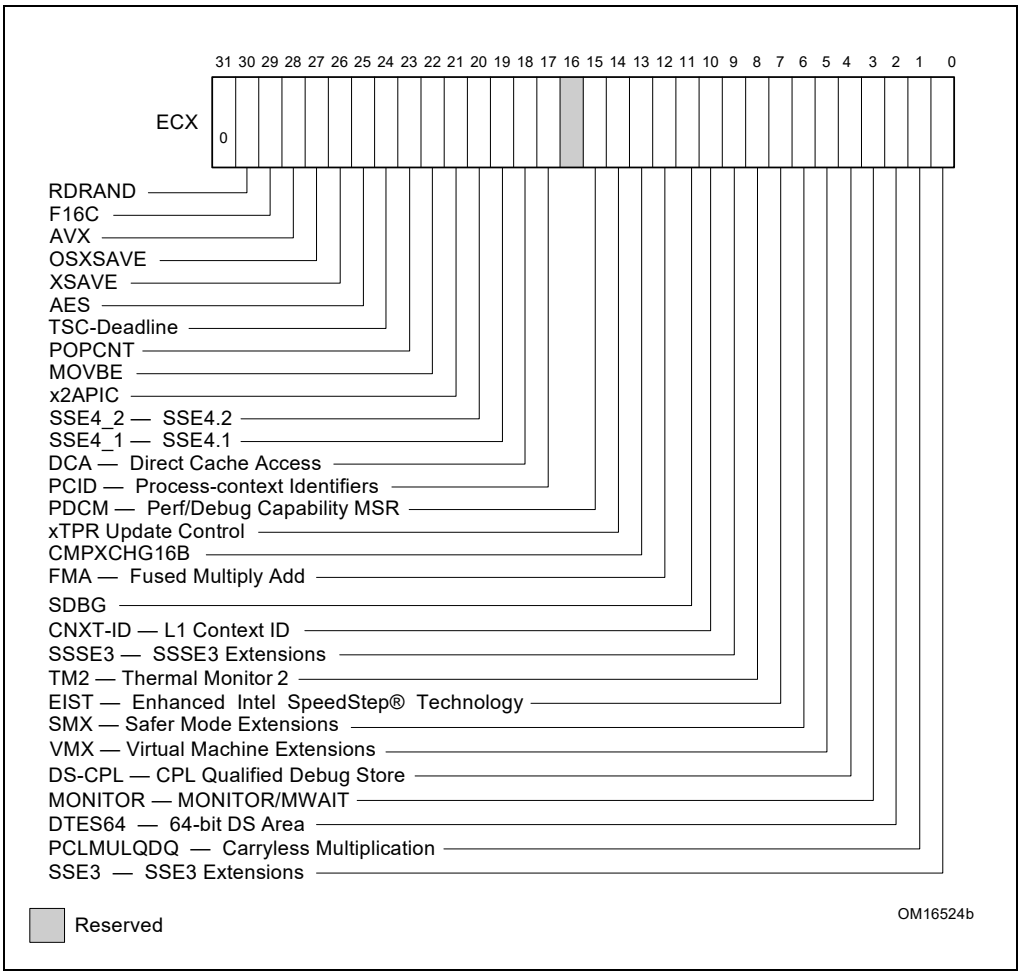


Figure 3-7. Feature Information Returned in the ECX Register

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.

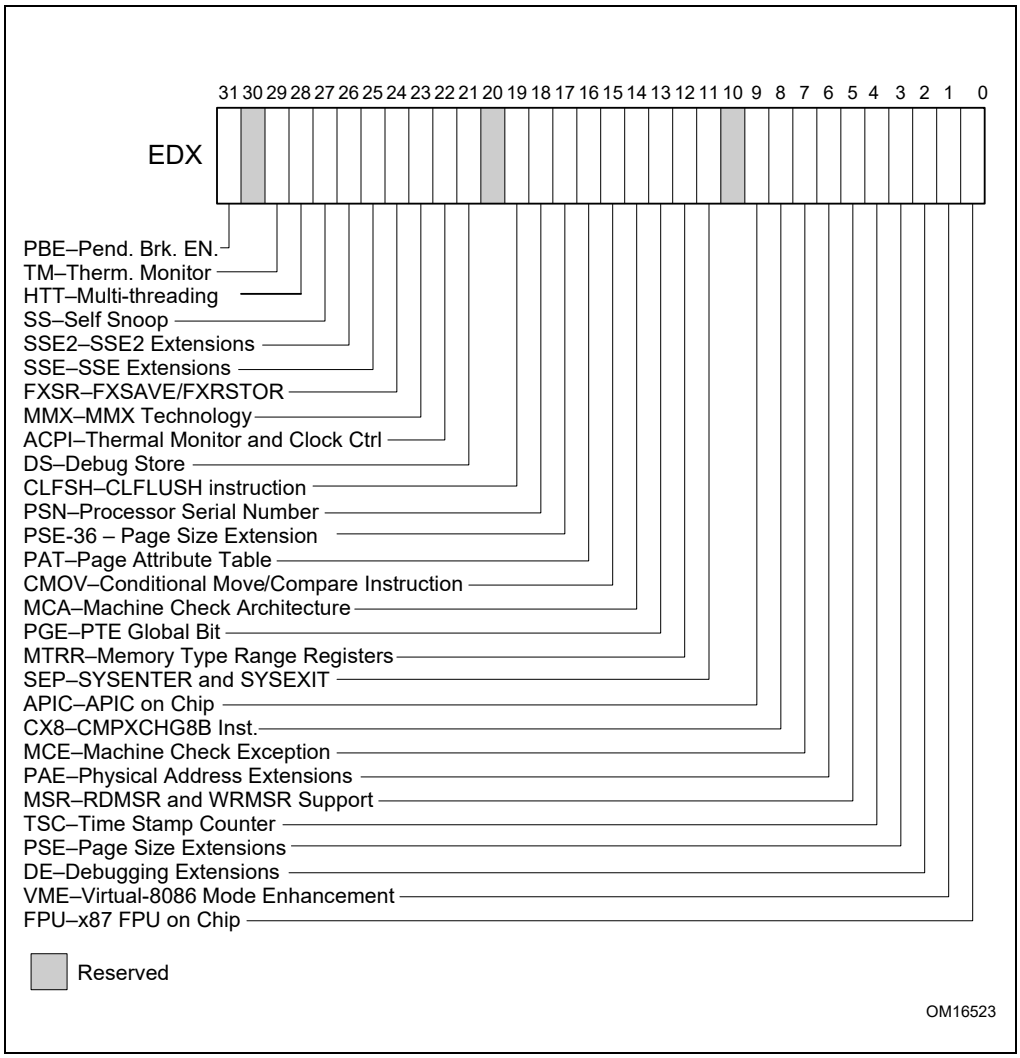


Figure 3-8. Feature Information Returned in the EDX Register

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

Descriptor Value	Type	Cache or TLB 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.)

Descriptor Value	Type	Cache or TLB 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-µop, 8-way set associative.
71H	Cache	Trace cache: 16 K-µop, 8-way set associative.
72H	Cache	Trace cache: 32 K-µop, 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.)

Descriptor Value	Type	Cache or TLB 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 1GBbyte 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

= (Ways + 1) * (Partitions + 1) * (Line_Size + 1) * (Sets + 1)

= (EBX[31:22] + 1) * (EBX[21:12] + 1) * (EBX[11:0] + 1) * (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 >= 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=0H):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 >= 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 TME-MK. 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 >= 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.

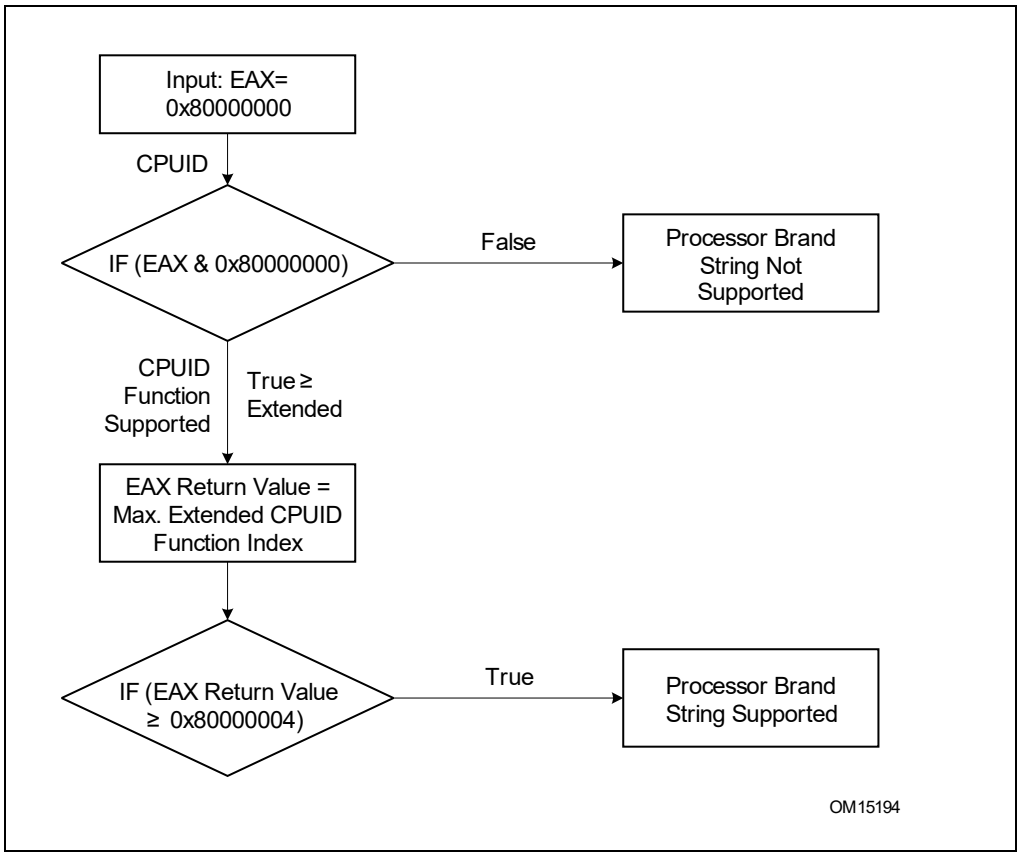


Figure 3-9. Determination of Support for the Processor Brand String

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	" " " " " " "nI "
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.

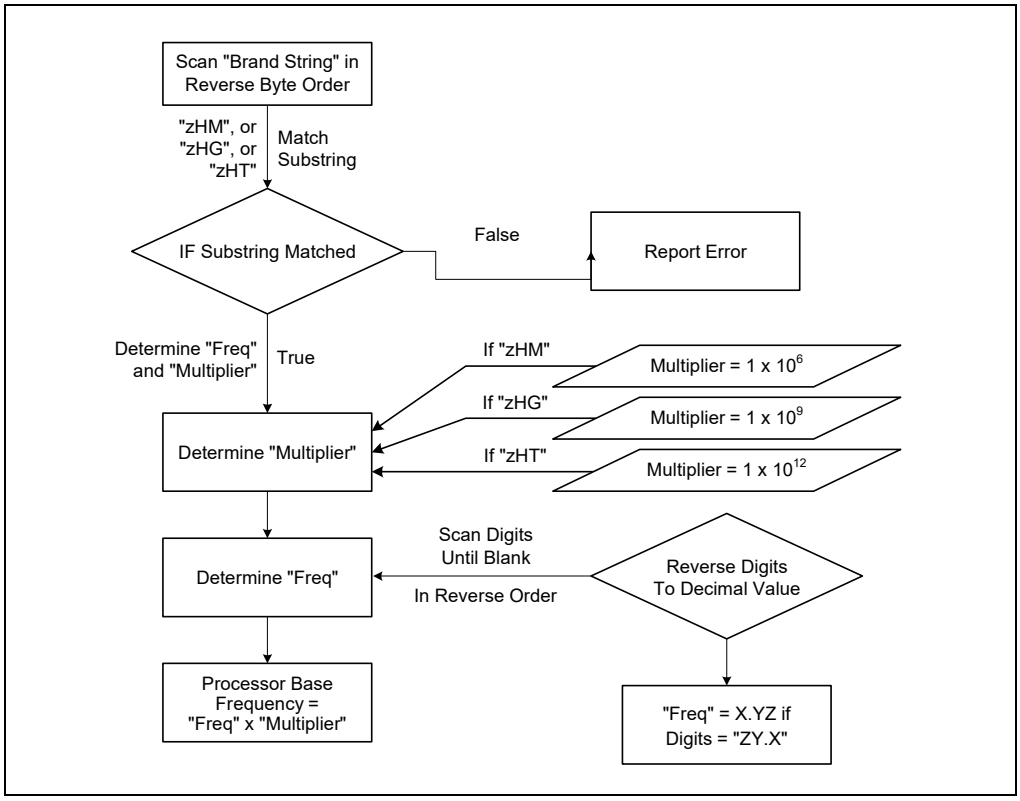


Figure 3-10. Algorithm for Extracting Processor Frequency

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 associate 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 3-323. *)
        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 := Address Size Information;
        EBX := Misc Feature Flags;
        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.

CVTTSS2SI—Convert With Truncation Scalar Single Precision Floating-Point Value to Signed Integer

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 2C /r CVTTSS2SI r32, xmm1/m32	A	V/V	SSE	Convert one single precision floating-point value from xmm1/m32 to one signed doubleword integer in r32 using truncation.
F3 REX.W 0F 2C /r CVTTSS2SI r64, xmm1/m32	A	V/N.E.	SSE	Convert one single precision floating-point value from xmm1/m32 to one signed quadword integer in r64 using truncation.
VEX.LIG.F3.0F.W0 2C /r [1] VCVTTSS2SI r32, xmm1/m32	A	V/V	AVX	Convert one single precision floating-point value from xmm1/m32 to one signed doubleword integer in r32 using truncation.
VEX.LIG.F3.0F.W1 2C /r [1] VCVTTSS2SI r64, xmm1/m32	A	V/N.E.[2]	AVX	Convert one single precision floating-point value from xmm1/m32 to one signed quadword integer in r64 using truncation.
EVEX.LLIG.F3.0F.W0 2C /r VCVTTSS2SI r32, xmm1/m32{sae}	B	V/V	AVX512F	Convert one single precision floating-point value from xmm1/m32 to one signed doubleword integer in r32 using truncation.
EVEX.LLIG.F3.0F.W1 2C /r VCVTTSS2SI r64, xmm1/m32{sae}	B	V/N.E.[2]	AVX512F	Convert one single precision floating-point value from xmm1/m32 to one signed quadword integer in r64 using truncation.

NOTES:

1. Software should ensure VCVTTSS2SI is encoded with VEX.L=0. Encoding VCVTTSS2SI with VEX.L=1 may encounter unpredictable behavior across different processor generations.
2. For this specific instruction, VEX.W/EVEX.W in non-64 bit is ignored; the instructions behaves as if the W0 version is used.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A
B	Tuple1 Fixed	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Converts a single precision floating-point value in the source operand (the second operand) to a signed doubleword integer (or signed quadword integer if operand size is 64 bits) in the destination operand (the first operand). The source operand can be an XMM register or a 32-bit memory location. The destination operand is a general purpose register. When the source operand is an XMM register, the single precision floating-point value is contained in the low doubleword of the register.

When a conversion is inexact, a truncated (round toward zero) result is returned. If a converted result is larger than the maximum signed doubleword integer, the floating-point invalid exception is raised. If this exception is masked, the indefinite integer value (80000000H or 80000000_00000000H if operand size is 64 bits) is returned.

Legacy SSE instructions: In 64-bit mode, Use of the REX.W prefix promotes the instruction to 64-bit operation. See the summary chart at the beginning of this section for encoding data and limits.

VEX.W1 and EVEX.W1 versions: promotes the instruction to produce 64-bit data in 64-bit mode.

Note: VEX.vvvv and EVEX.vvvv are reserved and must be 1111b, otherwise instructions will #UD.

Software should ensure VCVTTSS2SI is encoded with VEX.L=0. Encoding VCVTTSS2SI with VEX.L=1 may encounter unpredictable behavior across different processor generations.

Operation

(V)CVTTSS2SI (All Versions)

```
IF 64-Bit Mode and OperandSize = 64
THEN
    DEST[63:0] := Convert_Single_Precision_Floating_Point_To_Integer_Truncate(SRC[31:0]);
ELSE
    DEST[31:0] := Convert_Single_Precision_Floating_Point_To_Integer_Truncate(SRC[31:0]);
FI;
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VCVTTSS2SI int _mm_cvttss_i32( __m128 a);
VCVTTSS2SI int _mm_cvtt_roundss_i32( __m128 a, int sae);
VCVTTSS2SI __int64 _mm_cvttss_i64( __m128 a);
VCVTTSS2SI __int64 _mm_cvtt_roundss_i64( __m128 a, int sae);
CVTTSS2SI int _mm_cvttss_si32( __m128 a);
CVTTSS2SI __int64 _mm_cvttss_si64( __m128 a);
```

SIMD Floating-Point Exceptions

Invalid, Precision.

Other Exceptions

See Table 2-20, "Type 3 Class Exception Conditions," additionally:

#UD	If VEX.vvvv != 1111B.

EVEX-encoded instructions, see Table 2-48, "Type E3NF Class Exception Conditions."

ENDBR32—Terminate an Indirect Branch in 32-bit and Compatibility Mode

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 1E FB ENDBR32	ZO	V/V	CET_IBT	Terminate indirect branch in 32-bit and compatibility mode.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
ZO	N/A	N/A	N/A	N/A	N/A

Description

Terminate an indirect branch in 32 bit and compatibility mode. This opcode is a NOP when CET indirect branch tracking is not enabled and on processors that do not support CET.

Operation

```
IF EndbranchEnabled(CPL) & (IA32_EFER.LMA = 0 | (IA32_EFER.LMA=1 & CS.L = 0)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = IDLE
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = IDLE
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
```

Flags Affected

None.

Exceptions

#UD	If the LOCK prefix is used.

ENDBR64—Terminate an Indirect Branch in 64-bit Mode

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 1E FA ENDBR64	ZO	V/V	CET_IBT	Terminate indirect branch in 64-bit mode.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
ZO	N/A	N/A	N/A	N/A	N/A

Description

Terminate an indirect branch in 64 bit mode. This opcode is a NOP when CET indirect branch tracking is not enabled and on processors that do not support CET.

Operation

```
IF EndbranchEnabled(CPL) & IA32_EFER.LMA = 1 & CS.L = 1
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = IDLE
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = IDLE
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
```

Flags Affected

None.

Exceptions

#UD	If the LOCK prefix is used.

IMUL—Signed Multiply

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F6 /5	IMUL r/m8	M	Valid	Valid	AX:= AL ∗ r/m byte.
REX + F6 /5	IMUL r/m8[1]	M	Valid	N.E.	AX:= AL ∗ r/m byte.
F7 /5	IMUL r/m16	M	Valid	Valid	DX:AX := AX ∗ r/m word.
F7 /5	IMUL r/m32	M	Valid	Valid	EDX:EAX := EAX ∗ r/m32.
REX.W + F7 /5	IMUL r/m64	M	Valid	N.E.	RDX:RAX := RAX ∗ r/m64.
0F AF /r	IMUL r16, r/m16	RM	Valid	Valid	Word register := word register ∗ r/m16.
0F AF /r	IMUL r32, r/m32	RM	Valid	Valid	Doubleword register := doubleword register ∗ r/m32.
REX.W + 0F AF /r	IMUL r64, r/m64	RM	Valid	N.E.	Quadword register := Quadword register ∗ r/m64.
6B /r ib	IMUL r16, r/m16, imm8	RMI	Valid	Valid	Word register := r/m16 ∗ sign-extended immediate byte.
6B /r ib	IMUL r32, r/m32, imm8	RMI	Valid	Valid	Doubleword register := r/m32 ∗ sign-extended immediate byte.
REX.W + 6B /r ib	IMUL r64, r/m64, imm8	RMI	Valid	N.E.	Quadword register := r/m64 ∗ sign-extended immediate byte.
69 /r iw	IMUL r16, r/m16, imm16	RMI	Valid	Valid	Word register := r/m16 ∗ immediate word.
69 /r id	IMUL r32, r/m32, imm32	RMI	Valid	Valid	Doubleword register := r/m32 ∗ immediate doubleword.
REX.W + 69 /r id	IMUL r64, r/m64, imm32	RMI	Valid	N.E.	Quadword register := r/m64 ∗ immediate doubleword.

NOTES:

1. In 64-bit mode, r/m8 cannot be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r, w)	N/A	N/A	N/A
RM	ModRM:reg (r, w)	ModRM:r/m (r)	N/A	N/A
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	imm8/16/32	N/A

Description

Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.

- **One-operand form** — This form is identical to that used by the MUL instruction. Here, the source operand (in a general-purpose register or memory location) is multiplied by the value in the AL, AX, EAX, or RAX register (depending on the operand size) and the product (twice the size of the input operand) is stored in the AX, DX:AX, EDX:EAX, or RDX:RAX registers, respectively.
- **Two-operand form** — With this form the destination operand (the first operand) is multiplied by the source operand (second operand). The destination operand is a general-purpose register and the source operand is an immediate value, a general-purpose register, or a memory location. The intermediate product (twice the size of the input operand) is truncated and stored in the destination operand location.
- **Three-operand form** — This form requires a destination operand (the first operand) and two source operands (the second and the third operands). Here, the first source operand (which can be a general-purpose register or a memory location) is multiplied by the second source operand (an immediate value). The intermediate

product (twice the size of the first source operand) is truncated and stored in the destination operand (a general-purpose register).

When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The CF and OF flags are set when the signed integer value of the intermediate product differs from the sign extended operand-size-truncated product, otherwise the CF and OF flags are cleared.

The three forms of the IMUL instruction are similar in that the length of the product is calculated to twice the length of the operands. With the one-operand form, the product is stored exactly in the destination. With the two- and three- operand forms, however, the result is truncated to the length of the destination before it is stored in the destination register. Because of this truncation, the CF or OF flag should be tested to ensure that no significant bits are lost.

The two- and three-operand forms may also be used with unsigned operands because the lower half of the product is the same regardless if the operands are signed or unsigned. The CF and OF flags, however, cannot be used to determine if the upper half of the result is non-zero.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. Use of REX.W modifies the three forms of the instruction as follows.

- **One-operand form** —The source operand (in a 64-bit general-purpose register or memory location) is multiplied by the value in the RAX register and the product is stored in the RDX:RAX registers.
- **Two-operand form** — The source operand is promoted to 64 bits if it is a register or a memory location. The destination operand is promoted to 64 bits.
- **Three-operand form** — The first source operand (either a register or a memory location) and destination operand are promoted to 64 bits. If the source operand is an immediate, it is sign extended to 64 bits.

Operation

```
IF (NumberOfOperands = 1)
    THEN IF (OperandSize = 8)
        THEN
            TMP_XP := AL * SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *);
            AX := TMP_XP[15:0];
            IF SignExtend(TMP_XP[7:0]) = TMP_XP
                THEN CF := 0; OF := 0;
                ELSE CF := 1; OF := 1; FI;
        ELSE IF OperandSize = 16
            THEN
                TMP_XP := AX * SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *)
                DX:AX := TMP_XP[31:0];
                IF SignExtend(TMP_XP[15:0]) = TMP_XP
                    THEN CF := 0; OF := 0;
                    ELSE CF := 1; OF := 1; FI;
            ELSE IF OperandSize = 32
                THEN
                    TMP_XP := EAX * SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC*)
                    EDX:EAX := TMP_XP[63:0];
                    IF SignExtend(TMP_XP[31:0]) = TMP_XP
                        THEN CF := 0; OF := 0;
                        ELSE CF := 1; OF := 1; FI;
                ELSE (* OperandSize = 64 *)
                    TMP_XP := RAX * SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *)
                    EDX:EAX := TMP_XP[127:0];
                    IF SignExtend(TMP_XP[63:0]) = TMP_XP
                        THEN CF := 0; OF := 0;
```

```
                    ELSE CF := 1; OF := 1; FI;
                FI;
        FI;
    ELSE IF (NumberOfOperands = 2)
        THEN
            TMP_XP := DEST * SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *)
            DEST := TruncateToOperandSize(TMP_XP);
            IF SignExtend(DEST) ≠ TMP_XP
                THEN CF := 1; OF := 1;
                ELSE CF := 0; OF := 0; FI;
        ELSE (* NumberOfOperands = 3 *)
            TMP_XP := SRC1 * SRC2 (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC1 *)
            DEST := TruncateToOperandSize(TMP_XP);
            IF SignExtend(DEST) ≠ TMP_XP
                THEN CF := 1; OF := 1;
                ELSE CF := 0; OF := 0; FI;
    FI;
FI;
```

Flags Affected

For the one operand form of the instruction, the CF and OF flags are set when significant bits are carried into the upper half of the result and cleared when the result fits exactly in the lower half of the result. For the two- and three-operand forms of the instruction, the CF and OF flags are set when the result must be truncated to fit in the destination operand size and cleared when the result fits exactly in the destination operand size. The SF, ZF, AF, and PF flags are undefined.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register is used to access memory and it contains a NULL NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

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 a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

7. Updates to Chapter 4, Volume 2B

Change bars and violet 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 MOVD/MOVQ instruction pages to correct a typo. Previously, this instruction indicated that the use of REX.R permits access to additional registers (R8-R15), but it is REX.B that permits this. REX.R has no effect, and it is ignored.
- Updated the MOVSX/MOVSXD, MOVZX, and REP/REPE/REPZ/REPNE/REPNZ instructions to add the REX + <opcode> form.
- Updated the PABSB/PABSW/PABSD/PABSQ instruction pages to add PABSB with 64-bit operands information to the operation section.
- Updated the RDSSPD/RDSSPQ instruction page to add an exception to all operating modes indicating that the instruction will #UD if the LOCK prefix is used.
- Updated the UMONITOR instruction to update the list of exceptions. An exception was added to clarify that the instruction will #UD if the LOCK prefix is used.

MOVD/MOVQ—Move Doubleword/Move Quadword

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
NP 0F 6E /r MOVD mm, r/m32	A	V/V	MMX	Move doubleword from r/m32 to mm.
NP REX.W + 0F 6E /r MOVQ mm, r/m64	A	V/N.E.	MMX	Move quadword from r/m64 to mm.
NP 0F 7E /r MOVD r/m32, mm	B	V/V	MMX	Move doubleword from mm to r/m32.
NP REX.W + 0F 7E /r MOVQ r/m64, mm	B	V/N.E.	MMX	Move quadword from mm to r/m64.
66 0F 6E /r MOVD xmm, r/m32	A	V/V	SSE2	Move doubleword from r/m32 to xmm.
66 REX.W 0F 6E /r MOVQ xmm, r/m64	A	V/N.E.	SSE2	Move quadword from r/m64 to xmm.
66 0F 7E /r MOVD r/m32, xmm	B	V/V	SSE2	Move doubleword from xmm register to r/m32.
66 REX.W 0F 7E /r MOVQ r/m64, xmm	B	V/N.E.	SSE2	Move quadword from xmm register to r/m64.
VEX.128.66.0F.W0 6E / VMOVD xmm1, r32/m32	A	V/V	AVX	Move doubleword from r/m32 to xmm1.
VEX.128.66.0F.W1 6E /r VMOVQ xmm1, r64/m64	A	V/N.E[1].	AVX	Move quadword from r/m64 to xmm1.
VEX.128.66.0F.W0 7E /r VMOVD r32/m32, xmm1	B	V/V	AVX	Move doubleword from xmm1 register to r/m32.
VEX.128.66.0F.W1 7E /r VMOVQ r64/m64, xmm1	B	V/N.E[1].	AVX	Move quadword from xmm1 register to r/m64.
EVEX.128.66.0F.W0 6E /r VMOVD xmm1, r32/m32	C	V/V	AVX512F	Move doubleword from r/m32 to xmm1.
EVEX.128.66.0F.W1 6E /r VMOVQ xmm1, r64/m64	C	V/N.E.[1]	AVX512F	Move quadword from r/m64 to xmm1.
EVEX.128.66.0F.W0 7E /r VMOVD r32/m32, xmm1	D	V/V	AVX512F	Move doubleword from xmm1 register to r/m32.
EVEX.128.66.0F.W1 7E /r VMOVQ r64/m64, xmm1	D	V/N.E.[1]	AVX512F	Move quadword from xmm1 register to r/m64.

NOTES:

1. For this specific instruction, VEX.W/EVEX.W in non-64 bit is ignored; the instruction behaves as if the W0 version is used.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A
B	N/A	ModRM:r/m (w)	ModRM:reg (r)	N/A	N/A
C	Tuple1 Scalar	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A
D	Tuple1 Scalar	ModRM:r/m (w)	ModRM:reg (r)	N/A	N/A

Description

Copies a doubleword from the source operand (second operand) to the destination operand (first operand). The source and destination operands can be general-purpose registers, MMX technology registers, XMM registers, or 32-bit memory locations. This instruction can be used to move a doubleword to and from the low doubleword of an MMX technology register and a general-purpose register or a 32-bit memory location, or to and from the low doubleword of an XMM register and a general-purpose register or a 32-bit memory location. The instruction cannot be used to transfer data between MMX technology registers, between XMM registers, between general-purpose registers, or between memory locations.

When the destination operand is an MMX technology register, the source operand is written to the low doubleword of the register, and the register is zero-extended to 64 bits. When the destination operand is an XMM register, the source operand is written to the low doubleword of the register, and the register is zero-extended to 128 bits.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.B prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

MOVD/Q with XMM destination:

Moves a dword/qword integer from the source operand and stores it in the low 32/64-bits of the destination XMM register. The upper bits of the destination are zeroed. The source operand can be a 32/64-bit register or 32/64-bit memory location.

128-bit Legacy SSE version: Bits (MAXVL-1:128) of the corresponding YMM destination register remain unchanged. Qword operation requires the use of REX.W=1.

VEX.128 encoded version: Bits (MAXVL-1:128) of the destination register are zeroed. Qword operation requires the use of VEX.W=1.

EVEX.128 encoded version: Bits (MAXVL-1:128) of the destination register are zeroed. Qword operation requires the use of EVEX.W=1.

MOVD/Q with 32/64 reg/mem destination:

Stores the low dword/qword of the source XMM register to 32/64-bit memory location or general-purpose register. Qword operation requires the use of REX.W=1, VEX.W=1, or EVEX.W=1.

Note: VEX.vvvv and EVEX.vvvv are reserved and must be 1111b otherwise instructions will #UD.

If VMOVD or VMOVQ is encoded with VEX.L= 1, an attempt to execute the instruction encoded with VEX.L= 1 will cause an #UD exception.

Operation

MOVD (When Destination Operand is an MMX Technology Register)

```
DEST[31:0] := SRC;
DEST[63:32] := 00000000H;
```

MOVD (When Destination Operand is an XMM Register)

```
DEST[31:0] := SRC;
DEST[127:32] := 000000000000000000000000H;
DEST[MAXVL-1:128] (Unmodified)
```

MOVD (When Source Operand is an MMX Technology or XMM Register)

```
DEST := SRC[31:0];
```

VMOVD (VEX-Encoded Version when Destination is an XMM Register)

```
DEST[31:0] := SRC[31:0]
DEST[MAXVL-1:32] := 0
```

MOVQ (When Destination Operand is an XMM Register)

```
DEST[63:0] := SRC[63:0];
DEST[127:64] := 0000000000000000H;
DEST[MAXVL-1:128] (Unmodified)
```

MOVQ (When Destination Operand is r/m64)

```
DEST[63:0] := SRC[63:0];
```

MOVQ (When Source Operand is an XMM Register or r/m64)

```
DEST := SRC[63:0];
```

VMOVQ (VEX-Encoded Version When Destination is an XMM Register)

```
DEST[63:0] := SRC[63:0]
DEST[MAXVL-1:64] := 0
```

VMOVD (EVEX-Encoded Version When Destination is an XMM Register)

```
DEST[31:0] := SRC[31:0]
DEST[MAXVL-1:32] := 0
```

VMOVQ (EVEX-Encoded Version When Destination is an XMM Register)

```
DEST[63:0] := SRC[63:0]
DEST[MAXVL-1:64] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

MOVD __m64 _mm_cvtsi32_si64 (int i)
MOVD int _mm_cvtsi64_si32 (__m64m)
MOVD __m128i _mm_cvtsi32_si128 (int a)
MOVD int _mm_cvtsi128_si32 (__m128i a)
MOVQ __int64 _mm_cvtsi128_si64(__m128i);
MOVQ __m128i _mm_cvtsi64_si128(__int64);
VMOVD __m128i _mm_cvtsi32_si128(int);
VMOVD int _mm_cvtsi128_si32(__m128i);
VMOVQ __m128i _mm_cvtsi64_si128 (__int64);
VMOVQ __int64 _mm_cvtsi128_si64(__m128i);
VMOVQ __m128i _mm_loadl_epi64(__m128i * s);
VMOVQ void _mm_storel_epi64(__m128i * d, __m128i s);

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-22, "Type 5 Class Exception Conditions."
EVEX-encoded instruction, see Table 2-57, "Type E9NF Class Exception Conditions."
Additionally:

#UD	If VEX.L = 1.
	If VEX.vvvv != 1111B or EVEX.vvvv != 1111B.

MOVSX/MOVSXD—Move With Sign-Extension

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F BE /r	MOVSX r16, r/m8	RM	Valid	Valid	Move byte to word with sign-extension.
0F BE /r	MOVSX r32, r/m8	RM	Valid	Valid	Move byte to doubleword with sign-extension.
REX + 0F BE /r	MOVSX r16/r32, r/m8[1]	RM	Valid	N.E.	Move byte to word/doubleword with sign-extension.
REX.W + 0F BE /r	MOVSX r64, r/m8[1]	RM	Valid	N.E.	Move byte to quadword with sign-extension.
0F BF /r	MOVSX r32, r/m16	RM	Valid	Valid	Move word to doubleword, with sign-extension.
REX.W + 0F BF /r	MOVSX r64, r/m16	RM	Valid	N.E.	Move word to quadword with sign-extension.
63 /r[2]	MOVSXD r16, r/m16	RM	Valid	N.E.	Move word to word with sign-extension.
63 /r[1]	MOVSXD r32, r/m32	RM	Valid	N.E.	Move doubleword to doubleword with sign-extension.
REX.W + 63 /r	MOVSXD r64, r/m32	RM	Valid	N.E.	Move doubleword to quadword with sign-extension.

NOTES:

1. In 64-bit mode, r/m8 cannot be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.
2. The use of MOVSXD without REX.W in 64-bit mode is discouraged. Regular MOV should be used instead of using MOVSXD without REX.W.

Instruction Operand Encoding

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

Description

Copies the contents of the source operand (register or memory location) to the destination operand (register) and sign extends the value to 16 or 32 bits (see Figure 7-6 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). The size of the converted value depends on the operand-size attribute.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
DEST := SignExtend(SRC);
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.

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 a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

MOVZX—Move With Zero-Extend

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F B6 /r	MOVZX r16, r/m8	RM	Valid	Valid	Move byte to word with zero-extension.
0F B6 /r	MOVZX r32, r/m8	RM	Valid	Valid	Move byte to doubleword, zero-extension.
REX + 0F B6 /r	MOVZX r16/r32, r/m8[1]	RM	Valid	N.E.	Move byte to word/doubleword, zero-extension.
REX.W + 0F B6 /r	MOVZX r64, r/m8[1]	RM	Valid	N.E.	Move byte to quadword, zero-extension.
0F B7 /r	MOVZX r32, r/m16	RM	Valid	Valid	Move word to doubleword, zero-extension.
REX.W + 0F B7 /r	MOVZX r64, r/m16	RM	Valid	N.E.	Move word to quadword, zero-extension.

NOTES:

1. In 64-bit mode, r/m8 cannot be encoded to access the following byte registers if the REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

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

Description

Copies the contents of the source operand (register or memory location) to the destination operand (register) and zero extends the value. The size of the converted value depends on the operand-size attribute.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bit operands. See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
DEST := ZeroExtend(SRC);
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

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 a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

PABSB/PABSW/PABSD/PABSQ—Packed Absolute Value

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 38 1C /r[1] PABSB mm1, mm2/m64	A	V/V	SSSE3	Compute the absolute value of bytes in mm2/m64 and store UNSIGNED result in mm1.
66 0F 38 1C /r PABSB xmm1, xmm2/m128	A	V/V	SSSE3	Compute the absolute value of bytes in xmm2/m128 and store UNSIGNED result in xmm1.
NP 0F 38 1D /r[1] PABSW mm1, mm2/m64	A	V/V	SSSE3	Compute the absolute value of 16-bit integers in mm2/m64 and store UNSIGNED result in mm1.
66 0F 38 1D /r PABSW xmm1, xmm2/m128	A	V/V	SSSE3	Compute the absolute value of 16-bit integers in xmm2/m128 and store UNSIGNED result in xmm1.
NP 0F 38 1E /r[1] PABSD mm1, mm2/m64	A	V/V	SSSE3	Compute the absolute value of 32-bit integers in mm2/m64 and store UNSIGNED result in mm1.
66 0F 38 1E /r PABSD xmm1, xmm2/m128	A	V/V	SSSE3	Compute the absolute value of 32-bit integers in xmm2/m128 and store UNSIGNED result in xmm1.
VEX.128.66.0F38.WIG 1C /r VPABSB xmm1, xmm2/m128	A	V/V	AVX	Compute the absolute value of bytes in xmm2/m128 and store UNSIGNED result in xmm1.
VEX.128.66.0F38.WIG 1D /r VPABSW xmm1, xmm2/m128	A	V/V	AVX	Compute the absolute value of 16- bit integers in xmm2/m128 and store UNSIGNED result in xmm1.
VEX.128.66.0F38.WIG 1E /r VPABSD xmm1, xmm2/m128	A	V/V	AVX	Compute the absolute value of 32- bit integers in xmm2/m128 and store UNSIGNED result in xmm1.
VEX.256.66.0F38.WIG 1C /r VPABSB ymm1, ymm2/m256	A	V/V	AVX2	Compute the absolute value of bytes in ymm2/m256 and store UNSIGNED result in ymm1.
VEX.256.66.0F38.WIG 1D /r VPABSW ymm1, ymm2/m256	A	V/V	AVX2	Compute the absolute value of 16-bit integers in ymm2/m256 and store UNSIGNED result in ymm1.
VEX.256.66.0F38.WIG 1E /r VPABSD ymm1, ymm2/m256	A	V/V	AVX2	Compute the absolute value of 32-bit integers in ymm2/m256 and store UNSIGNED result in ymm1.
EVEX.128.66.0F38.WIG 1C /r VPABSB xmm1 {k1}{z}, xmm2/m128	B	V/V	AVX512VL AVX512BW	Compute the absolute value of bytes in xmm2/m128 and store UNSIGNED result in xmm1 using writemask k1.
EVEX.256.66.0F38.WIG 1C /r VPABSB ymm1 {k1}{z}, ymm2/m256	B	V/V	AVX512VL AVX512BW	Compute the absolute value of bytes in ymm2/m256 and store UNSIGNED result in ymm1 using writemask k1.
EVEX.512.66.0F38.WIG 1C /r VPABSB zmm1 {k1}{z}, zmm2/m512	B	V/V	AVX512BW	Compute the absolute value of bytes in zmm2/m512 and store UNSIGNED result in zmm1 using writemask k1.
EVEX.128.66.0F38.WIG 1D /r VPABSW xmm1 {k1}{z}, xmm2/m128	B	V/V	AVX512VL AVX512BW	Compute the absolute value of 16-bit integers in xmm2/m128 and store UNSIGNED result in xmm1 using writemask k1.
EVEX.256.66.0F38.WIG 1D /r VPABSW ymm1 {k1}{z}, ymm2/m256	B	V/V	AVX512VL AVX512BW	Compute the absolute value of 16-bit integers in ymm2/m256 and store UNSIGNED result in ymm1 using writemask k1.
EVEX.512.66.0F38.WIG 1D /r VPABSW zmm1 {k1}{z}, zmm2/m512	B	V/V	AVX512BW	Compute the absolute value of 16-bit integers in zmm2/m512 and store UNSIGNED result in zmm1 using writemask k1.

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F38.W0 1E /r VPABSD xmm1 {k1}{z}, xmm2/m128/m32bcst	C	V/V	AVX512VL AVX512F	Compute the absolute value of 32-bit integers in xmm2/m128/m32bcst and store UNSIGNED result in xmm1 using writemask k1.
EVEX.256.66.0F38.W0 1E /r VPABSD ymm1 {k1}{z}, ymm2/m256/m32bcst	C	V/V	AVX512VL AVX512F	Compute the absolute value of 32-bit integers in ymm2/m256/m32bcst and store UNSIGNED result in ymm1 using writemask k1.
EVEX.512.66.0F38.W0 1E /r VPABSD zmm1 {k1}{z}, zmm2/m512/m32bcst	C	V/V	AVX512F	Compute the absolute value of 32-bit integers in zmm2/m512/m32bcst and store UNSIGNED result in zmm1 using writemask k1.
EVEX.128.66.0F38.W1 1F /r VPABSQ xmm1 {k1}{z}, xmm2/m128/m64bcst	C	V/V	AVX512VL AVX512F	Compute the absolute value of 64-bit integers in xmm2/m128/m64bcst and store UNSIGNED result in xmm1 using writemask k1.
EVEX.256.66.0F38.W1 1F /r VPABSQ ymm1 {k1}{z}, ymm2/m256/m64bcst	C	V/V	AVX512VL AVX512F	Compute the absolute value of 64-bit integers in ymm2/m256/m64bcst and store UNSIGNED result in ymm1 using writemask k1.
EVEX.512.66.0F38.W1 1F /r VPABSQ zmm1 {k1}{z}, zmm2/m512/m64bcst	C	V/V	AVX512F	Compute the absolute value of 64-bit integers in zmm2/m512/m64bcst and store UNSIGNED result in zmm1 using writemask k1.

NOTES:

1. See note in Section 2.5, "Intel® AVX and Intel® SSE Instruction Exception Classification," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, and Section 23.25.3, "Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A
B	Full Mem	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A
C	Full	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

PABSB/W/D computes the absolute value of each data element of the source operand (the second operand) and stores the UNSIGNED results in the destination operand (the first operand). PABSB operates on signed bytes, PABSW operates on signed 16-bit words, and PABSD operates on signed 32-bit integers.

EVEX encoded VPABSD/Q: The source operand is a ZMM/YMM/XMM register, a 512/256/128-bit memory location, or a 512/256/128-bit vector broadcasted from a 32/64-bit memory location. The destination operand is a ZMM/YMM/XMM register updated according to the writemask.

EVEX encoded VPABSB/W: The source operand is a ZMM/YMM/XMM register, or a 512/256/128-bit memory location. The destination operand is a ZMM/YMM/XMM register updated according to the writemask.

VEX.256 encoded versions: The source operand is a YMM register or a 256-bit memory location. The destination operand is a YMM register. The upper bits (MAXVL-1:256) of the corresponding register destination are zeroed.

VEX.128 encoded versions: The source operand is an XMM register or 128-bit memory location. The destination operand is an XMM register. The upper bits (MAXVL-1:128) of the corresponding register destination are zeroed.

128-bit Legacy SSE version: The source operand can be an XMM register or an 128-bit memory location. The destination is an XMM register. The upper bits (VL_MAX-1:128) of the corresponding register destination are unmodified.

VEX.vvvv and EVEX.vvvv are reserved and must be 1111b otherwise instructions will #UD.

Operation

PABSB With 64-bit Operands:

```
Unsigned DEST[7:0] := ABS(SRC[7: 0])
Repeat operation for 2nd through 7th bytes
Unsigned DEST[63:56] := ABS(SRC[63:56])
```

PABSB With 128-bit Operands:

```
Unsigned DEST[7:0] := ABS(SRC[7: 0])
Repeat operation for 2nd through 15th bytes
Unsigned DEST[127:120] := ABS(SRC[127:120])
```

VPABSB With 128-bit Operands:

```
Unsigned DEST[7:0] := ABS(SRC[7: 0])
Repeat operation for 2nd through 15th bytes
Unsigned DEST[127:120] := ABS(SRC[127:120])
```

VPABSB With 256-bit Operands:

```
Unsigned DEST[7:0] := ABS(SRC[7: 0])
Repeat operation for 2nd through 31st bytes
Unsigned DEST[255:248] := ABS(SRC[255:248])
```

VPABSB (EVEX Encoded Versions)

```
(KL, VL) = (16, 128), (32, 256), (64, 512)

FOR j := 0 TO KL-1
    i := j * 8
    IF k1[j] OR *no writemask*
        THEN
            Unsigned DEST[i+7:i] := ABS(SRC[i+7:i])
        ELSE
            IF *merging-masking*                    ; merging-masking
                THEN *DEST[i+7:i] remains unchanged*
                ELSE *zeroing-masking*              ; zeroing-masking
                    DEST[i+7:i] := 0
            FI
    FI;
ENDFOR;
DEST[MAXVL-1:VL] := 0
```

PABSW With 128-bit Operands:

```
Unsigned DEST[15:0] := ABS(SRC[15:0])
Repeat operation for 2nd through 7th 16-bit words
Unsigned DEST[127:112] := ABS(SRC[127:112])
```

VPABSW With 128-bit Operands:

```
Unsigned DEST[15:0] := ABS(SRC[15:0])
Repeat operation for 2nd through 7th 16-bit words
Unsigned DEST[127:112] := ABS(SRC[127:112])
```

VPABSW With 256-bit Operands:

```
Unsigned DEST[15:0] := ABS(SRC[15:0])
Repeat operation for 2nd through 15th 16-bit words
Unsigned DEST[255:240] := ABS(SRC[255:240])
```

VPABSW (EVEX Encoded Versions)

```
    (KL, VL) = (8, 128), (16, 256), (32, 512)

FOR j := 0 TO KL-1
    i := j * 16
    IF k1[j] OR *no writemask*
        THEN
            Unsigned DEST[i+15:i] := ABS(SRC[i+15:i])
        ELSE
            IF *merging-masking*                    ; merging-masking
                THEN *DEST[i+15:i] remains unchanged*
                ELSE *zeroing-masking*              ; zeroing-masking
                    DEST[i+15:i] := 0
            FI
    FI;
ENDFOR;
DEST[MAXVL-1:VL] := 0
```

PABSD With 128-bit Operands:

```
    Unsigned DEST[31:0] := ABS(SRC[31:0])
    Repeat operation for 2nd through 3rd 32-bit double words
    Unsigned DEST[127:96] := ABS(SRC[127:96])
```

VPABSD With 128-bit Operands:

```
    Unsigned DEST[31:0] := ABS(SRC[31:0])
    Repeat operation for 2nd through 3rd 32-bit double words
    Unsigned DEST[127:96] := ABS(SRC[127:96])
```

VPABSD With 256-bit Operands:

```
    Unsigned DEST[31:0] := ABS(SRC[31:0])
    Repeat operation for 2nd through 7th 32-bit double words
    Unsigned DEST[255:224] := ABS(SRC[255:224])
```

VPABSD (EVEX Encoded Versions)

```
(KL, VL) = (4, 128), (8, 256), (16, 512)
FOR j := 0 TO KL-1
    i := j * 32
    IF k1[j] OR *no writemask*
        THEN
            IF (EVEX.b = 1) AND (SRC *is memory*)
                THEN
                    Unsigned DEST[i+31:i] := ABS(SRC[31:0])
                ELSE
                    Unsigned DEST[i+31:i] := ABS(SRC[i+31:i])
            FI;
        ELSE
            IF *merging-masking*                    ; merging-masking
                THEN *DEST[i+31:i] remains unchanged*
                ELSE *zeroing-masking*              ; zeroing-masking
                    DEST[i+31:i] := 0
            FI
    FI;
ENDFOR;
DEST[MAXVL-1:VL] := 0
```

VPABSQ (EVEX Encoded Versions)

```
(KL, VL) = (2, 128), (4, 256), (8, 512)
FOR j := 0 TO KL-1
    i := j * 64
    IF k1[j] OR *no writemask*
        THEN
            IF (EVEX.b = 1) AND (SRC *is memory*)
                THEN
                    Unsigned DEST[i+63:i] := ABS(SRC[63:0])
                ELSE
                    Unsigned DEST[i+63:i] := ABS(SRC[i+63:i])
            FI;
        ELSE
            IF *merging-masking*                    ; merging-masking
                THEN *DEST[i+63:i] remains unchanged*
                ELSE *zeroing-masking*              ; zeroing-masking
                    DEST[i+63:i] := 0
            FI
    FI;
ENDFOR;
DEST[MAXVL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalents

VPABSB __m512i _mm512_abs_epi8 (__m512i a)
VPABSW __m512i _mm512_abs_epi16 (__m512i a)
VPABSB __m512i _mm512_mask_abs_epi8 (__m512i s, __mmask64 m, __m512i a)
VPABSW __m512i _mm512_mask_abs_epi16 (__m512i s, __mmask32 m, __m512i a)
VPABSB __m512i _mm512_maskz_abs_epi8 (__mmask64 m, __m512i a)
VPABSW __m512i _mm512_maskz_abs_epi16 (__mmask32 m, __m512i a)
VPABSB __m256i _mm256_mask_abs_epi8 (__m256i s, __mmask32 m, __m256i a)
VPABSW __m256i _mm256_mask_abs_epi16 (__m256i s, __mmask16 m, __m256i a)
VPABSB __m256i _mm256_maskz_abs_epi8 (__mmask32 m, __m256i a)
VPABSW __m256i _mm256_maskz_abs_epi16 (__mmask16 m, __m256i a)
VPABSB __m128i _mm_mask_abs_epi8 (__m128i s, __mmask16 m, __m128i a)
VPABSW __m128i _mm_mask_abs_epi16 (__m128i s, __mmask8 m, __m128i a)
VPABSB __m128i _mm_maskz_abs_epi8 (__mmask16 m, __m128i a)
VPABSW __m128i _mm_maskz_abs_epi16 (__mmask8 m, __m128i a)
VPABSD __m256i _mm256_mask_abs_epi32(__m256i s, __mmask8 k, __m256i a);
VPABSD __m256i _mm256_maskz_abs_epi32(__mmask8 k, __m256i a);
VPABSD __m128i _mm_mask_abs_epi32(__m128i s, __mmask8 k, __m128i a);
VPABSD __m128i _mm_maskz_abs_epi32(__mmask8 k, __m128i a);
VPABSD __m512i _mm512_abs_epi32(__m512i a);
VPABSD __m512i _mm512_mask_abs_epi32(__m512i s, __mmask16 k, __m512i a);
VPABSD __m512i _mm512_maskz_abs_epi32(__mmask16 k, __m512i a);
VPABSQ __m512i _mm512_abs_epi64(__m512i a);
VPABSQ __m512i _mm512_mask_abs_epi64(__m512i s, __mmask8 k, __m512i a);
VPABSQ __m512i _mm512_maskz_abs_epi64(__mmask8 k, __m512i a);
VPABSQ __m256i _mm256_mask_abs_epi64(__m256i s, __mmask8 k, __m256i a);
VPABSQ __m256i _mm256_maskz_abs_epi64(__mmask8 k, __m256i a);
VPABSQ __m128i _mm_mask_abs_epi64(__m128i s, __mmask8 k, __m128i a);
VPABSQ __m128i _mm_maskz_abs_epi64(__mmask8 k, __m128i a);
PABSB __m128i _mm_abs_epi8 (__m128i a)

VPABSB __m128i _mm_abs_epi8 (__m128i a)
VPABSB __m256i _mm256_abs_epi8 (__m256i a)
PABSW __m128i _mm_abs_epi16 (__m128i a)
VPABSW __m128i _mm_abs_epi16 (__m128i a)
VPABSW __m256i _mm256_abs_epi16 (__m256i a)
PABSD __m128i _mm_abs_epi32 (__m128i a)
VPABSD __m128i _mm_abs_epi32 (__m128i a)
VPABSD __m256i _mm256_abs_epi32 (__m256i a)

SIMD Floating-Point Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-21, "Type 4 Class Exception Conditions."
EVEX-encoded VPABSD/Q, see Table 2-49, "Type E4 Class Exception Conditions."
EVEX-encoded VPABSB/W, see Exceptions Type E4.nb in Table 2-49, "Type E4 Class Exception Conditions."

RDSSPD/RDSSPQ—Read Shadow Stack Pointer

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 1E /1 (mod=11) RDSSPD r32	R	V/V	CET_SS	Copy low 32 bits of shadow stack pointer (SSP) to r32.
F3 REX.W 0F 1E /1 (mod=11) RDSSPQ r64	R	V/N.E.	CET_SS	Copies shadow stack pointer (SSP) to r64.

Instruction Operand Encoding

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

Description

Copies the current shadow stack pointer (SSP) register to the register destination. This opcode is a NOP when CET shadow stacks are not enabled and on processors that do not support CET.

Operation

```
IF CPL = 3
    IF CR4.CET & IA32_U_CET.SH_STK_EN
        IF (operand size is 64 bit)
            THEN
                Dest := SSP;
            ELSE
                Dest := SSP[31:0];
        FI;
    FI;
ELSE
    IF CR4.CET & IA32_S_CET.SH_STK_EN
        IF (operand size is 64 bit)
            THEN
                Dest := SSP;
            ELSE
                Dest := SSP[31:0];
        FI;
    FI;
FI;
```

Flags Affected

None.

C/C++ Compiler Intrinsic Equivalent

RDSSPD__int32 _rdsspd_i32(void);
RDSSPQ__int64 _rdsspq_i64(void);

Exceptions (All Operating Modes)

#UD	If the LOCK prefix is used.

REP/REPE/REPZ/REPNE/REPNZ—Repeat String Operation Prefix

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F3 6C	REP INS m8, DX	ZO	Valid	Valid	Input (E)CX bytes from port DX into ES:[(E)DI].
F3 6C	REP INS m8, DX	ZO	Valid	N.E.	Input RCX bytes from port DX into [RDI].
F3 6D	REP INS m16, DX	ZO	Valid	Valid	Input (E)CX words from port DX into ES:[(E)DI.]
F3 6D	REP INS m32, DX	ZO	Valid	Valid	Input (E)CX doublewords from port DX into ES:[(E)DI].
F3 6D	REP INS r/m32, DX	ZO	Valid	N.E.	Input RCX default size from port DX into [RDI].
F3 A4	REP MOVS m8, m8	ZO	Valid	Valid	Move (E)CX bytes from DS:[(E)SI] to ES:[(E)DI].
F3 REX.W A4	REP MOVS m8, m8	ZO	Valid	N.E.	Move RCX bytes from [RSI] to [RDI].
F3 A5	REP MOVS m16, m16	ZO	Valid	Valid	Move (E)CX words from DS:[(E)SI] to ES:[(E)DI].
F3 A5	REP MOVS m32, m32	ZO	Valid	Valid	Move (E)CX doublewords from DS:[(E)SI] to ES:[(E)DI].
F3 REX.W A5	REP MOVS m64, m64	ZO	Valid	N.E.	Move RCX quadwords from [RSI] to [RDI].
F3 6E	REP OUTS DX, r/m8	ZO	Valid	Valid	Output (E)CX bytes from DS:[(E)SI] to port DX.
F3 REX 6E	REP OUTS DX, r/m8[1]	ZO	Valid	N.E.	Output (E)CX bytes from DS:[(E)SI] to port DX.
F3 REX.W 6E	REP OUTS DX, r/m8[1]	ZO	Valid	N.E.	Output RCX bytes from [RSI] to port DX.
F3 6F	REP OUTS DX, r/m16	ZO	Valid	Valid	Output (E)CX words from DS:[(E)SI] to port DX.
F3 6F	REP OUTS DX, r/m32	ZO	Valid	Valid	Output (E)CX doublewords from DS:[(E)SI] to port DX.
F3 REX.W 6F	REP OUTS DX, r/m32	ZO	Valid	N.E.	Output RCX default size from [RSI] to port DX.
F3 AC	REP LODS AL	ZO	Valid	Valid	Load (E)CX bytes from DS:[(E)SI] to AL.
F3 REX.W AC	REP LODS AL	ZO	Valid	N.E.	Load RCX bytes from [RSI] to AL.
F3 AD	REP LODS AX	ZO	Valid	Valid	Load (E)CX words from DS:[(E)SI] to AX.
F3 AD	REP LODS EAX	ZO	Valid	Valid	Load (E)CX doublewords from DS:[(E)SI] to EAX.
F3 REX.W AD	REP LODS RAX	ZO	Valid	N.E.	Load RCX quadwords from [RSI] to RAX.
F3 AA	REP STOS m8	ZO	Valid	Valid	Fill (E)CX bytes at ES:[(E)DI] with AL.
F3 REX.W AA	REP STOS m8	ZO	Valid	N.E.	Fill RCX bytes at [RDI] with AL.
F3 AB	REP STOS m16	ZO	Valid	Valid	Fill (E)CX words at ES:[(E)DI] with AX.
F3 AB	REP STOS m32	ZO	Valid	Valid	Fill (E)CX doublewords at ES:[(E)DI] with EAX.
F3 REX.W AB	REP STOS m64	ZO	Valid	N.E.	Fill RCX quadwords at [RDI] with RAX.
F3 A6	REPE CMPS m8, m8	ZO	Valid	Valid	Find nonmatching bytes in ES:[(E)DI] and DS:[(E)SI].
F3 REX.W A6	REPE CMPS m8, m8	ZO	Valid	N.E.	Find non-matching bytes in [RDI] and [RSI].
F3 A7	REPE CMPS m16, m16	ZO	Valid	Valid	Find nonmatching words in ES:[(E)DI] and DS:[(E)SI].
F3 A7	REPE CMPS m32, m32	ZO	Valid	Valid	Find nonmatching doublewords in ES:[(E)DI] and DS:[(E)SI].
F3 REX.W A7	REPE CMPS m64, m64	ZO	Valid	N.E.	Find non-matching quadwords in [RDI] and [RSI].
F3 AE	REPE SCAS m8	ZO	Valid	Valid	Find non-AL byte starting at ES:[(E)DI].
F3 REX.W AE	REPE SCAS m8	ZO	Valid	N.E.	Find non-AL byte starting at [RDI].
F3 AF	REPE SCAS m16	ZO	Valid	Valid	Find non-AX word starting at ES:[(E)DI].

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F3 AF	REPE SCAS m32	ZO	Valid	Valid	Find non-EAX doubleword starting at ES:[(E)DI].
F3 REX.W AF	REPE SCAS m64	ZO	Valid	N.E.	Find non-RAX quadword starting at [RDI].
F2 A6	REPNE CMPS m8, m8	ZO	Valid	Valid	Find matching bytes in ES:[(E)DI] and DS:[(E)SI].
F2 REX.W A6	REPNE CMPS m8, m8	ZO	Valid	N.E.	Find matching bytes in [RDI] and [RSI].
F2 A7	REPNE CMPS m16, m16	ZO	Valid	Valid	Find matching words in ES:[(E)DI] and DS:[(E)SI].
F2 A7	REPNE CMPS m32, m32	ZO	Valid	Valid	Find matching doublewords in ES:[(E)DI] and DS:[(E)SI].
F2 REX.W A7	REPNE CMPS m64, m64	ZO	Valid	N.E.	Find matching doublewords in [RDI] and [RSI].
F2 AE	REPNE SCAS m8	ZO	Valid	Valid	Find AL, starting at ES:[(E)DI].
F2 REX.W AE	REPNE SCAS m8	ZO	Valid	N.E.	Find AL, starting at [RDI].
F2 AF	REPNE SCAS m16	ZO	Valid	Valid	Find AX, starting at ES:[(E)DI].
F2 AF	REPNE SCAS m32	ZO	Valid	Valid	Find EAX, starting at ES:[(E)DI].
F2 REX.W AF	REPNE SCAS m64	ZO	Valid	N.E.	Find RAX, starting at [RDI].

NOTES:

1. In 64-bit mode, r/m8 cannot be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

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

Description

Repeats a string instruction the number of times specified in the count register or until the indicated condition of the ZF flag is no longer met. The REP (repeat), REPE (repeat while equal), REPNE (repeat while not equal), REPZ (repeat while zero), and REPNZ (repeat while not zero) mnemonics are prefixes that can be added to one of the string instructions. The REP prefix can be added to the INS, OUTS, MOVS, LODS, and STOS instructions, and the REPE, REPNE, REPZ, and REPNZ prefixes can be added to the CMPS and SCAS instructions. (The REPZ and REPNZ prefixes are synonymous forms of the REPE and REPNE prefixes, respectively.) The F3H prefix is defined for the following instructions and undefined for the rest:

- F3H as REP/REPE/REPZ for string and input/output instruction.
- F3H is a mandatory prefix for POPCNT, LZCNT, and ADOX.

The REP prefixes apply only to one string instruction at a time. To repeat a block of instructions, use the LOOP instruction or another looping construct. All of these repeat prefixes cause the associated instruction to be repeated until the count in register is decremented to 0. See Table 4-17.

Table 4-17. Repeat Prefixes

Repeat Prefix	Termination Condition 1*	Termination Condition 2
REP	RCX or (E)CX = 0	None
REPE/REPZ	RCX or (E)CX = 0	ZF = 0
REPNE/REPNZ	RCX or (E)CX = 0	ZF = 1

NOTES:

* Count register is CX, ECX or RCX by default, depending on attributes of the operating modes.

The REPE, REPNE, REPZ, and REPNZ prefixes also check the state of the ZF flag after each iteration and terminate the repeat loop if the ZF flag is not in the specified state. When both termination conditions are tested, the cause of a repeat termination can be determined either by testing the count register with a JECXZ instruction or by testing the ZF flag (with a JZ, JNZ, or JNE instruction).

When the REPE/REPZ and REPNE/REPNZ prefixes are used, the ZF flag does not require initialization because both the CMPS and SCAS instructions affect the ZF flag according to the results of the comparisons they make.

A repeating string operation can be suspended by an exception or interrupt. When this happens, the state of the registers is preserved to allow the string operation to be resumed upon a return from the exception or interrupt handler. The source and destination registers point to the next string elements to be operated on, the EIP register points to the string instruction, and the ECX register has the value it held following the last successful iteration of the instruction. This mechanism allows long string operations to proceed without affecting the interrupt response time of the system.

When a fault occurs during the execution of a CMPS or SCAS instruction that is prefixed with REPE or REPNE, the EFLAGS value is restored to the state prior to the execution of the instruction. Since the SCAS and CMPS instructions do not use EFLAGS as an input, the processor can resume the instruction after the page fault handler.

Use the REP INS and REP OUTS instructions with caution. Not all I/O ports can handle the rate at which these instructions execute. Note that a REP STOS instruction is the fastest way to initialize a large block of memory.

In 64-bit mode, the operand size of the count register is associated with the address size attribute. Thus the default count register is RCX; REX.W has no effect on the address size and the count register. In 64-bit mode, if 67H is used to override address size attribute, the count register is ECX and any implicit source/destination operand will use the corresponding 32-bit index register. See the summary chart at the beginning of this section for encoding data and limits.

REP INS may read from the I/O port without writing to the memory location if an exception or VM exit occurs due to the write (e.g., #PF). If this would be problematic, for example because the I/O port read has side-effects, software should ensure the write to the memory location does not cause an exception or VM exit.

Operation

```
IF AddressSize = 16
   THEN
      Use CX for CountReg;
      Implicit Source/Dest operand for memory use of SI/DI;
   ELSE IF AddressSize = 64
      THEN Use RCX for CountReg;
      Implicit Source/Dest operand for memory use of RSI/RDI;
   ELSE
      Use ECX for CountReg;
      Implicit Source/Dest operand for memory use of ESI/EDI;
FI;
WHILE CountReg ≠ 0
    DO
         Service pending interrupts (if any);
         Execute associated string instruction;
         CountReg := (CountReg – 1);
         IF CountReg = 0
               THEN exit WHILE loop; FI;
         IF (Repeat prefix is REPZ or REPE) and (ZF = 0)
         or (Repeat prefix is REPNZ or REPNE) and (ZF = 1)
               THEN exit WHILE loop; FI;
    OD;
```

Flags Affected

None; however, the CMPS and SCAS instructions do set the status flags in the EFLAGS register.

Exceptions (All Operating Modes)

Exceptions may be generated by an instruction associated with the prefix.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.

UMONITOR—User Level Set Up Monitor Address

Opcode / Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F AE /6 UMONITOR r16/r32/r64	A	V/V	WAITPKG	Sets up a linear address range to be monitored by hardware and activates the monitor. The address range should be a write-back memory caching type. The address is contained in r16/r32/r64.

Instruction Operand Encoding[1]

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

Description

The UMONITOR instruction arms address monitoring hardware using an address specified in the source register (the address range that the monitoring hardware checks for store operations can be determined by using the CPUID monitor leaf function, EAX=05H). A store to an address within the specified address range triggers the monitoring hardware. The state of monitor hardware is used by UMWAIT.

The content of the source register is an effective address. By default, the DS segment is used to create a linear address that is monitored. Segment overrides can be used. The address range must use memory of the write-back type. Only write-back memory is guaranteed to correctly trigger the monitoring hardware. Additional information on determining what address range to use in order to prevent false wake-ups is described in Chapter 9, "Multiple-Processor Management," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

The UMONITOR instruction is ordered as a load operation with respect to other memory transactions. The instruction is subject to the permission checking and faults associated with a byte load. Like a load, UMONITOR sets the A-bit but not the D-bit in page tables.

UMONITOR and UMWAIT are available when CPUID.7.0:ECX.WAITPKG[bit 5] is enumerated as 1. UMONITOR and UMWAIT may be executed at any privilege level. Except for the width of the source register, the instruction's operation is the same in non-64-bit modes and in 64-bit mode.

UMONITOR does not interoperate with the legacy MWAIT instruction. If UMONITOR was executed prior to executing MWAIT and following the most recent execution of the legacy MONITOR instruction, MWAIT will not enter an optimized state. Execution will continue to the instruction following MWAIT.

The UMONITOR instruction causes a transactional abort when used inside a transactional region.

The width of the source register (16b, 32b or 64b) is determined by the effective addressing width, which is affected in the standard way by the machine mode settings and 67 prefix.

Operation

UMONITOR sets up an address range for the monitor hardware using the content of source register as an effective address and puts the monitor hardware in armed state. A store to the specified address range will trigger the monitor hardware.

Intel C/C++ Compiler Intrinsic Equivalent

UMONITOR void _umonitor(void *address);

Numeric Exceptions

None.

1. The Mod field of the ModR/M byte must have value 11B.

Protected Mode Exceptions

#GP(0)	If the specified segment is not SS and the source register is outside the specified segment limit.
	If the specified segment register contains a NULL segment selector.
#SS(0)	If the specified segment is SS and the source register is outside the SS segment limit.
#PF(fault-code)	For a page fault.
#UD	If CPUID.7.0:ECX.WAITPKG[bit 5]=0.
	If the LOCK prefix is used.

Real Address Mode Exceptions

#GP	If the specified segment is not SS and the source register is outside of the effective address space from 0 to FFFFH.
#SS	If the specified segment is SS and the source register is outside of the effective address space from 0 to FFFFH.
#UD	If CPUID.7.0:ECX.WAITPKG[bit 5]=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

#GP(0)	If the specified segment is not SS and the linear address is in non-canonical form.
#SS(0)	If the specified segment is SS and the source register is in non-canonical form.
#PF(fault-code)	For a page fault.
#UD	If CPUID.7.0:ECX.WAITPKG[bit 5]=0.
	If the LOCK prefix is used.

8. Updates to Chapter 5, Volume 2C

Change bars and violet 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 VFIXUPIMMPD, VFIXUPIMMPS, VFIXUPIMMSD, and VFIXUPIMMSS instruction intrinsics.

VFIXUPIMMPD—Fix Up Special Packed Float64 Values

Opcode/ Instruction	Op / En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F3A.W1 54 /r ib VFIXUPIMMPD xmm1 {k1}{z}, xmm2, xmm3/m128/m64bcst, imm8	A	V/V	AVX512VL AVX512F	Fix up special numbers in float64 vector xmm1, float64 vector xmm2 and int64 vector xmm3/m128/m64bcst and store the result in xmm1, under writemask.
EVEX.256.66.0F3A.W1 54 /r ib VFIXUPIMMPD ymm1 {k1}{z}, ymm2, ymm3/m256/m64bcst, imm8	A	V/V	AVX512VL AVX512F	Fix up special numbers in float64 vector ymm1, float64 vector ymm2 and int64 vector ymm3/m256/m64bcst and store the result in ymm1, under writemask.
EVEX.512.66.0F3A.W1 54 /r ib VFIXUPIMMPD zmm1 {k1}{z}, zmm2, zmm3/m512/m64bcst{sae}, imm8	A	V/V	AVX512F	Fix up elements of float64 vector in zmm2 using int64 vector table in zmm3/m512/m64bcst, combine with preserved elements from zmm1, and store the result in zmm1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Perform fix-up of quad-word elements encoded in double precision floating-point format in the first source operand (the second operand) using a 32-bit, two-level look-up table specified in the corresponding quadword element of the second source operand (the third operand) with exception reporting specifier imm8. The elements that are fixed-up are selected by mask bits of 1 specified in the opmask k1. Mask bits of 0 in the opmask k1 or table response action of 0000b preserves the corresponding element of the first operand. The fixed-up elements from the first source operand and the preserved element in the first operand are combined as the final results in the destination operand (the first operand).

The destination and the first source operands are ZMM/YMM/XMM registers. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcasted from a 64-bit memory location.

The two-level look-up table perform a fix-up of each double precision floating-point input data in the first source operand by decoding the input data encoding into 8 token types. A response table is defined for each token type that converts the input encoding in the first source operand with one of 16 response actions.

This instruction is specifically intended for use in fixing up the results of arithmetic calculations involving one source so that they match the spec, although it is generally useful for fixing up the results of multiple-instruction sequences to reflect special-number inputs. For example, consider rcp(0). Input 0 to rcp, and you should get INF according to the DX10 spec. However, evaluating rcp via Newton-Raphson, where x=approx(1/0), yields an incorrect result. To deal with this, VFIXUPIMMPD can be used after the N-R reciprocal sequence to set the result to the correct value (i.e., INF when the input is 0).

If MXCSR.DAZ is not set, denormal input elements in the first source operand are considered as normal inputs and do not trigger any fixup nor fault reporting.

Imm8 is used to set the required flags reporting. It supports #ZE and #IE fault reporting (see details below).

MXCSR mask bits are ignored and are treated as if all mask bits are set to masked response). If any of the imm8 bits is set and the condition met for fault reporting, MXCSR.IE or MXCSR.ZE might be updated.

This instruction is writemasked, so only those elements with the corresponding bit set in vector mask register k1 are computed and stored into zmm1. Elements in the destination with the corresponding bit clear in k1 retain their previous values or are set to 0.

Operation

```
enum TOKEN_TYPE
{
    QNAN_TOKEN := 0,
    SNAN_TOKEN := 1,
    ZERO_VALUE_TOKEN := 2,
    POS_ONE_VALUE_TOKEN := 3,
    NEG_INF_TOKEN := 4,
    POS_INF_TOKEN := 5,
    NEG_VALUE_TOKEN := 6,
    POS_VALUE_TOKEN := 7
}

FIXUPIMM_DP (dest[63:0], src1[63:0],tbl3[63:0], imm8 [7:0]){
    tsrc[63:0] := ((src1[62:52] = 0) AND (MXCSR.DAZ =1)) ? 0.0 : src1[63:0]
    CASE(tsrc[63:0] of TOKEN_TYPE) {
        QNAN_TOKEN: j := 0;
        SNAN_TOKEN: j := 1;
        ZERO_VALUE_TOKEN: j := 2;
        POS_ONE_VALUE_TOKEN: j := 3;
        NEG_INF_TOKEN: j := 4;
        POS_INF_TOKEN: j := 5;
        NEG_VALUE_TOKEN: j := 6;
        POS_VALUE_TOKEN: j := 7;
    }    ; end source special CASE(tsrc...)

    ; The required response from src3 table is extracted
    token_response[3:0] = tbl3[3+4*j:4*j];

    CASE(token_response[3:0]) {
        0000: dest[63:0] := dest[63:0];          ; preserve content of DEST
        0001: dest[63:0] := tsrc[63:0];          ; pass through src1 normal input value, denormal as zero
        0010: dest[63:0] := QNaN(tsrc[63:0]);
        0011: dest[63:0] := QNAN_Indefinite;
        0100: dest[63:0] := -INF;
        0101: dest[63:0] := +INF;
        0110: dest[63:0] := tsrc.sign? -INF : +INF;
        0111: dest[63:0] := -0;
        1000: dest[63:0] := +0;
        1001: dest[63:0] := -1;
        1010: dest[63:0] := +1;
        1011: dest[63:0] := ½;
        1100: dest[63:0] := 90.0;
        1101: dest[63:0] := PI/2;
        1110: dest[63:0] := MAX_FLOAT;
        1111: dest[63:0] := -MAX_FLOAT;
    }          ; end of token_response CASE

    ; The required fault reporting from imm8 is extracted
    ; TOKENs are mutually exclusive and TOKENs priority defines the order.
    ; Multiple faults related to a single token can occur simultaneously.
    IF (tsrc[63:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[0] then set #ZE;
    IF (tsrc[63:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[1] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[2] then set #ZE;
```

```
    IF (tsrc[63:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[3] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: SNAN_TOKEN) AND imm8[4] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: NEG_INF_TOKEN) AND imm8[5] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: NEG_VALUE_TOKEN) AND imm8[6] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: POS_INF_TOKEN) AND imm8[7] then set #IE;
        ; end fault reporting
    return dest[63:0];
}       ; end of FIXUPIMM_DP()
```

VFIXUPIMMPD

```
(KL, VL) = (2, 128), (4, 256), (8, 512)
FOR j := 0 TO KL-1
    i := j * 64
    IF k1[j] OR *no writemask*
        THEN
            IF (EVEX.b = 1) AND (SRC2 *is memory*)
                THEN
                    DEST[i+63:i] := FIXUPIMM_DP(DEST[i+63:i], SRC1[i+63:i], SRC2[63:0], imm8 [7:0])
                ELSE
                    DEST[i+63:i] := FIXUPIMM_DP(DEST[i+63:i], SRC1[i+63:i], SRC2[i+63:i], imm8 [7:0])
            FI;
        ELSE
            IF *merging-masking*                    ; merging-masking
                THEN *DEST[i+63:i] remains unchanged*
                ELSE  DEST[i+63:i] := 0             ; zeroing-masking
            FI
    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0
```

Immediate Control Description:

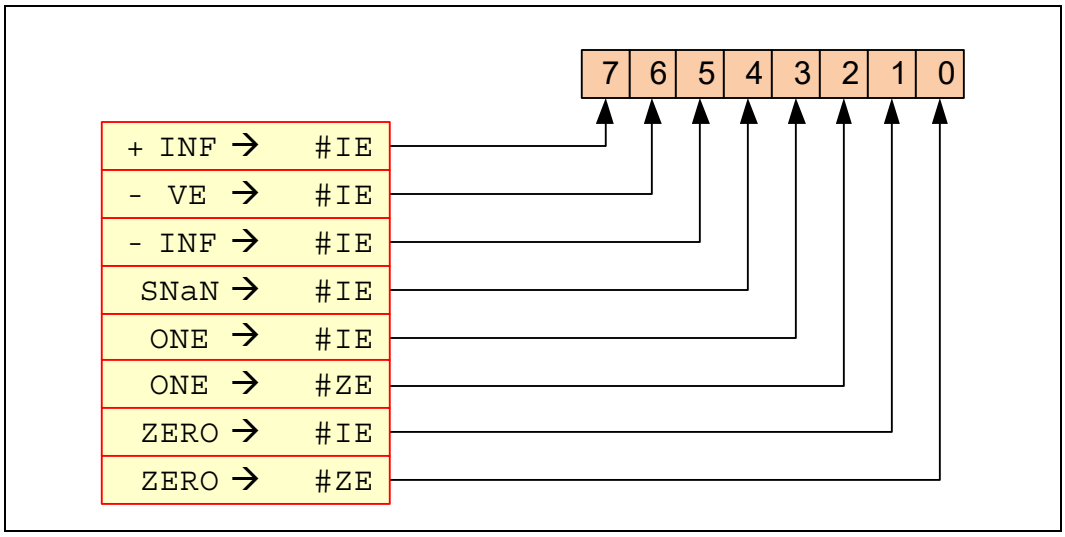


Figure 5-9. VFIXUPIMMPD Immediate Control Description

Intel C/C++ Compiler Intrinsic Equivalent

VFIXUPIMMPD __m512d _mm512_fixupimm_pd(__m512d a, __m512d b, __m512i c, int imm8);
VFIXUPIMMPD __m512d _mm512_mask_fixupimm_pd(__m512d a, __mmask8 k, __m512d b, __m512i c, int imm8);
VFIXUPIMMPD __m512d _mm512_maskz_fixupimm_pd(__mmask8 k, __m512d a, __m512d b, __m512i c, int imm8);
VFIXUPIMMPD __m512d _mm512_fixupimm_round_pd(__m512d a, __m512d b, __m512i c, int imm8, int sae);
VFIXUPIMMPD __m512d _mm512_mask_fixupimm_round_pd(__m512d a, __mmask8 k, __m512d b, __m512i c, int imm8, int sae);
VFIXUPIMMPD __m512d _mm512_maskz_fixupimm_round_pd(__mmask8 k, __m512d a, __m512d b, __m512i c, int imm8, int sae);
VFIXUPIMMPD __m256d _mm256_fixupimm_pd(__m256d a, m256d b, __m256i c, int imm8);
VFIXUPIMMPD __m256d _mm256_mask_fixupimm_pd(__m256d a, __mmask8 k, __m256d b, __m256i c, int imm8);
VFIXUPIMMPD __m256d _mm256_maskz_fixupimm_pd(__mmask8 k, __m256d a, __m256d b, __m256i c, int imm8);
VFIXUPIMMPD __m128d _mm_fixupimm_pd(__m128d a, __m128d b, __m128i c, int imm8);
VFIXUPIMMPD __m128d _mm_mask_fixupimm_pd(__m128d a, __mmask8 k, __m128d b, __m128i c, int imm8);
VFIXUPIMMPD __m128d _mm_maskz_fixupimm_pd(__mmask8 k, __m128d a, __m128d b, __m128i c, int imm8);

SIMD Floating-Point Exceptions

Zero, Invalid.

Other Exceptions

See Table 2-46, "Type E2 Class Exception Conditions."

VFIXUPIMMPS—Fix Up Special Packed Float32 Values

Opcode/ Instruction	Op/ En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F3A.W0 54 /r VFIXUPIMMPS xmm1 {k1}{z}, xmm2, xmm3/m128/m32bcst, imm8	A	V/V	AVX512VL AVX512F	Fix up special numbers in float32 vector xmm1, float32 vector xmm2 and int32 vector xmm3/m128/m32bcst and store the result in xmm1, under writemask.
EVEX.256.66.0F3A.W0 54 /r VFIXUPIMMPS ymm1 {k1}{z}, ymm2, ymm3/m256/m32bcst, imm8	A	V/V	AVX512VL AVX512F	Fix up special numbers in float32 vector ymm1, float32 vector ymm2 and int32 vector ymm3/m256/m32bcst and store the result in ymm1, under writemask.
EVEX.512.66.0F3A.W0 54 /r ib VFIXUPIMMPS zmm1 {k1}{z}, zmm2, zmm3/m512/m32bcst{sae}, imm8	A	V/V	AVX512F	Fix up elements of float32 vector in zmm2 using int32 vector table in zmm3/m512/m32bcst, combine with preserved elements from zmm1, and store the result in zmm1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Perform fix-up of doubleword elements encoded in single precision floating-point format in the first source operand (the second operand) using a 32-bit, two-level look-up table specified in the corresponding doubleword element of the second source operand (the third operand) with exception reporting specifier imm8. The elements that are fixed-up are selected by mask bits of 1 specified in the opmask k1. Mask bits of 0 in the opmask k1 or table response action of 0000b preserves the corresponding element of the first operand. The fixed-up elements from the first source operand and the preserved element in the first operand are combined as the final results in the destination operand (the first operand).

The destination and the first source operands are ZMM/YMM/XMM registers. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcasted from a 64-bit memory location.

The two-level look-up table perform a fix-up of each single precision floating-point input data in the first source operand by decoding the input data encoding into 8 token types. A response table is defined for each token type that converts the input encoding in the first source operand with one of 16 response actions.

This instruction is specifically intended for use in fixing up the results of arithmetic calculations involving one source so that they match the spec, although it is generally useful for fixing up the results of multiple-instruction sequences to reflect special-number inputs. For example, consider rcp(0). Input 0 to rcp, and you should get INF according to the DX10 spec. However, evaluating rcp via Newton-Raphson, where x=approx(1/0), yields an incorrect result. To deal with this, VFIXUPIMMPS can be used after the N-R reciprocal sequence to set the result to the correct value (i.e., INF when the input is 0).

If MXCSR.DAZ is not set, denormal input elements in the first source operand are considered as normal inputs and do not trigger any fixup nor fault reporting.

Imm8 is used to set the required flags reporting. It supports #ZE and #IE fault reporting (see details below).

MXCSR.DAZ is used and refer to zmm2 only (i.e., zmm1 is not considered as zero in case MXCSR.DAZ is set).

MXCSR mask bits are ignored and are treated as if all mask bits are set to masked response). If any of the imm8 bits is set and the condition met for fault reporting, MXCSR.IE or MXCSR.ZE might be updated.

Operation

```
enum TOKEN_TYPE
{
    QNAN_TOKEN := 0,
    SNAN_TOKEN := 1,
    ZERO_VALUE_TOKEN := 2,
    POS_ONE_VALUE_TOKEN := 3,
    NEG_INF_TOKEN := 4,
    POS_INF_TOKEN := 5,
    NEG_VALUE_TOKEN := 6,
    POS_VALUE_TOKEN := 7
}

FIXUPIMM_SP ( dest[31:0], src1[31:0],tbl3[31:0], imm8 [7:0]){
    tsrc[31:0] := ((src1[30:23] = 0) AND (MXCSR.DAZ =1)) ? 0.0 : src1[31:0]
    CASE(tsrc[31:0] of TOKEN_TYPE) {
        QNAN_TOKEN: j := 0;
        SNAN_TOKEN: j := 1;
        ZERO_VALUE_TOKEN: j := 2;
        POS_ONE_VALUE_TOKEN: j := 3;
        NEG_INF_TOKEN: j := 4;
        POS_INF_TOKEN: j := 5;
        NEG_VALUE_TOKEN: j := 6;
        POS_VALUE_TOKEN: j := 7;
    }           ; end source special CASE(tsrc...)

    ; The required response from src3 table is extracted
    token_response[3:0] = tbl3[3+4*j:4*j];

    CASE(token_response[3:0]) {
        0000: dest[31:0] := dest[31:0];         ; preserve content of DEST
        0001: dest[31:0] := tsrc[31:0];         ; pass through src1 normal input value, denormal as zero
        0010: dest[31:0] := QNaN(tsrc[31:0]);
        0011: dest[31:0] := QNAN_Indefinite;
        0100: dest[31:0] := -INF;
        0101: dest[31:0] := +INF;
        0110: dest[31:0] := tsrc.sign? –INF : +INF;
        0111: dest[31:0] := -0;
        1000: dest[31:0] := +0;
        1001: dest[31:0] := -1;
        1010: dest[31:0] := +1;
        1011:  dest[31:0] := ½;
        1100: dest[31:0] := 90.0;
        1101: dest[31:0] := PI/2;
        1110: dest[31:0] := MAX_FLOAT;
        1111: dest[31:0] := -MAX_FLOAT;
    }           ; end of token_response CASE

    ; The required fault reporting from imm8 is extracted
    ; TOKENs are mutually exclusive and TOKENs priority defines the order.
    ; Multiple faults related to a single token can occur simultaneously.
    IF (tsrc[31:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[0] then set #ZE;
    IF (tsrc[31:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[1] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[2] then set #ZE;
```

```
    IF (tsrc[31:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[3] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: SNAN_TOKEN) AND imm8[4] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: NEG_INF_TOKEN) AND imm8[5] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: NEG_VALUE_TOKEN) AND imm8[6] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: POS_INF_TOKEN) AND imm8[7] then set #IE;
        ; end fault reporting
    return dest[31:0];
}       ; end of FIXUPIMM_SP()
```

VFIXUPIMMPS (EVEX)

(KL, VL) = (4, 128), (8, 256), (16, 512)

```
FOR j := 0 TO KL-1
    i := j * 32
    IF k1[j] OR *no writemask*
        THEN
            IF (EVEX.b = 1) AND (SRC2 *is memory*)
                THEN
                    DEST[i+31:i] := FIXUPIMM_SP(DEST[i+31:i], SRC1[i+31:i], SRC2[31:0], imm8 [7:0])
                ELSE
                    DEST[i+31:i] := FIXUPIMM_SP(DEST[i+31:i], SRC1[i+31:i], SRC2[i+31:i], imm8 [7:0])
            FI;
        ELSE
            IF *merging-masking*                ; merging-masking
                THEN *DEST[i+31:i] remains unchanged*
                ELSE  DEST[i+31:i] := 0         ; zeroing-masking
            FI
    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0
```

Immediate Control Description:

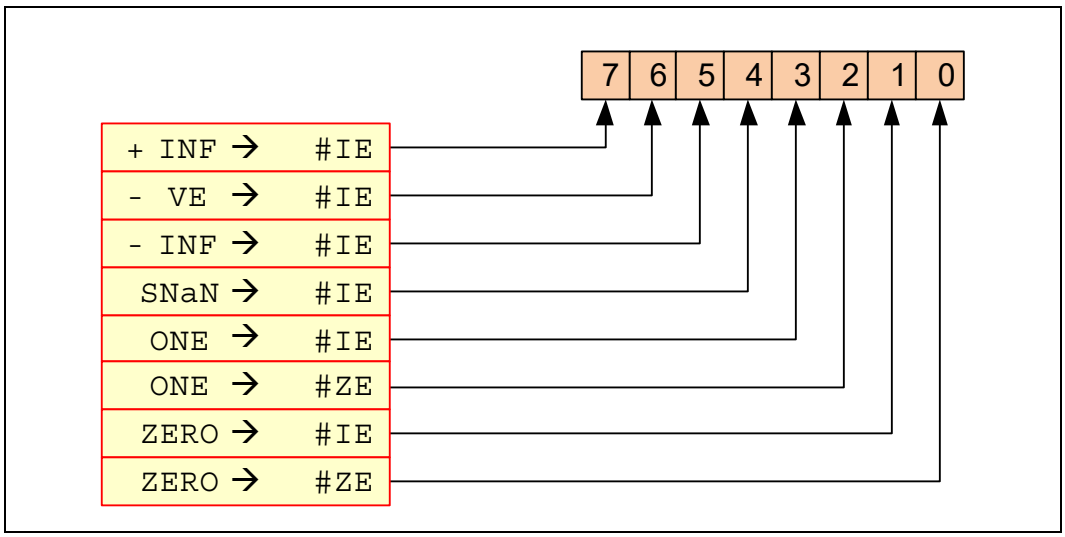


Figure 5-10. VFIXUPIMMPS Immediate Control Description

Intel C/C++ Compiler Intrinsic Equivalent

VFIXUPIMMPS __m512 _mm512_fixupimm_ps(__m512 a, __m512 b, __m512i c, int imm8);
VFIXUPIMMPS __m512 _mm512_mask_fixupimm_ps(__m512 a, __mmask16 k, __m512 b, __m512i c, int imm8);
VFIXUPIMMPS __m512 _mm512_maskz_fixupimm_ps(__mmask16 k, __m512 a, __m512 b, __m512i c, int imm8);
VFIXUPIMMPS __m512 _mm512_fixupimm_round_ps(__m512 a, __m512 b, __m512i c, int imm8, int sae);
VFIXUPIMMPS __m512 _mm512_mask_fixupimm_round_ps(__m512 a, __mmask16 k, __m512 b, __m512i c, int imm8, int sae);
VFIXUPIMMPS __m512 _mm512_maskz_fixupimm_round_ps(__mmask16 k, __m512 a, __m512 b, __m512i c, int imm8, int sae);
VFIXUPIMMPS __m256 _mm256_fixupimm_ps(__m256 a, __m256 b, __m256i c, int imm8);
VFIXUPIMMPS __m256 _mm256_mask_fixupimm_ps(__m256 a, __mmask8 k, __m256 b, __m256i c, int imm8);
VFIXUPIMMPS __m256 _mm256_maskz_fixupimm_ps(__mmask8 k, __m256 a, __m256 b, __m256i c, int imm8);
VFIXUPIMMPS __m128 _mm_fixupimm_ps(__m128 a, __m128 b, __m128i c, int imm8);
VFIXUPIMMPS __m128 _mm_mask_fixupimm_ps(__m128 a, __mmask8 k, __m128 b, __m128i c, int imm8);
VFIXUPIMMPS __m128 _mm_maskz_fixupimm_ps(__mmask8 k, __m128 a, __m128 b, __m128i c, int imm8);

SIMD Floating-Point Exceptions

Zero, Invalid.

Other Exceptions

See Table 2-46, "Type E2 Class Exception Conditions."

VFIXUPIMMSD—Fix Up Special Scalar Float64 Value

Opcode/ Instruction	Op/ En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.LLIG.66.0F3A.W1 55 /r ib VFIXUPIMMSD xmm1 {k1}{z}, xmm2, xmm3/m64{sae}, imm8	A	V/V	AVX512F	Fix up a float64 number in the low quadword element of xmm2 using scalar int32 table in xmm3/m64 and store the result in xmm1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Perform a fix-up of the low quadword element encoded in double precision floating-point format in the first source operand (the second operand) using a 32-bit, two-level look-up table specified in the low quadword element of the second source operand (the third operand) with exception reporting specifier imm8. The element that is fixed-up is selected by mask bit of 1 specified in the opmask k1. Mask bit of 0 in the opmask k1 or table response action of 0000b preserves the corresponding element of the first operand. The fixed-up element from the first source operand or the preserved element in the first operand becomes the low quadword element of the destination operand (the first operand). Bits 127:64 of the destination operand is copied from the corresponding bits of the first source operand. The destination and first source operands are XMM registers. The second source operand can be a XMM register or a 64- bit memory location.

The two-level look-up table perform a fix-up of each double precision floating-point input data in the first source operand by decoding the input data encoding into 8 token types. A response table is defined for each token type that converts the input encoding in the first source operand with one of 16 response actions.

This instruction is specifically intended for use in fixing up the results of arithmetic calculations involving one source so that they match the spec, although it is generally useful for fixing up the results of multiple-instruction sequences to reflect special-number inputs. For example, consider rcp(0). Input 0 to rcp, and you should get INF according to the DX10 spec. However, evaluating rcp via Newton-Raphson, where x=approx(1/0), yields an incorrect result. To deal with this, VFIXUPIMMPD can be used after the N-R reciprocal sequence to set the result to the correct value (i.e., INF when the input is 0).

If MXCSR.DAZ is not set, denormal input elements in the first source operand are considered as normal inputs and do not trigger any fixup nor fault reporting.

Imm8 is used to set the required flags reporting. It supports #ZE and #IE fault reporting (see details below).

MXCSR.DAZ is used and refer to zmm2 only (i.e., zmm1 is not considered as zero in case MXCSR.DAZ is set).

MXCSR mask bits are ignored and are treated as if all mask bits are set to masked response). If any of the imm8 bits is set and the condition met for fault reporting, MXCSR.IE or MXCSR.ZE might be updated.

Operation

```
enum TOKEN_TYPE
{
    QNAN_TOKEN := 0,
    SNAN_TOKEN := 1,
    ZERO_VALUE_TOKEN := 2,
    POS_ONE_VALUE_TOKEN := 3,
    NEG_INF_TOKEN := 4,
    POS_INF_TOKEN := 5,
    NEG_VALUE_TOKEN := 6,
    POS_VALUE_TOKEN := 7
}
```

```
FIXUPIMM_DP (dest[63:0], src1[63:0],tbl3[63:0], imm8 [7:0]){
    tsrc[63:0] := ((src1[62:52] = 0) AND (MXCSR.DAZ =1)) ? 0.0 : src1[63:0]
    CASE(tsrc[63:0] of TOKEN_TYPE) {
        QNAN_TOKEN: j := 0;
        SNAN_TOKEN: j := 1;
        ZERO_VALUE_TOKEN: j := 2;
        POS_ONE_VALUE_TOKEN: j := 3;
        NEG_INF_TOKEN: j := 4;
        POS_INF_TOKEN: j := 5;
        NEG_VALUE_TOKEN: j := 6;
        POS_VALUE_TOKEN: j := 7;
    }           ; end source special CASE(tsrc...)

    ; The required response from src3 table is extracted
    token_response[3:0] = tbl3[3+4*j:4*j];

    CASE(token_response[3:0]) {
        0000: dest[63:0] := dest[63:0]          ; preserve content of DEST
        0001: dest[63:0] := tsrc[63:0];         ; pass through src1 normal input value, denormal as zero
        0010: dest[63:0] := QNaN(tsrc[63:0]);
        0011: dest[63:0] := QNAN_Indefinite;
        0100:dest[63:0] := -INF;
        0101: dest[63:0] := +INF;
        0110: dest[63:0] := tsrc.sign? –INF : +INF;
        0111: dest[63:0] := -0;
        1000: dest[63:0] := +0;
        1001: dest[63:0] := -1;
        1010: dest[63:0] := +1;
        1011: dest[63:0] := ½;
        1100: dest[63:0] := 90.0;
        1101: dest[63:0] := PI/2;
        1110: dest[63:0] := MAX_FLOAT;
        1111: dest[63:0] := -MAX_FLOAT;
    }           ; end of token_response CASE

    ; The required fault reporting from imm8 is extracted
    ; TOKENs are mutually exclusive and TOKENs priority defines the order.
    ; Multiple faults related to a single token can occur simultaneously.
    IF (tsrc[63:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[0] then set #ZE;
    IF (tsrc[63:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[1] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[2] then set #ZE;
    IF (tsrc[63:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[3] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: SNAN_TOKEN) AND imm8[4] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: NEG_INF_TOKEN) AND imm8[5] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: NEG_VALUE_TOKEN) AND imm8[6] then set #IE;
    IF (tsrc[63:0] of TOKEN_TYPE: POS_INF_TOKEN) AND imm8[7] then set #IE;
        ; end fault reporting
    return dest[63:0];
}       ; end of FIXUPIMM_DP()
```

VFIXUPIMMSD (EVEX encoded version)

```
IF k1[0] OR *no writemask*
    THEN DEST[63:0] := FIXUPIMM_DP(DEST[63:0], SRC1[63:0], SRC2[63:0], imm8 [7:0])
    ELSE
        IF *merging-masking*                    ; merging-masking
            THEN *DEST[63:0] remains unchanged*
            ELSE  DEST[63:0] := 0               ; zeroing-masking
        FI
FI;
DEST[127:64] := SRC1[127:64]
DEST[MAXVL-1:128] := 0
```

Immediate Control Description:

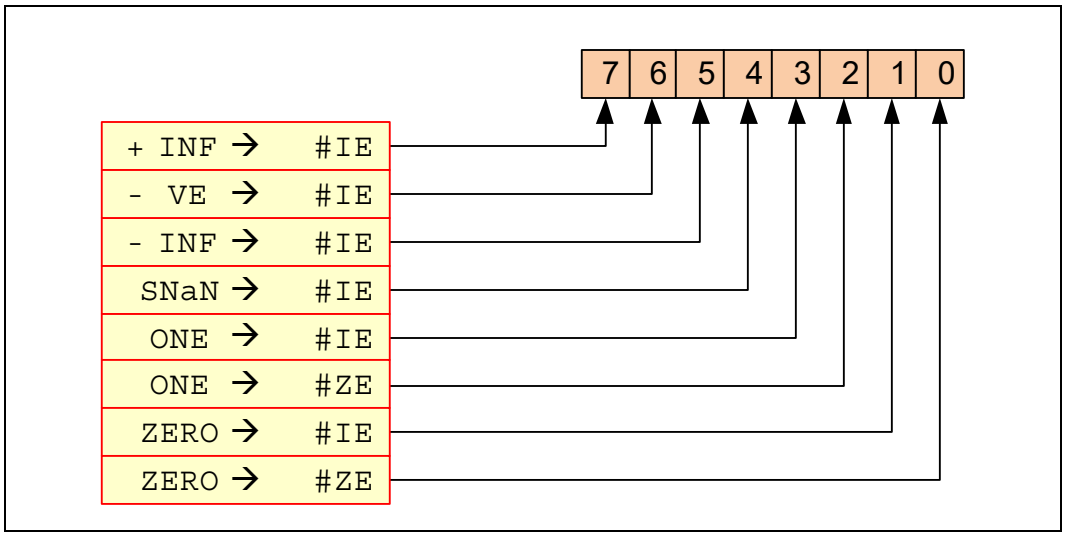


Figure 5-11. VFIXUPIMMSD Immediate Control Description

Intel C/C++ Compiler Intrinsic Equivalent

VFIXUPIMMSD __m128d _mm_fixupimm_sd(__m128d a, __m128d b, __m128i c, int imm8);
VFIXUPIMMSD __m128d _mm_mask_fixupimm_sd(__m128d a, __mmask8 k, __m128d b, __m128i c, int imm8);
VFIXUPIMMSD __m128d _mm_maskz_fixupimm_sd(__mmask8 k, __m128d a, __m128d b, __m128i c, int imm8);
VFIXUPIMMSD __m128d _mm_fixupimm_round_sd(__m128d a, __m128d b, __m128i c, int imm8, int sae);
VFIXUPIMMSD __m128d _mm_mask_fixupimm_round_sd(__m128d a, __mmask8 k, __m128d b, __m128i c, int imm8, int sae);
VFIXUPIMMSD __m128d _mm_maskz_fixupimm_round_sd(__mmask8 k, __m128d a, __m128d b, __m128i c, int imm8, int sae);

SIMD Floating-Point Exceptions

Zero, Invalid

Other Exceptions

See Table 2-47, "Type E3 Class Exception Conditions."

VFIXUPIMMSS—Fix Up Special Scalar Float32 Value

Opcode/ Instruction	Op / En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.LLIG.66.0F3A.W0 55 /r ib VFIXUPIMMSS xmm1 {k1}{z}, xmm2, xmm3/m32{sae}, imm8	A	V/V	AVX512F	Fix up a float32 number in the low doubleword element in xmm2 using scalar int32 table in xmm3/m32 and store the result in xmm1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Perform a fix-up of the low doubleword element encoded in single precision floating-point format in the first source operand (the second operand) using a 32-bit, two-level look-up table specified in the low doubleword element of the second source operand (the third operand) with exception reporting specifier imm8. The element that is fixed-up is selected by mask bit of 1 specified in the opmask k1. Mask bit of 0 in the opmask k1 or table response action of 0000b preserves the corresponding element of the first operand. The fixed-up element from the first source operand or the preserved element in the first operand becomes the low doubleword element of the destination operand (the first operand) Bits 127:32 of the destination operand is copied from the corresponding bits of the first source operand. The destination and first source operands are XMM registers. The second source operand can be a XMM register or a 32-bit memory location.

The two-level look-up table perform a fix-up of each single precision floating-point input data in the first source operand by decoding the input data encoding into 8 token types. A response table is defined for each token type that converts the input encoding in the first source operand with one of 16 response actions.

This instruction is specifically intended for use in fixing up the results of arithmetic calculations involving one source so that they match the spec, although it is generally useful for fixing up the results of multiple-instruction sequences to reflect special-number inputs. For example, consider rcp(0). Input 0 to rcp, and you should get INF according to the DX10 spec. However, evaluating rcp via Newton-Raphson, where x=approx(1/0), yields an incorrect result. To deal with this, VFIXUPIMMPD can be used after the N-R reciprocal sequence to set the result to the correct value (i.e., INF when the input is 0).

If MXCSR.DAZ is not set, denormal input elements in the first source operand are considered as normal inputs and do not trigger any fixup nor fault reporting.

Imm8 is used to set the required flags reporting. It supports #ZE and #IE fault reporting (see details below).

MXCSR.DAZ is used and refer to zmm2 only (i.e., zmm1 is not considered as zero in case MXCSR.DAZ is set).

MXCSR mask bits are ignored and are treated as if all mask bits are set to masked response). If any of the imm8 bits is set and the condition met for fault reporting, MXCSR.IE or MXCSR.ZE might be updated.

Operation

```
enum TOKEN_TYPE
{
    QNAN_TOKEN := 0,
    SNAN_TOKEN := 1,
    ZERO_VALUE_TOKEN := 2,
    POS_ONE_VALUE_TOKEN := 3,
    NEG_INF_TOKEN := 4,
    POS_INF_TOKEN := 5,
    NEG_VALUE_TOKEN := 6,
    POS_VALUE_TOKEN := 7
}
```

```
FIXUPIMM_SP (dest[31:0], src1[31:0],tbl3[31:0], imm8 [7:0]){
    tsrc[31:0] := ((src1[30:23] = 0) AND (MXCSR.DAZ =1)) ? 0.0 : src1[31:0]
    CASE(tsrc[63:0] of TOKEN_TYPE) {
        QNAN_TOKEN: j := 0;
        SNAN_TOKEN: j := 1;
        ZERO_VALUE_TOKEN: j := 2;
        POS_ONE_VALUE_TOKEN: j := 3;
        NEG_INF_TOKEN: j := 4;
        POS_INF_TOKEN: j := 5;
        NEG_VALUE_TOKEN: j := 6;
        POS_VALUE_TOKEN: j := 7;
    }           ; end source special CASE(tsrc...)

    ; The required response from src3 table is extracted
    token_response[3:0] = tbl3[3+4*j:4*j];

    CASE(token_response[3:0]) {
        0000: dest[31:0] := dest[31:0];         ; preserve content of DEST
        0001: dest[31:0] := tsrc[31:0];         ; pass through src1 normal input value, denormal as zero
        0010: dest[31:0] := QNaN(tsrc[31:0]);
        0011: dest[31:0] := QNAN_Indefinite;
        0100: dest[31:0] := -INF;
        0101: dest[31:0] := +INF;
        0110: dest[31:0] := tsrc.sign? –INF : +INF;
        0111: dest[31:0] := -0;
        1000: dest[31:0] := +0;
        1001: dest[31:0] := -1;
        1010: dest[31:0] := +1;
        1011: dest[31:0] := ½;
        1100: dest[31:0] := 90.0;
        1101: dest[31:0] := PI/2;
        1110: dest[31:0] := MAX_FLOAT;
        1111: dest[31:0] := -MAX_FLOAT;
    }           ; end of token_response CASE

    ; The required fault reporting from imm8 is extracted
    ; TOKENs are mutually exclusive and TOKENs priority defines the order.
    ; Multiple faults related to a single token can occur simultaneously.
    IF (tsrc[31:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[0] then set #ZE;
    IF (tsrc[31:0] of TOKEN_TYPE: ZERO_VALUE_TOKEN) AND imm8[1] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[2] then set #ZE;
    IF (tsrc[31:0] of TOKEN_TYPE: ONE_VALUE_TOKEN) AND imm8[3] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: SNAN_TOKEN) AND imm8[4] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: NEG_INF_TOKEN) AND imm8[5] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: NEG_VALUE_TOKEN) AND imm8[6] then set #IE;
    IF (tsrc[31:0] of TOKEN_TYPE: POS_INF_TOKEN) AND imm8[7] then set #IE;
        ; end fault reporting
    return dest[31:0];
}       ; end of FIXUPIMM_SP()
```

VFIXUPIMMSS (EVEX encoded version)

```
IF k1[0] OR *no writemask*
    THEN DEST[31:0] := FIXUPIMM_SP(DEST[31:0], SRC1[31:0], SRC2[31:0], imm8 [7:0])
    ELSE
        IF *merging-masking*                    ; merging-masking
            THEN *DEST[31:0] remains unchanged*
            ELSE  DEST[31:0] := 0               ; zeroing-masking
        FI
FI;
DEST[127:32] := SRC1[127:32]
DEST[MAXVL-1:128] := 0
```

Immediate Control Description:

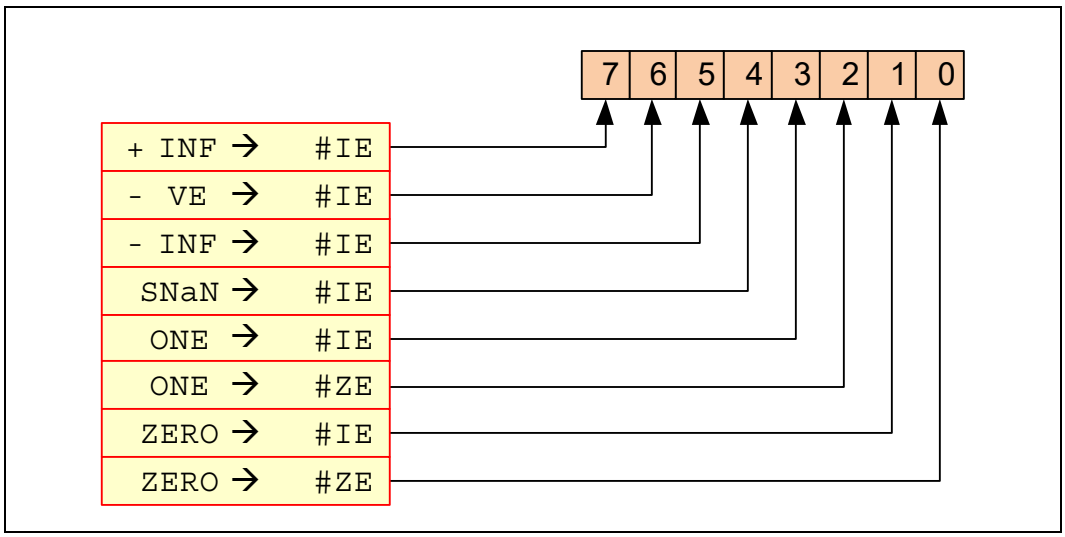


Figure 5-12. VFIXUPIMMSS Immediate Control Description

Intel C/C++ Compiler Intrinsic Equivalent

VFIXUPIMMSS __m128 _mm_fixupimm_ss(__m128 a, __m128 b, __m128i c, int imm8);
VFIXUPIMMSS __m128 _mm_mask_fixupimm_ss(__m128 a, __mmask8 k, __m128 b, __m128i c, int imm8);
VFIXUPIMMSS __m128 _mm_maskz_fixupimm_ss(__mmask8 k, __m128 a, __m128 b, __m128i c, int imm8);
VFIXUPIMMSS __m128 _mm_fixupimm_round_ss(__m128 a, __m128 b, __m128i c, int imm8, int sae);
VFIXUPIMMSS __m128 _mm_mask_fixupimm_round_ss(__m128 a, __mmask8 k, __m128 b, __m128i c, int imm8, int sae);
VFIXUPIMMSS __m128 _mm_maskz_fixupimm_round_ss(__mmask8 k, __m128 a, __m128 b, __m128i c, int imm8, int sae);

SIMD Floating-Point Exceptions

Zero, Invalid

Other Exceptions

See Table 2-47, "Type E3 Class Exception Conditions."

9. Updates to Chapter 11, Volume 3A

Change bars and violet text show changes to Chapter 11 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 11.5.4.1, "TSC-Deadline Mode," to clarify that race conditions may result in the delivery of a timer interrupt associated with the original deadline. The previous wording was not clear regarding this.

CHAPTER 11
ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (APIC)

The Advanced Programmable Interrupt Controller (APIC), referred to in the following sections as the local APIC, was introduced into the IA-32 processors with the Pentium processor (see Section 23.27, "Advanced Programmable Interrupt Controller (APIC)") and is included in the P6 family, Pentium 4, Intel Xeon processors, and other more recent Intel 64 and IA-32 processor families (see Section 11.4.2, "Presence of the Local APIC"). The local APIC performs two primary functions for the processor:

- It receives interrupts from the processor's interrupt pins, from internal sources and from an external I/O APIC (or other external interrupt controller). It sends these to the processor core for handling.
- In multiple processor (MP) systems, it sends and receives interprocessor interrupt (IPI) messages to and from other logical processors on the system bus. IPI messages can be used to distribute interrupts among the processors in the system or to execute system wide functions (such as, booting up processors or distributing work among a group of processors).

The external **I/O APIC** is part of Intel's system chipset. Its primary function is to receive external interrupt events from the system and its associated I/O devices and relay them to the local APIC as interrupt messages. In MP systems, the I/O APIC also provides a mechanism for distributing external interrupts to the local APICs of selected processors or groups of processors on the system bus.

This chapter provides a description of the local APIC and its programming interface. It also provides an overview of the interface between the local APIC and the I/O APIC. Contact Intel for detailed information about the I/O APIC.

When a local APIC has sent an interrupt to its processor core for handling, the processor uses the interrupt and exception handling mechanism described in Chapter 6, "Interrupt and Exception Handling." See Section 6.1, "Interrupt and Exception Overview," for an introduction to interrupt and exception handling.

11.1 LOCAL AND I/O APIC OVERVIEW

Each local APIC consists of a set of APIC registers (see Table 11-1) and associated hardware that control the delivery of interrupts to the processor core and the generation of IPI messages. The APIC registers are memory mapped and can be read and written to using the MOV instruction.

Local APICs can receive interrupts from the following sources:

- **Locally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected directly to the processor's local interrupt pins (LINT0 and LINT1). The I/O devices may also be connected to an 8259-type interrupt controller that is in turn connected to the processor through one of the local interrupt pins.
- **Externally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected to the interrupt input pins of an I/O APIC. Interrupts are sent as I/O interrupt messages from the I/O APIC to one or more of the processors in the system.
- **Inter-processor interrupts (IPIs)** — An Intel 64 or IA-32 processor can use the IPI mechanism to interrupt another processor or group of processors on the system bus. IPIs are used for software self-interrupts, interrupt forwarding, or preemptive scheduling.
- **APIC timer generated interrupts** — The local APIC timer can be programmed to send a local interrupt to its associated processor when a programmed count is reached (see Section 11.5.4, "APIC Timer").
- **Performance monitoring counter interrupts** — P6 family, Pentium 4, and Intel Xeon processors provide the ability to send an interrupt to its associated processor when a performance-monitoring counter overflows (see Section 20.6.3.5.8, "Generating an Interrupt on Overflow").
- **Thermal Sensor interrupts** — Pentium 4 and Intel Xeon processors provide the ability to send an interrupt to themselves when the internal thermal sensor has been tripped (see Section 15.8.2, "Thermal Monitor").
- **APIC internal error interrupts** — When an error condition is recognized within the local APIC (such as an attempt to access an unimplemented register), the APIC can be programmed to send an interrupt to its associated processor (see Section 11.5.3, "Error Handling").

Of these interrupt sources: the processor's LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, the thermal sensor, and the internal APIC error detector are referred to as **local interrupt sources**. Upon receiving a signal from a local interrupt source, the local APIC delivers the interrupt to the processor core using an interrupt delivery protocol that has been set up through a group of APIC registers called the **local vector table** or **LVT** (see Section 11.5.1, "Local Vector Table"). A separate entry is provided in the local vector table for each local interrupt source, which allows a specific interrupt delivery protocol to be set up for each source. For example, if the LINT1 pin is going to be used as an NMI pin, the LINT1 entry in the local vector table can be set up to deliver an interrupt with vector number 2 (NMI interrupt) to the processor core.

The local APIC handles interrupts from the other two interrupt sources (externally connected I/O devices and IPIs) through its IPI message handling facilities.

A processor can generate IPIs by programming the interrupt command register (ICR) in its local APIC (see Section 11.6.1, "Interrupt Command Register (ICR)"). The act of writing to the ICR causes an IPI message to be generated and issued on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for Pentium and P6 family processors). See Section 11.2, "System Bus Vs. APIC Bus."

IPIs can be sent to other processors in the system or to the originating processor (self-interrupts). When the target processor receives an IPI message, its local APIC handles the message automatically (using information included in the message such as vector number and trigger mode). See Section 11.6, "Issuing Interprocessor Interrupts," for a detailed explanation of the local APIC's IPI message delivery and acceptance mechanism.

The local APIC can also receive interrupts from externally connected devices through the I/O APIC (see Figure 11-1). The I/O APIC is responsible for receiving interrupts generated by system hardware and I/O devices and forwarding them to the local APIC as interrupt messages.

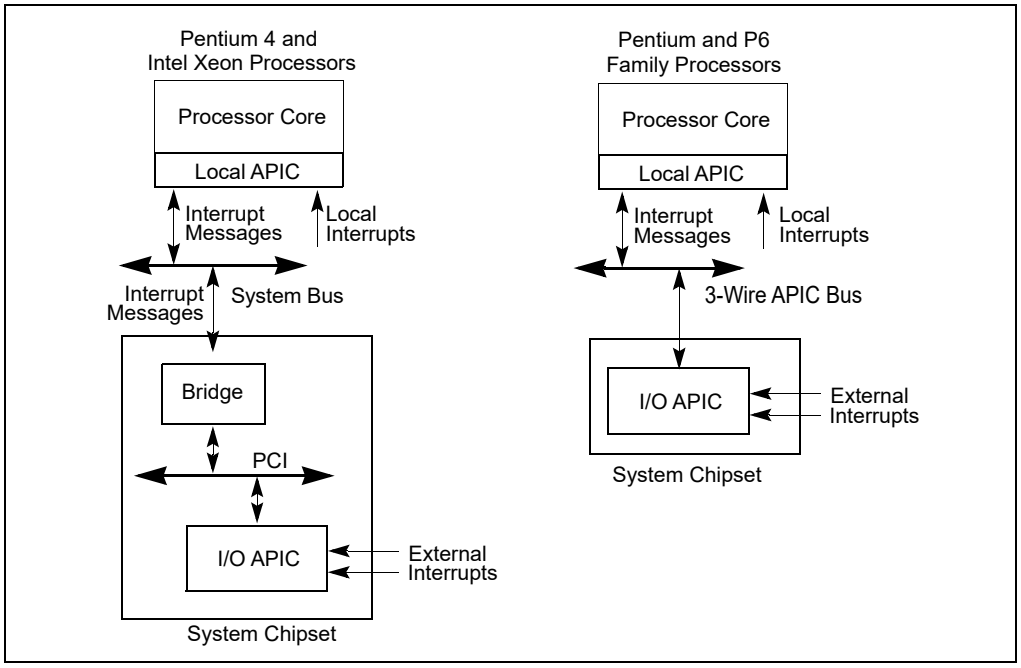


Figure 11-1. Relationship of Local APIC and I/O APIC In Single-Processor Systems

Individual pins on the I/O APIC can be programmed to generate a specific interrupt vector when asserted. The I/O APIC also has a "virtual wire mode" that allows it to communicate with a standard 8259A-style external interrupt controller. Note that the local APIC can be disabled (see Section 11.4.3, "Enabling or Disabling the Local APIC"). This allows an associated processor core to receive interrupts directly from an 8259A interrupt controller.

Both the local APIC and the I/O APIC are designed to operate in MP systems (see Figures 11-2 and 11-3). Each local APIC handles interrupts from the I/O APIC, IPIs from processors on the system bus, and self-generated interrupts. Interrupts can also be delivered to the individual processors through the local interrupt pins; however, this mechanism is commonly not used in MP systems.

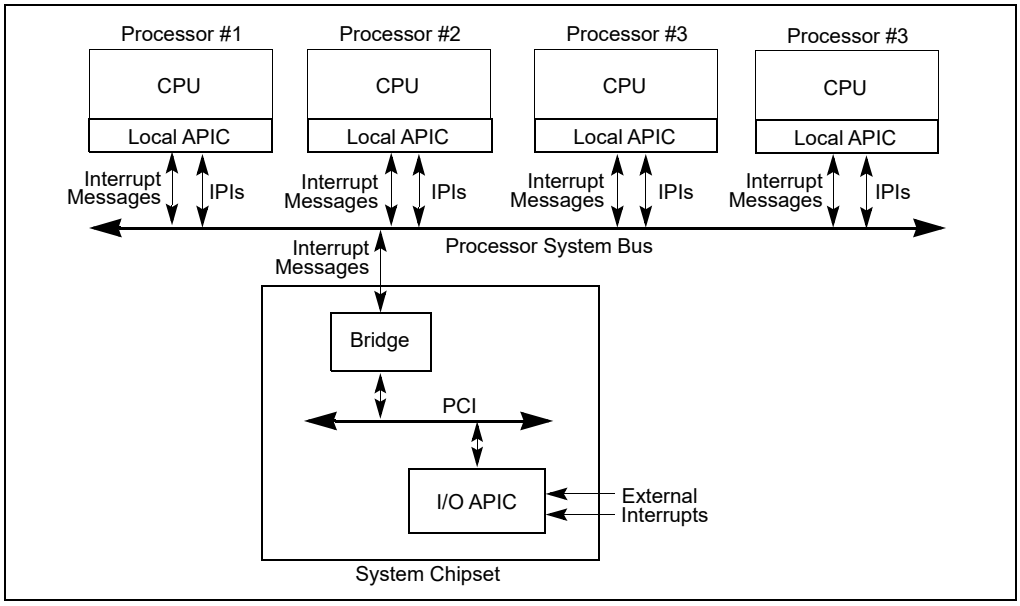


Figure 11-2. Local APICs and I/O APIC When Intel Xeon Processors Are Used in Multiple-Processor Systems

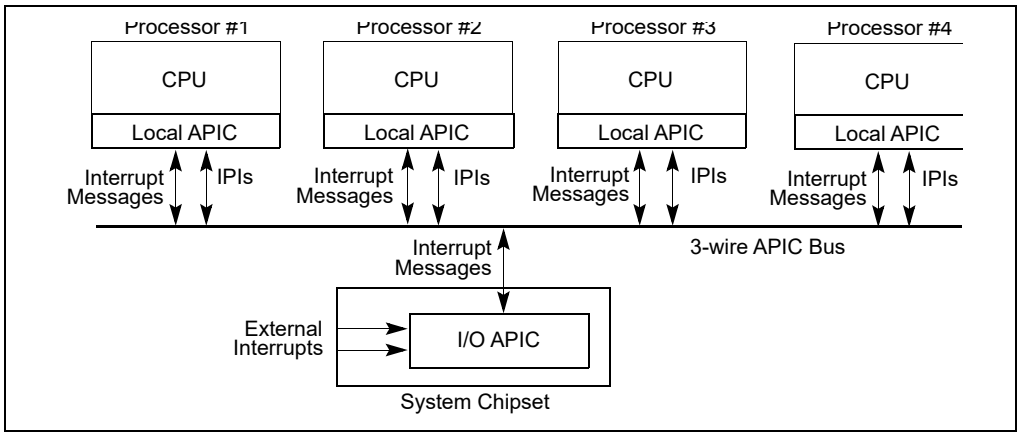


Figure 11-3. Local APICs and I/O APIC When P6 Family Processors Are Used in Multiple-Processor Systems

The IPI mechanism is typically used in MP systems to send fixed interrupts (interrupts for a specific vector number) and special-purpose interrupts to processors on the system bus. For example, a local APIC can use an IPI to forward a fixed interrupt to another processor for servicing. Special-purpose IPIs (including NMI, INIT, SMI, and SIPI IPIs) allow one or more processors on the system bus to perform system-wide boot-up and control functions.

The following sections focus on the local APIC and its implementation in the Pentium 4, Intel Xeon, and P6 family processors. In these sections, the terms "local APIC" and "I/O APIC" refer to local and I/O APICs used with the P6 family processors and to local and I/O xAPICs used with the Pentium 4 and Intel Xeon processors (see Section 11.3, "The Intel® 82489DX External APIC, the APIC, the xAPIC, and the X2APIC").

11.2 SYSTEM BUS VS. APIC BUS

For the P6 family and Pentium processors, the I/O APIC and local APICs communicate through the 3-wire inter-APIC bus (see Figure 11-3). Local APICs also use the APIC bus to send and receive IPIs. The APIC bus and its messages are invisible to software and are not classed as architectural.

Beginning with the Pentium 4 and Intel Xeon processors, the I/O APIC and local APICs (using the xAPIC architecture) communicate through the system bus (see Figure 11-2). The I/O APIC sends interrupt requests to the processors on the system bus through bridge hardware that is part of the Intel chipset. The bridge hardware generates the interrupt messages that go to the local APICs. IPIs between local APICs are transmitted directly on the system bus.

11.3 THE INTEL® 82489DX EXTERNAL APIC, THE APIC, THE XAPIC, AND THE X2APIC

The local APIC in the P6 family and Pentium processors is an architectural subset of the Intel® 82489DX external APIC. See Section 23.27.1, "Software Visible Differences Between the Local APIC and the 82489DX."

The APIC architecture used in the Pentium 4 and Intel Xeon processors (called the xAPIC architecture) is an extension of the APIC architecture found in the P6 family processors. The primary difference between the APIC and xAPIC architectures is that with the xAPIC architecture, the local APICs and the I/O APIC communicate through the system bus. With the APIC architecture, they communication through the APIC bus (see Section 11.2, "System Bus Vs. APIC Bus"). Also, some APIC architectural features have been extended and/or modified in the xAPIC architecture. These extensions and modifications are described in Section 11.4 through Section 11.10.

The basic operating mode of the xAPIC is **xAPIC mode**. The x2APIC architecture is an extension of the xAPIC architecture, primarily to increase processor addressability. The x2APIC architecture provides backward compatibility to the xAPIC architecture and forward extendability for future Intel platform innovations. These extensions and modifications are supported by a new mode of execution (**x2APIC mode**) are detailed in Section 11.12.

11.4 LOCAL APIC

The following sections describe the architecture of the local APIC and how to detect it, identify it, and determine its status. Descriptions of how to program the local APIC are given in Section 11.5.1, "Local Vector Table," and Section 11.6.1, "Interrupt Command Register (ICR)."

11.4.1 The Local APIC Block Diagram

Figure 11-4 gives a functional block diagram for the local APIC. Software interacts with the local APIC by reading and writing its registers. APIC registers are memory-mapped to a 4-KByte region of the processor's physical address space with an initial starting address of FEE00000H. For correct APIC operation, this address space must be mapped to an area of memory that has been designated as strong uncacheable (UC). See Section 12.3, "Methods of Caching Available."

In MP system configurations, the APIC registers for Intel 64 or IA-32 processors on the system bus are initially mapped to the same 4-KByte region of the physical address space. Software has the option of changing initial mapping to a different 4-KByte region for all the local APICs or of mapping the APIC registers for each local APIC to its own 4-KByte region. Section 11.4.5, "Relocating the Local APIC Registers," describes how to relocate the base address for APIC registers.

On processors supporting x2APIC architecture (indicated by CPUID.01H:ECX[21] = 1), the local APIC supports operation both in xAPIC mode and (if enabled by software) in x2APIC mode. x2APIC mode provides extended processor addressability (see Section 11.12).

NOTE

For P6 family, Pentium 4, and Intel Xeon processors, the APIC handles all memory accesses to addresses within the 4-KByte APIC register space internally and no external bus cycles are produced. For the Pentium processors with an on-chip APIC, bus cycles are produced for accesses to the APIC register space. Thus, for software intended to run on Pentium processors, system software should explicitly not map the APIC register space to regular system memory. Doing so can result in an invalid opcode exception (#UD) being generated or unpredictable execution.

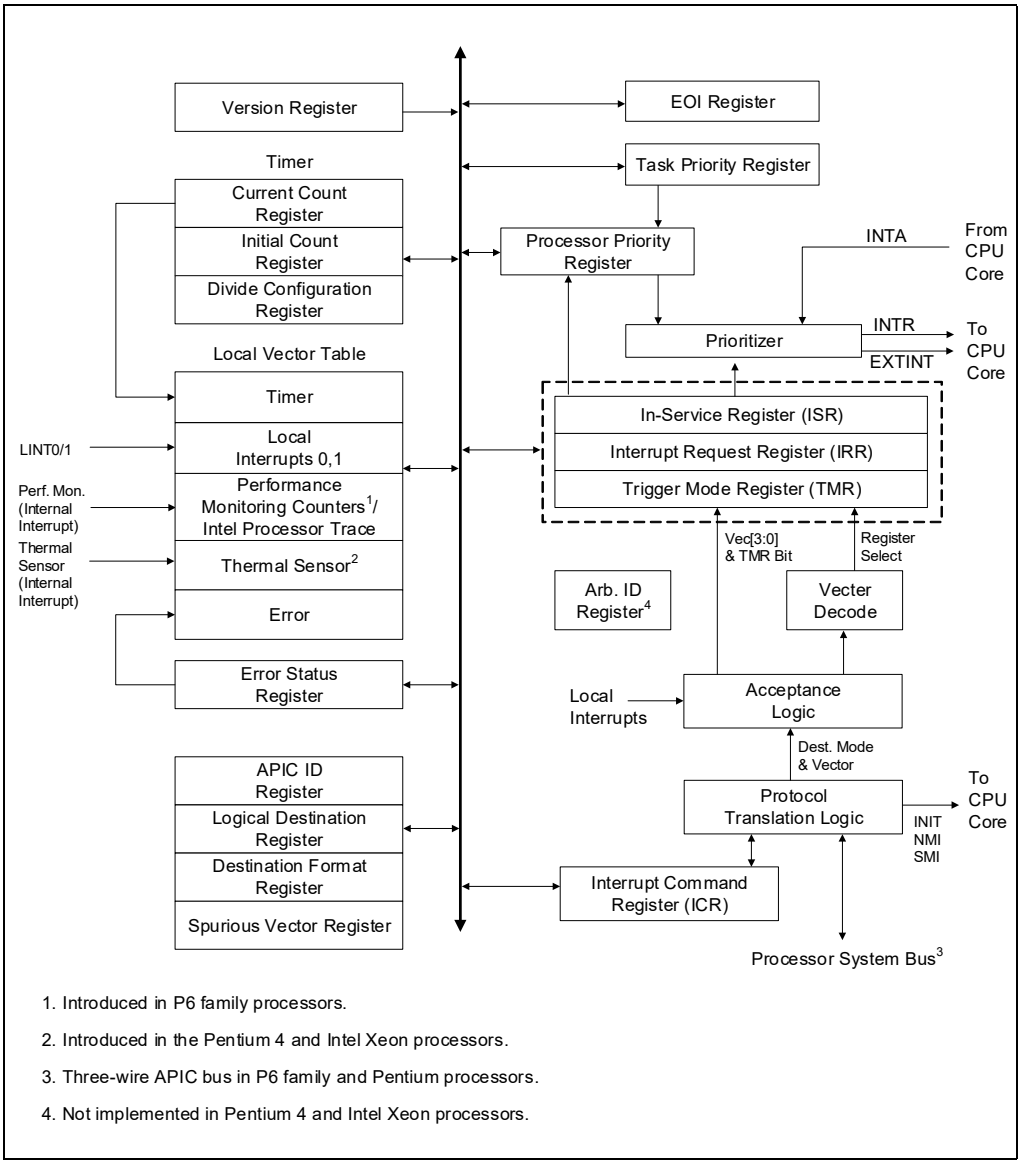


Figure 11-4. Local APIC Structure

Table 11-1 shows how the APIC registers are mapped into the 4-KByte APIC register space. Registers are 32 bits, 64 bits, or 256 bits in width; all are aligned on 128-bit boundaries. All 32-bit registers should be accessed using 128-bit aligned 32-bit loads or stores. Some processors may support loads and stores of less than 32 bits to some of the APIC registers. This is model specific behavior and is not guaranteed to work on all processors. Any FP/MMX/SSE access to an APIC register, or any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results or unexpected exceptions, including machine checks, and may vary between implementations. Wider registers (64-bit or 256-bit) must be accessed using multiple 32-bit loads or stores, with all accesses being 128-bit aligned.

The local APIC registers listed in Table 11-1 are not MSRs. The only MSR associated with the programming of the local APIC is the IA32_APIC_BASE MSR (see Section 11.4.3, “Enabling or Disabling the Local APIC”).

NOTE

In processors based on Nehalem[1] microarchitecture, the Local APIC ID Register is no longer Read/Write; it is Read Only.

Table 11-1. Local APIC Register Address Map

Address	Register Name	Software Read/Write
FEE0 0000H	Reserved	
FEE0 0010H	Reserved	
FEE0 0020H	Local APIC ID Register	Read/Write.
FEE0 0030H	Local APIC Version Register	Read Only.
FEE0 0040H	Reserved	
FEE0 0050H	Reserved	
FEE0 0060H	Reserved	
FEE0 0070H	Reserved	
FEE0 0080H	Task Priority Register (TPR)	Read/Write.
FEE0 0090H	Arbitration Priority Register[1] (APR)	Read Only.
FEE0 00A0H	Processor Priority Register (PPR)	Read Only.
FEE0 00B0H	EOI Register	Write Only.
FEE0 00C0H	Remote Read Register[1] (RRD)	Read Only
FEE0 00D0H	Logical Destination Register	Read/Write.
FEE0 00E0H	Destination Format Register	Read/Write (see Section 11.6.2.2).
FEE0 00F0H	Spurious Interrupt Vector Register	Read/Write (see Section 11.9.
FEE0 0100H	In-Service Register (ISR); bits 31:0	Read Only.
FEE0 0110H	In-Service Register (ISR); bits 63:32	Read Only.
FEE0 0120H	In-Service Register (ISR); bits 95:64	Read Only.
FEE0 0130H	In-Service Register (ISR); bits 127:96	Read Only.
FEE0 0140H	In-Service Register (ISR); bits 159:128	Read Only.
FEE0 0150H	In-Service Register (ISR); bits 191:160	Read Only.
FEE0 0160H	In-Service Register (ISR); bits 223:192	Read Only.
FEE0 0170H	In-Service Register (ISR); bits 255:224	Read Only.
FEE0 0180H	Trigger Mode Register (TMR); bits 31:0	Read Only.
FEE0 0190H	Trigger Mode Register (TMR); bits 63:32	Read Only.
FEE0 01A0H	Trigger Mode Register (TMR); bits 95:64	Read Only.
FEE0 01B0H	Trigger Mode Register (TMR); bits 127:96	Read Only.
FEE0 01C0H	Trigger Mode Register (TMR); bits 159:128	Read Only.
FEE0 01D0H	Trigger Mode Register (TMR); bits 191:160	Read Only.
FEE0 01E0H	Trigger Mode Register (TMR); bits 223:192	Read Only.
FEE0 01F0H	Trigger Mode Register (TMR); bits 255:224	Read Only.
FEE0 0200H	Interrupt Request Register (IRR); bits 31:0	Read Only.
FEE0 0210H	Interrupt Request Register (IRR); bits 63:32	Read Only.
FEE0 0220H	Interrupt Request Register (IRR); bits 95:64	Read Only.

1. See Table 2-1, "CPUID Signature Values of DisplayFamily_DisplayModel," on page 1, and Section 2.8, "MSRs In Processors Based on Nehalem Microarchitecture," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, to determine which processors are based on Nehalem microarchitecture.

Table 11-1. Local APIC Register Address Map (Contd.)

Address	Register Name	Software Read/Write
FEE0 0230H	Interrupt Request Register (IRR); bits 127:96	Read Only.
FEE0 0240H	Interrupt Request Register (IRR); bits 159:128	Read Only.
FEE0 0250H	Interrupt Request Register (IRR); bits 191:160	Read Only.
FEE0 0260H	Interrupt Request Register (IRR); bits 223:192	Read Only.
FEE0 0270H	Interrupt Request Register (IRR); bits 255:224	Read Only.
FEE0 0280H	Error Status Register	Write/Read; see Section 11.5.3.
FEE0 0290H through FEE0 02E0H	Reserved	
FEE0 02F0H	LVT Corrected Machine Check Interrupt (CMCI) Register	Read/Write.
FEE0 0300H	Interrupt Command Register (ICR); bits 0-31	Read/Write.
FEE0 0310H	Interrupt Command Register (ICR); bits 32-63	Read/Write.
FEE0 0320H	LVT Timer Register	Read/Write.
FEE0 0330H	LVT Thermal Sensor Register[2]	Read/Write.
FEE0 0340H	LVT Performance Monitoring Counters Register[3]	Read/Write.
FEE0 0350H	LVT LINT0 Register	Read/Write.
FEE0 0360H	LVT LINT1 Register	Read/Write.
FEE0 0370H	LVT Error Register	Read/Write.
FEE0 0380H	Initial Count Register (for Timer)	Read/Write.
FEE0 0390H	Current Count Register (for Timer)	Read Only.
FEE0 03A0H through FEE0 03D0H	Reserved	
FEE0 03E0H	Divide Configuration Register (for Timer)	Read/Write.
FEE0 03F0H	Reserved	

NOTES:

1. Not supported in the Pentium 4 and Intel Xeon processors. The Illegal Register Access bit (7) of the ESR will not be set when writing to these registers.
2. Introduced in the Pentium 4 and Intel Xeon processors. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.
3. Introduced in the Pentium Pro processor. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.

11.4.2 Presence of the Local APIC

Beginning with the P6 family processors, the presence or absence of an on-chip local APIC can be detected using the CPUID instruction. When the CPUID instruction is executed with a source operand of 1 in the EAX register, bit 9 of the CPUID feature flags returned in the EDX register indicates the presence (set) or absence (clear) of a local APIC.

11.4.3 Enabling or Disabling the Local APIC

The local APIC can be enabled or disabled in either of two ways:

1. Using the APIC global enable/disable flag in the IA32_APIC_BASE MSR (MSR address 1BH; see Figure 11-5):

— When IA32_APIC_BASE[11] is 0, the processor is functionally equivalent to an IA-32 processor without an on-chip APIC. The CPUID feature flag for the APIC (see Section 11.4.2, "Presence of the Local APIC") is also set to 0.

— When IA32_APIC_BASE[11] is set to 0, processor APICs based on the 3-wire APIC bus cannot be generally re-enabled until a system hardware reset. The 3-wire bus loses track of arbitration that would be necessary for complete re-enabling. Certain APIC functionality can be enabled (for example: performance and thermal monitoring interrupt generation).

— For processors that use Front Side Bus (FSB) delivery of interrupts, software may disable or enable the APIC by setting and resetting IA32_APIC_BASE[11]. A hardware reset is not required to re-start APIC functionality, if software guarantees no interrupt will be sent to the APIC as IA32_APIC_BASE[11] is cleared.

— When IA32_APIC_BASE[11] is set to 0, prior initialization to the APIC may be lost and the APIC may return to the state described in Section 11.4.7.1, "Local APIC State After Power-Up or Reset."

2. Using the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 11-23):

— If IA32_APIC_BASE[11] is 1, software can temporarily disable a local APIC at any time by clearing the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 11-23). The state of the local APIC when in this software-disabled state is described in Section 11.4.7.2, "Local APIC State After It Has Been Software Disabled."

— When the local APIC is in the software-disabled state, it can be re-enabled at any time by setting the APIC software enable/disable flag to 1.

For the Pentium processor, the APICEN pin (which is shared with the PICD1 pin) is used during power-up or reset to disable the local APIC.

Note that each entry in the LVT has a mask bit that can be used to inhibit interrupts from being delivered to the processor from selected local interrupt sources (the LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, Intel® Processor Trace, the thermal sensor, and/or the internal APIC error detector).

11.4.4 Local APIC Status and Location

The status and location of the local APIC are contained in the IA32_APIC_BASE MSR (see Figure 11-5). MSR bit functions are described below:

- **BSP flag, bit 8** — Indicates if the processor is the bootstrap processor (BSP). See Section 9.4, "Multiple-Processor (MP) Initialization." Following a power-up or reset, this flag is set to 1 for the processor selected as the BSP and set to 0 for the remaining processors (APs).
- **APIC Global Enable flag, bit 11** — Enables or disables the local APIC (see Section 11.4.3, "Enabling or Disabling the Local APIC"). This flag is available in the Pentium 4, Intel Xeon, and P6 family processors. It is not guaranteed to be available or available at the same location in future Intel 64 or IA-32 processors.
- **APIC Base field, bits 12 through 35** — Specifies the base address of the APIC registers. This 24-bit value is extended by 12 bits at the low end to form the base address. This automatically aligns the address on a 4-KByte boundary. Following a power-up or reset, the field is set to FEE0 0000H.
- Bits 0 through 7, bits 9 and 10, and bits MAXPHYADDR[1] through 63 in the IA32_APIC_BASE MSR are reserved.

1. The MAXPHYADDR is 36 bits for processors that do not support CPUID leaf 80000008H, or indicated by CPUID.80000008H:EAX[bits 7:0] for processors that support CPUID leaf 80000008H.

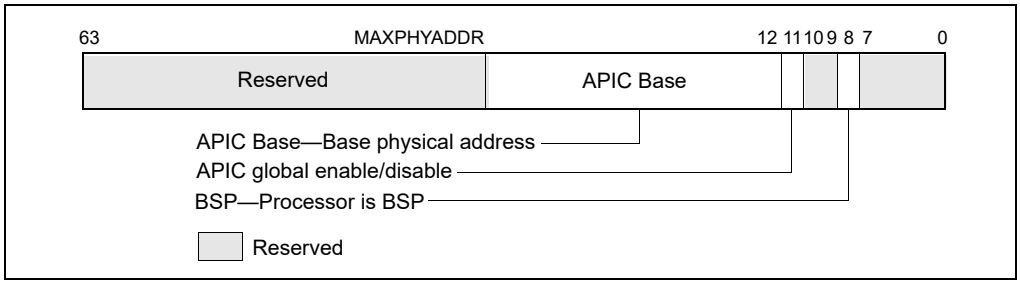


Figure 11-5. IA32_APIC_BASE MSR (APIC_BASE_MSR in P6 Family)

11.4.5 Relocating the Local APIC Registers

The Pentium 4, Intel Xeon, and P6 family processors permit the starting address of the APIC registers to be relocated from FEE00000H to another physical address by modifying the value in the base address field of the IA32_APIC_BASE MSR. This extension of the APIC architecture is provided to help resolve conflicts with memory maps of existing systems and to allow individual processors in an MP system to map their APIC registers to different locations in physical memory.

11.4.6 Local APIC ID

At power up, system hardware assigns a unique APIC ID to each local APIC on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for P6 family and Pentium processors). The hardware assigned APIC ID is based on system topology and includes encoding for socket position and cluster information (see Figure 9-2 and Section 9.9.1, "Hierarchical Mapping of Shared Resources").

In MP systems, the local APIC ID is also used as a processor ID by the BIOS and the operating system. Some processors permit software to modify the APIC ID. However, the ability of software to modify the APIC ID is processor model specific. Because of this, operating system software should avoid writing to the local APIC ID register. The value returned by bits 31-24 of the EBX register (when the CPUID instruction is executed with a source operand value of 1 in the EAX register) is always the Initial APIC ID (determined by the platform initialization). This is true even if software has changed the value in the Local APIC ID register.

The processor receives the hardware assigned APIC ID (or Initial APIC ID) by sampling pins A11# and A12# and pins BR0# through BR3# (for the Pentium 4, Intel Xeon, and P6 family processors) and pins BE0# through BE3# (for the Pentium processor). The APIC ID latched from these pins is stored in the APIC ID field of the local APIC ID register (see Figure 11-6), and is used as the Initial APIC ID for the processor.

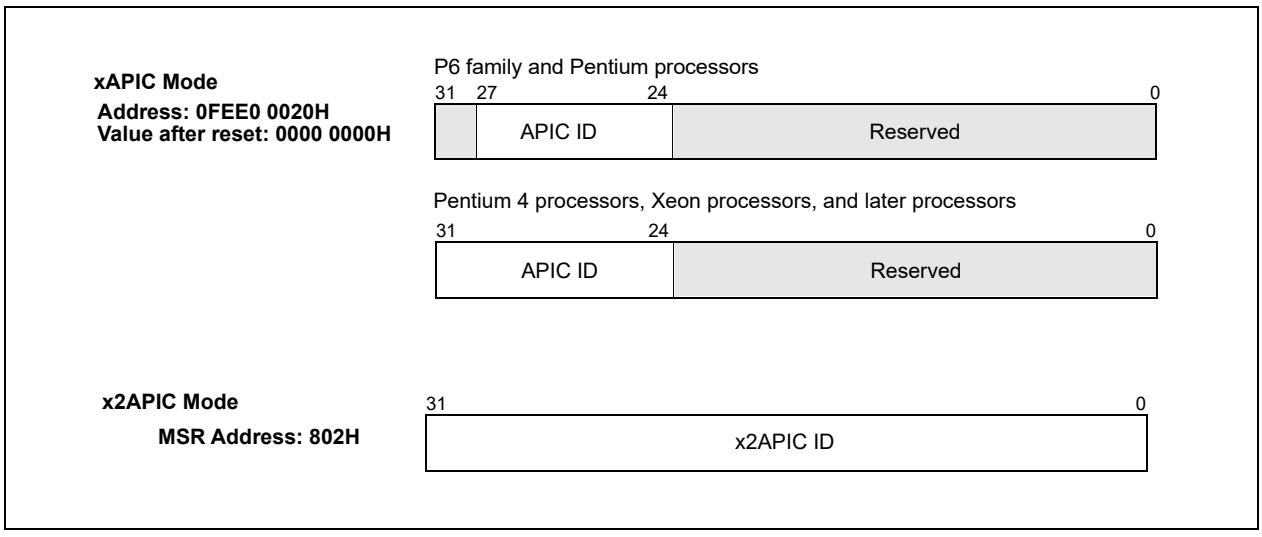


Figure 11-6. Local APIC ID Register

For the P6 family and Pentium processors, the local APIC ID field in the local APIC ID register is 4 bits. Encodings 0H through EH can be used to uniquely identify 15 different processors connected to the APIC bus. For the Pentium 4 and Intel Xeon processors, the xAPIC specification extends the local APIC ID field to 8 bits. These can be used to identify up to 255 processors in the system.

11.4.7 Local APIC State

The following sections describe the state of the local APIC and its registers following a power-up or reset, after the local APIC has been software disabled, following an INIT reset, and following an INIT-deassert message.

x2APIC will introduce 32-bit ID; see Section 11.12.

11.4.7.1 Local APIC State After Power-Up or Reset

Following a power-up or reset of the processor, the state of local APIC and its registers are as follows:

- The following registers are reset to all 0s.
    - IRR, ISR, TMR, ICR, LDR, and TPR.
    - Timer initial count and timer current count registers.
    - Divide configuration register.
- The DFR register is reset to all 1s.
- The LVT register is reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The local APIC ID register is set to a unique APIC ID. (Pentium and P6 family processors only). The Arb ID register is set to the value in the APIC ID register.
- The spurious-interrupt vector register is initialized to 000000FFH. By setting bit 8 to 0, software disables the local APIC.
- If the processor is the only processor in the system or it is the BSP in an MP system (see Section 9.4.1, "BSP and AP Processors"); the local APIC will respond normally to INIT and NMI messages, to INIT# signals and to STPCLK# signals. If the processor is in an MP system and has been designated as an AP; the local APIC will respond the same as for the BSP. In addition, it will respond to SIPI messages. For P6 family processors only, an AP will not respond to a STPCLK# signal.

11.4.7.2 Local APIC State After It Has Been Software Disabled

When the APIC software enable/disable flag in the spurious interrupt vector register has been explicitly cleared (as opposed to being cleared during a power up or reset), the local APIC is temporarily disabled (see Section 11.4.3, "Enabling or Disabling the Local APIC"). The operation and response of a local APIC while in this software-disabled state is as follows:

- The local APIC will respond normally to INIT, NMI, SMI, and SIPI messages.
- Pending interrupts in the IRR and ISR registers are held and require masking or handling by the CPU.
- The local APIC can still issue IPIs. It is software's responsibility to avoid issuing IPIs through the IPI mechanism and the ICR register if sending interrupts through this mechanism is not desired.
- The reception of any interrupt or transmission of any IPIs that are in progress when the local APIC is disabled are completed before the local APIC enters the software-disabled state.
- The mask bits for all the LVT entries are set. Attempts to reset these bits will be ignored.
- (For Pentium and P6 family processors) The local APIC continues to listen to all bus messages in order to keep its arbitration ID synchronized with the rest of the system.

11.4.7.3 Local APIC State After an INIT Reset ("Wait-for-SIPI" State)

An INIT reset of the processor can be initiated in either of two ways:

- By asserting the processor's INIT# pin.
- By sending the processor an INIT IPI (an IPI with the delivery mode set to INIT).

Upon receiving an INIT through either of these mechanisms, the processor responds by beginning the initialization process of the processor core and the local APIC. The state of the local APIC following an INIT reset is the same as it is after a power-up or hardware reset, except that the APIC ID and arbitration ID registers are not affected. This state is also referred to at the "wait-for-SIPI" state (see also: Section 9.4.2, "MP Initialization Protocol Requirements and Restrictions").

11.4.7.4 Local APIC State After It Receives an INIT-Deassert IPI

Only the Pentium and P6 family processors support the INIT-deassert IPI. An INIT-deassert IPI has no affect on the state of the APIC, other than to reload the arbitration ID register with the value in the APIC ID register.

11.4.8 Local APIC Version Register

The local APIC contains a hardwired version register. Software can use this register to identify the APIC version (see Figure 11-7). In addition, the register specifies the number of entries in the local vector table (LVT) for a specific implementation.

The fields in the local APIC version register are as follows:

Version The version numbers of the local APIC:

- 0XH — 82489DX discrete APIC.
- 10H - 15H — Integrated APIC.
- Other values reserved.

Max LVT Entry Shows the number of LVT entries minus 1. For the Pentium 4 and Intel Xeon processors (which have 6 LVT entries), the value returned in the Max LVT field is 5; for the P6 family processors (which have 5 LVT entries), the value returned is 4; for the Pentium processor (which has 4 LVT entries), the value returned is 3. For processors based on the Nehalem microarchitecture (which has 7 LVT entries) and onward, the value returned is 6.

Suppress EOI-broadcasts Indicates whether software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register; see Section 11.8.5 and Section 11.9.

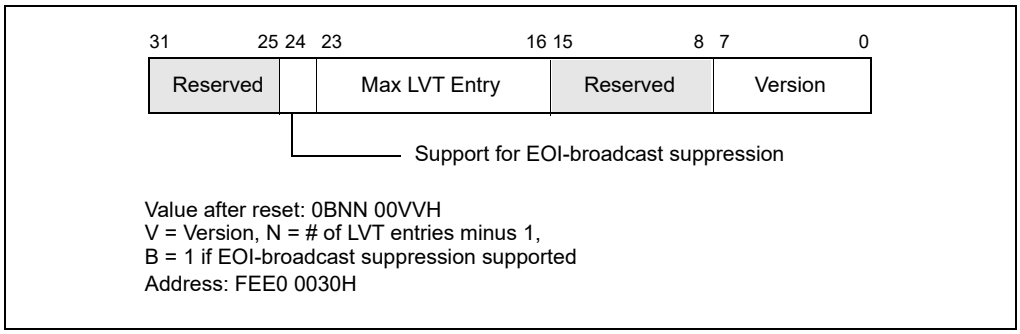


Figure 11-7. Local APIC Version Register

11.5 HANDLING LOCAL INTERRUPTS

The following sections describe facilities that are provided in the local APIC for handling local interrupts. These include: the processor's LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, Intel Processor Trace, the thermal sensor, and the internal APIC error detector. Local interrupt handling facilities include: the LVT, the error status register (ESR), the divide configuration register (DCR), and the initial count and current count registers.

11.5.1 Local Vector Table

The local vector table (LVT) allows software to specify the manner in which the local interrupts are delivered to the processor core. It consists of the following 32-bit APIC registers (see Figure 11-8), one for each local interrupt:

- **LVT CMCI Register (FEE0 02F0H)** — Specifies interrupt delivery when an overflow condition of corrected machine check error count reaching a threshold value occurred in a machine check bank supporting CMCI (see Section 16.5.1, "CMCI Local APIC Interface").
- **LVT Timer Register (FEE0 0320H)** — Specifies interrupt delivery when the APIC timer signals an interrupt (see Section 11.5.4, "APIC Timer").
- **LVT Thermal Monitor Register (FEE0 0330H)** — Specifies interrupt delivery when the thermal sensor generates an interrupt (see Section 15.8.2, "Thermal Monitor"). This LVT entry is implementation specific, not architectural. If implemented, it will always be at base address FEE0 0330H.
- **LVT Performance Counter Register (FEE0 0340H)** — Specifies interrupt delivery when a performance counter generates an interrupt on overflow (see Section 20.6.3.5.8, "Generating an Interrupt on Overflow") or when Intel PT signals a ToPA PMI (see Section 33.2.7.2). This LVT entry is implementation specific, not architectural. If implemented, it is not guaranteed to be at base address FEE0 0340H.
- **LVT LINT0 Register (FEE0 0350H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT0 pin.
- **LVT LINT1 Register (FEE0 0360H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT1 pin.
- **LVT Error Register (FEE0 0370H)** — Specifies interrupt delivery when the APIC detects an internal error (see Section 11.5.3, "Error Handling").

The LVT performance counter register and its associated interrupt were introduced in the P6 processors and are also present in the Pentium 4 and Intel Xeon processors. The LVT thermal monitor register and its associated interrupt were introduced in the Pentium 4 and Intel Xeon processors. The LVT CMCI register and its associated interrupt were introduced in the Intel Xeon 5500 processors.

As shown in Figure 11-8, some of these fields and flags are not available (and reserved) for some entries.

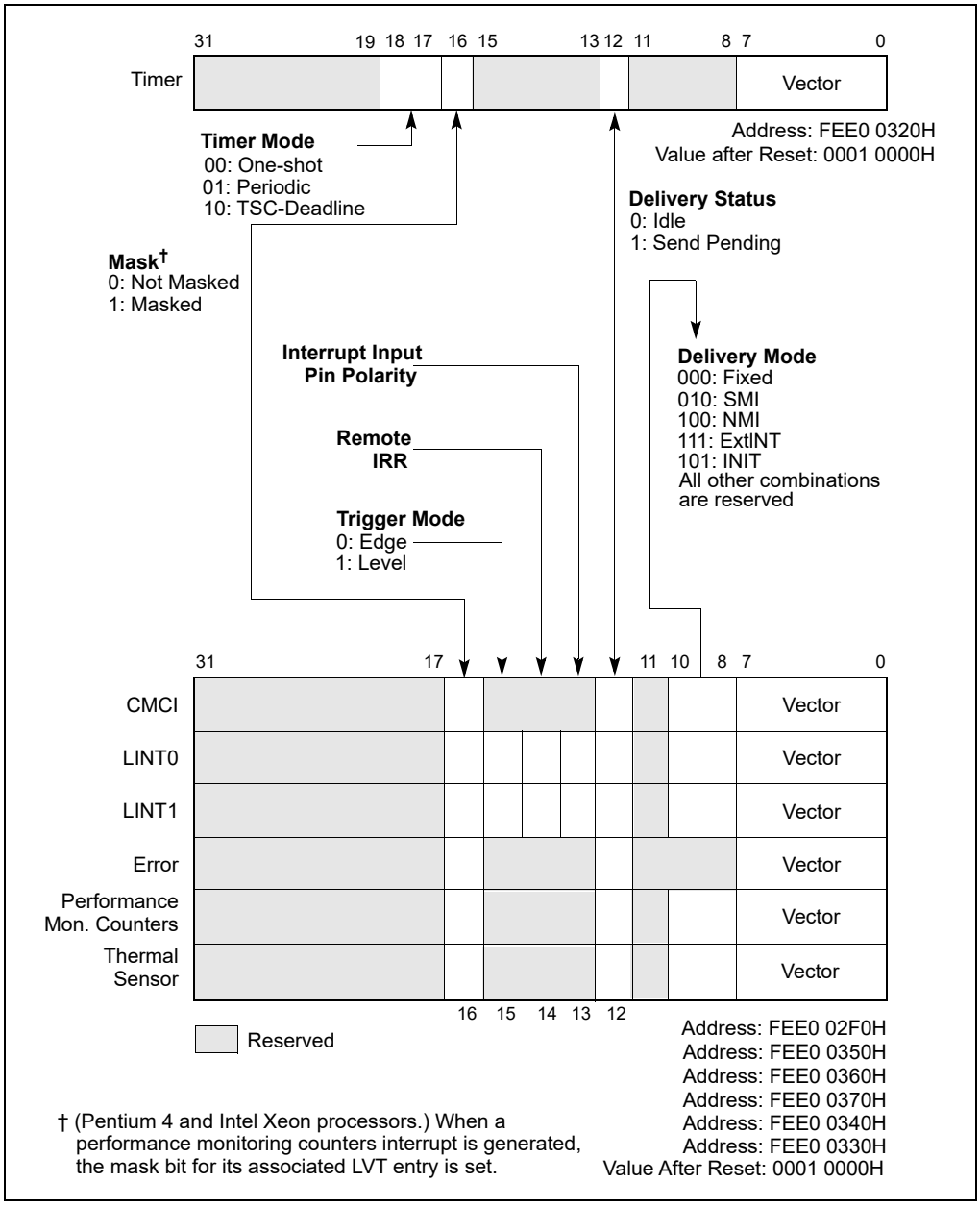


Figure 11-8. Local Vector Table (LVT)

The setup information that can be specified in the registers of the LVT table is as follows:

Vector Interrupt vector number.

Delivery Mode Specifies the type of interrupt to be sent to the processor. Some delivery modes will only operate as intended when used in conjunction with a specific trigger mode. The allowable delivery modes are as follows:

- **000 (Fixed)** Delivers the interrupt specified in the vector field.
- **010 (SMI)** Delivers an SMI interrupt to the processor core through the processor's local SMI signal path. When using this delivery mode, the vector field should be set to 00H for future compatibility.

100 (NMI) Delivers an NMI interrupt to the processor. The vector information is ignored.

101 (INIT) Delivers an INIT request to the processor core, which causes the processor to perform an INIT. When using this delivery mode, the vector field should be set to 00H for future compatibility. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.

110 Reserved; not supported for any LVT register.

111 (ExtINT) Causes the processor to respond to the interrupt as if the interrupt originated in an externally connected (8259A-compatible) interrupt controller. A special INTA bus cycle corresponding to ExtINT, is routed to the external controller. The external controller is expected to supply the vector information. The APIC architecture supports only one ExtINT source in a system, usually contained in the compatibility bridge. Only one processor in the system should have an LVT entry configured to use the ExtINT delivery mode. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.

Delivery Status (Read Only)
Indicates the interrupt delivery status, as follows:

0 (Idle) There is currently no activity for this interrupt source, or the previous interrupt from this source was delivered to the processor core and accepted.

1 (Send Pending)
Indicates that an interrupt from this source has been delivered to the processor core but has not yet been accepted (see Section 11.5.5, "Local Interrupt Acceptance").

Interrupt Input Pin Polarity
Specifies the polarity of the corresponding interrupt pin: (0) active high or (1) active low.

Remote IRR Flag (Read Only)
For fixed mode, level-triggered interrupts; this flag is set when the local APIC accepts the interrupt for servicing and is reset when an EOI command is received from the processor. The meaning of this flag is undefined for edge-triggered interrupts and other delivery modes.

Trigger Mode Selects the trigger mode for the local LINT0 and LINT1 pins: (0) edge sensitive and (1) level sensitive. This flag is only used when the delivery mode is Fixed. When the delivery mode is NMI, SMI, or INIT, the trigger mode is always edge sensitive. When the delivery mode is ExtINT, the trigger mode is always level sensitive. The timer and error interrupts are always treated as edge sensitive.

If the local APIC is not used in conjunction with an I/O APIC and fixed delivery mode is selected; the Pentium 4, Intel Xeon, and P6 family processors will always use level-sensitive triggering, regardless if edge-sensitive triggering is selected.

Software should always set the trigger mode in the LVT LINT1 register to 0 (edge sensitive). Level-sensitive interrupts are not supported for LINT1.

Mask Interrupt mask: (0) enables reception of the interrupt and (1) inhibits reception of the interrupt. When the local APIC handles a performance-monitoring counters interrupt, it automatically sets the mask flag in the LVT performance counter register. This flag is set to 1 on reset. It can be cleared only by software.

Timer Mode Bits 18:17 selects the timer mode (see Section 11.5.4):

(00b) one-shot mode using a count-down value,

(01b) periodic mode reloading a count-down value,

(10b) TSC-Deadline mode using absolute target value in IA32_TSC_DEADLINE MSR (see Section 11.5.4.1),

(11b) is reserved.

11.5.2 Valid Interrupt Vectors

The Intel 64 and IA-32 architectures define 256 vector numbers, ranging from 0 through 255 (see Section 6.2, "Exception and Interrupt Vectors"). Local and I/O APICs support 240 of these vectors (in the range of 16 to 255) as valid interrupts.

When an interrupt vector in the range of 0 to 15 is sent or received through the local APIC, the APIC indicates an illegal vector in its Error Status Register (see Section 11.5.3, "Error Handling"). The Intel 64 and IA-32 architectures reserve vectors 16 through 31 for predefined interrupts, exceptions, and Intel-reserved encodings (see Table 6-1). However, the local APIC does not treat vectors in this range as illegal.

When an illegal vector value (0 to 15) is written to an LVT entry and the delivery mode is Fixed (bits 8-11 equal 0), the APIC may signal an illegal vector error, without regard to whether the mask bit is set or whether an interrupt is actually seen on the input.

11.5.3 Error Handling

The local APIC records errors detected during interrupt handling in the error status register (ESR). The format of the ESR is given in Figure 11-9; it contains the following flags:

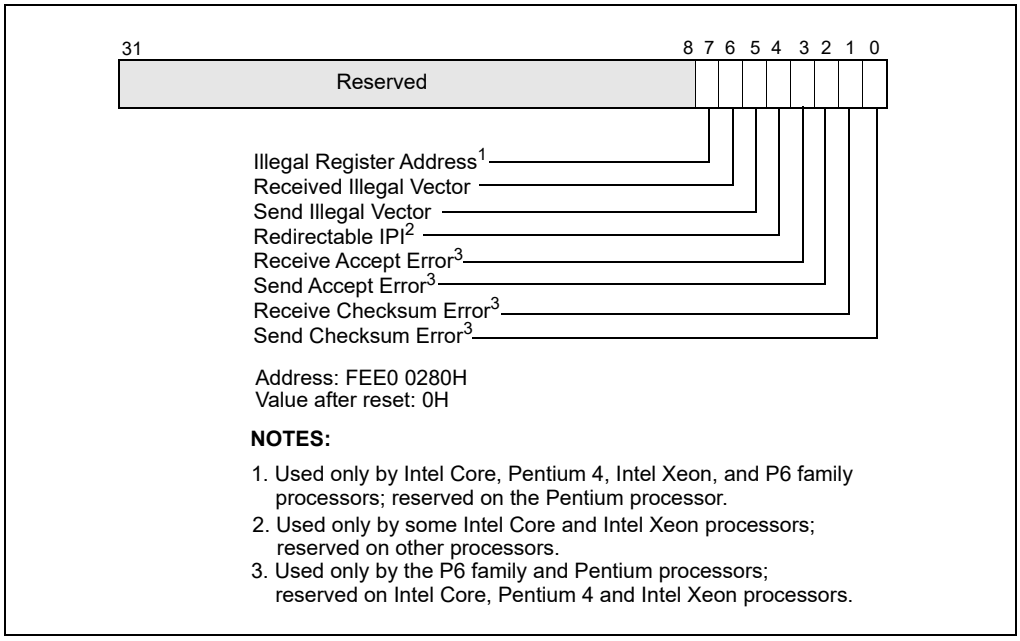


Figure 11-9. Error Status Register (ESR)

- Bit 0: Send Checksum Error.
  Set when the local APIC detects a checksum error for a message that it sent on the APIC bus. Used only on P6 family and Pentium processors.
- Bit 1: Receive Checksum Error.
  Set when the local APIC detects a checksum error for a message that it received on the APIC bus. Used only on P6 family and Pentium processors.
- Bit 2: Send Accept Error.
  Set when the local APIC detects that a message it sent was not accepted by any APIC on the APIC bus. Used only on P6 family and Pentium processors.
- Bit 3: Receive Accept Error.
  Set when the local APIC detects that the message it received was not accepted by any APIC on the APIC bus, including itself. Used only on P6 family and Pentium processors.

- Bit 4: Redirectable IPI.
  Set when the local APIC detects an attempt to send an IPI with the lowest-priority delivery mode and the local APIC does not support the sending of such IPIs. This bit is used on some Intel Core and Intel Xeon processors. As noted in Section 11.6.2, the ability of a processor to send a lowest-priority IPI is model-specific and should be avoided.
- Bit 5: Send Illegal Vector.
  Set when the local APIC detects an illegal vector (one in the range 0 to 15) in the message that it is sending. This occurs as the result of a write to the ICR (in both xAPIC and x2APIC modes) or to SELF IPI register (x2APIC mode only) with an illegal vector.

  If the local APIC does not support the sending of lowest-priority IPIs and software writes the ICR to send a lowest-priority IPI with an illegal vector, the local APIC sets only the "redirectable IPI" error bit. The interrupt is not processed and hence the "Send Illegal Vector" bit is not set in the ESR.
- Bit 6: Receive Illegal Vector.
  Set when the local APIC detects an illegal vector (one in the range 0 to 15) in an interrupt message it receives or in an interrupt generated locally from the local vector table or via a self IPI. Such interrupts are not delivered to the processor; the local APIC will never set an IRR bit in the range 0 to 15.
- Bit 7: Illegal Register Address
  Set when the local APIC is in xAPIC mode and software attempts to access a register that is reserved in the processor's local-APIC register-address space; see Table 10-1. (The local-APIC register-address space comprises the 4 KBytes at the physical address specified in the IA32_APIC_BASE MSR.) Used only on Intel Core, Intel Atom, Pentium 4, Intel Xeon, and P6 family processors.

  In x2APIC mode, software accesses the APIC registers using the RDMSR and WRMSR instructions. Use of one of these instructions to access a reserved register cause a general-protection exception (see Section 10.12.1.3). They do not set the "Illegal Register Access" bit in the ESR.

The ESR is a write/read register. Before attempt to read from the ESR, software should first write to it. (The value written does not affect the values read subsequently; only zero may be written in x2APIC mode.) This write clears any previously logged errors and updates the ESR with any errors detected since the last write to the ESR. This write also rearms the APIC error interrupt triggering mechanism.

The LVT Error Register (see Section 11.5.1) allows specification of the vector of the interrupt to be delivered to the processor core when APIC error is detected. The register also provides a means of masking an APIC-error interrupt. This masking only prevents delivery of APIC-error interrupts; the APIC continues to record errors in the ESR.

11.5.4 APIC Timer

The local APIC unit contains a 32-bit programmable timer that is available to software to time events or operations. This timer is set up by programming four registers: the divide configuration register (see Figure 11-10), the initial-count and current-count registers (see Figure 11-11), and the LVT timer register (see Figure 11-8).

If CPUID.06H:EAX.ARAT[bit 2] = 1, the processor's APIC timer runs at a constant rate regardless of P-state transitions and it continues to run at the same rate in deep C-states.

If CPUID.06H:EAX.ARAT[bit 2] = 0 or if CPUID 06H is not supported, the APIC timer may temporarily stop while the processor is in deep C-states or during transitions caused by Enhanced Intel SpeedStep® Technology.

The APIC timer frequency will be the processor's bus clock or core crystal clock frequency (when TSC/core crystal clock ratio is enumerated in CPUID leaf 0x15) divided by the value specified in the divide configuration register.

The timer can be configured through the timer LVT entry for one-shot or periodic operation. In one-shot mode, the timer is started by programming its initial-count register. The initial count value is then copied into the current-count register and count-down begins. After the timer reaches zero, a timer interrupt is generated and the timer remains at its 0 value until reprogrammed.

In periodic mode, the timer is started by writing to the initial-count register (as in one-shot mode), and the value written is copied into the current-count register, which counts down. The current-count register is automatically reloaded from the initial-count register when the count reaches 0 and a timer interrupt is generated, and the count-down is repeated. If during the count-down process the initial-count register is set, counting will restart, using the new initial-count value. The initial-count register is a read-write register; the current-count register is read only.

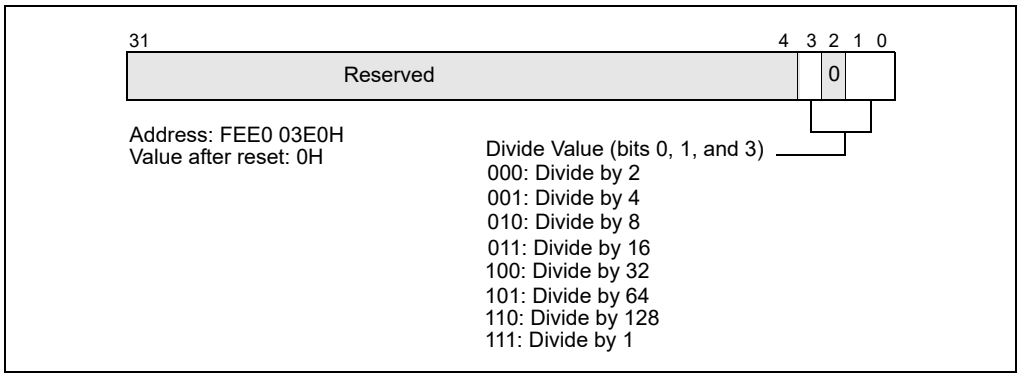


Figure 11-10. Divide Configuration Register

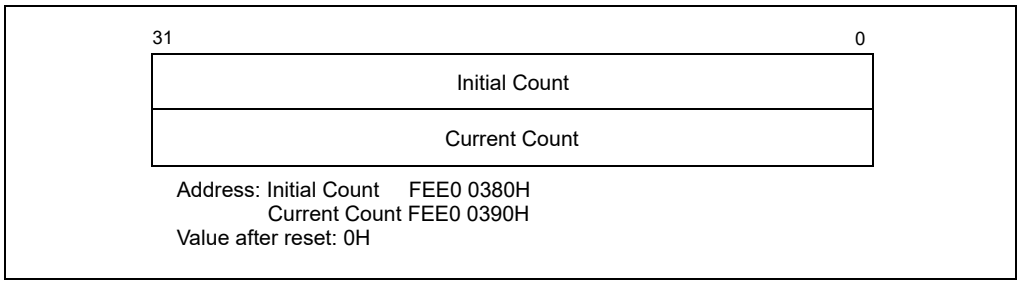


Figure 11-11. Initial Count and Current Count Registers

A write of 0 to the initial-count register effectively stops the local APIC timer, in both one-shot and periodic mode.

The LVT timer register determines the vector number that is delivered to the processor with the timer interrupt that is generated when the timer count reaches zero. The mask flag in the LVT timer register can be used to mask the timer interrupt.

NOTE

Changing the mode of the APIC timer (from one-shot to periodic or vice versa) by writing to the timer LVT entry does not start the timer. To start the timer, it is necessary to write to the initial-count register as described above.

11.5.4.1 TSC-Deadline Mode

The mode of operation of the local-APIC timer is determined by the LVT Timer Register. Specifically:

- If CPUID.01H:ECX.TSC_Deadline[bit 24] = 0, the mode is determined by bit 17 of the register.
- If CPUID.01H:ECX.TSC_Deadline[bit 24] = 1, the mode is determined by bits 18:17. See Figure 11-8. (If CPUID.01H:ECX.TSC_Deadline[bit 24] = 0, bit 18 of the register is reserved.)

The supported timer modes are given in Table 11-2. The three modes of the local APIC timer are mutually exclusive.

Table 11-2. Local APIC Timer Modes

LVT Bits [18:17]	Timer Mode
00b	One-shot mode, program count-down value in an initial-count register. See Section 11.5.4
01b	Periodic mode, program interval value in an initial-count register. See Section 11.5.4
10b	TSC-Deadline mode, program target value in IA32_TSC_DEADLINE MSR.
11b	Reserved

TSC-deadline mode allows software to use the local APIC timer to signal an interrupt at an absolute time. In TSC-deadline mode, writes to the initial-count register are ignored; and current-count register always reads 0. Instead, timer behavior is controlled using the IA32_TSC_DEADLINE MSR.

The IA32_TSC_DEADLINE MSR (MSR address 6E0H) is a per-logical processor MSR that specifies the time at which a timer interrupt should occur. Writing a non-zero 64-bit value into IA32_TSC_DEADLINE arms the timer. An interrupt is generated when the logical processor's time-stamp counter equals or exceeds the target value in the IA32_TSC_DEADLINE MSR.[1] When the timer generates an interrupt, it disarms itself and clears the IA32_TSC_DEADLINE MSR. Thus, each write to the IA32_TSC_DEADLINE MSR generates at most one timer interrupt.

In TSC-deadline mode, writing 0 to the IA32_TSC_DEADLINE MSR disarms the local-APIC timer. Transitioning between TSC-deadline mode and other timer modes also disarms the timer.

The hardware reset value of the IA32_TSC_DEADLINE MSR is 0. In other timer modes (LVT bit 18 = 0), the IA32_TSC_DEADLINE MSR reads zero and writes are ignored.

Software can configure the TSC-deadline timer to deliver a single interrupt using the following algorithm:

1. Detect support for TSC-deadline mode by verifying CPUID.1:ECX.24 = 1.
2. Select the TSC-deadline mode by programming bits 18:17 of the LVT Timer register with 10b.
3. Program the IA32_TSC_DEADLINE MSR with the target TSC value at which the timer interrupt is desired. This causes the processor to arm the timer.
4. The processor generates a timer interrupt when the value of time-stamp counter is greater than or equal to that of IA32_TSC_DEADLINE. It then disarms the timer and clear the IA32_TSC_DEADLINE MSR. (Both the time-stamp counter and the IA32_TSC_DEADLINE MSR are 64-bit unsigned integers.)
5. Software can re-arm the timer by repeating step 3.

The following are usage guidelines for TSC-deadline mode:

- Writes to the IA32_TSC_DEADLINE MSR are not serialized. Therefore, system software should not use WRMSR to the IA32_TSC_DEADLINE MSR as a serializing instruction. Read and write accesses to the IA32_TSC_DEADLINE and other MSR registers will occur in program order.
- Software can disarm the timer at any time by writing 0 to the IA32_TSC_DEADLINE MSR.
- If timer is armed, software can change the deadline (forward or backward) by writing a new value to the IA32_TSC_DEADLINE MSR.
- If software disarms the timer or postpones the deadline, race conditions may result in delivery of a timer interrupt associated with the original deadline. If the deadline has been postponed, software can identify such interrupts by reading the time-stamp counter and comparing its value to the new deadline.[1]
- In xAPIC mode (in which the local-APIC registers are memory-mapped), software must order the memory-mapped write to the LVT entry that enables TSC-deadline mode and any subsequent WRMSR to the IA32_TSC_DEADLINE MSR. Software can assure proper ordering by executing the MFENCE instruction after the memory-mapped write and before any WRMSR. (In x2APIC mode, the WRMSR instruction is used to write to the LVT entry. The processor ensures the ordering of this write and any subsequent WRMSR to the deadline; no fencing is required.)

11.5.5 Local Interrupt Acceptance

When a local interrupt is sent to the processor core, it is subject to the acceptance criteria specified in the interrupt acceptance flow chart in Figure 11-17. If the interrupt is accepted, it is logged into the IRR register and handled by the processor according to its priority (see Section 11.8.4, "Interrupt Acceptance for Fixed Interrupts"). If the interrupt is not accepted, it is sent back to the local APIC and retried.

1. If the logical processor is in VMX non-root operation, a read of the time-stamp counter (using either RDMSR, RDTSC, or RDTSCP) may not return the actual value of the time-stamp counter; see Chapter 26 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C. It is the responsibility of software operating in VMX root operation to coordinate the virtualization of the time-stamp counter and the IA32_TSC_DEADLINE MSR.

11.6 ISSUING INTERPROCESSOR INTERRUPTS

The following sections describe the local APIC facilities that are provided for issuing interprocessor interrupts (IPIs) from software. The primary local APIC facility for issuing IPIs is the interrupt command register (ICR). The ICR can be used for the following functions:

- To send an interrupt to another processor.
- To allow a processor to forward an interrupt that it received but did not service to another processor for servicing.
- To direct the processor to interrupt itself (perform a self interrupt).
- To deliver special IPIs, such as the start-up IPI (SIPI) message, to other processors.

Interrupts generated with this facility are delivered to the other processors in the system through the system bus (for Pentium 4 and Intel Xeon processors) or the APIC bus (for P6 family and Pentium processors). The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.

11.6.1 Interrupt Command Register (ICR)

The interrupt command register (ICR) is a 64-bit[1] local APIC register (see Figure 11-12) that allows software running on the processor to specify and send interprocessor interrupts (IPIs) to other processors in the system.

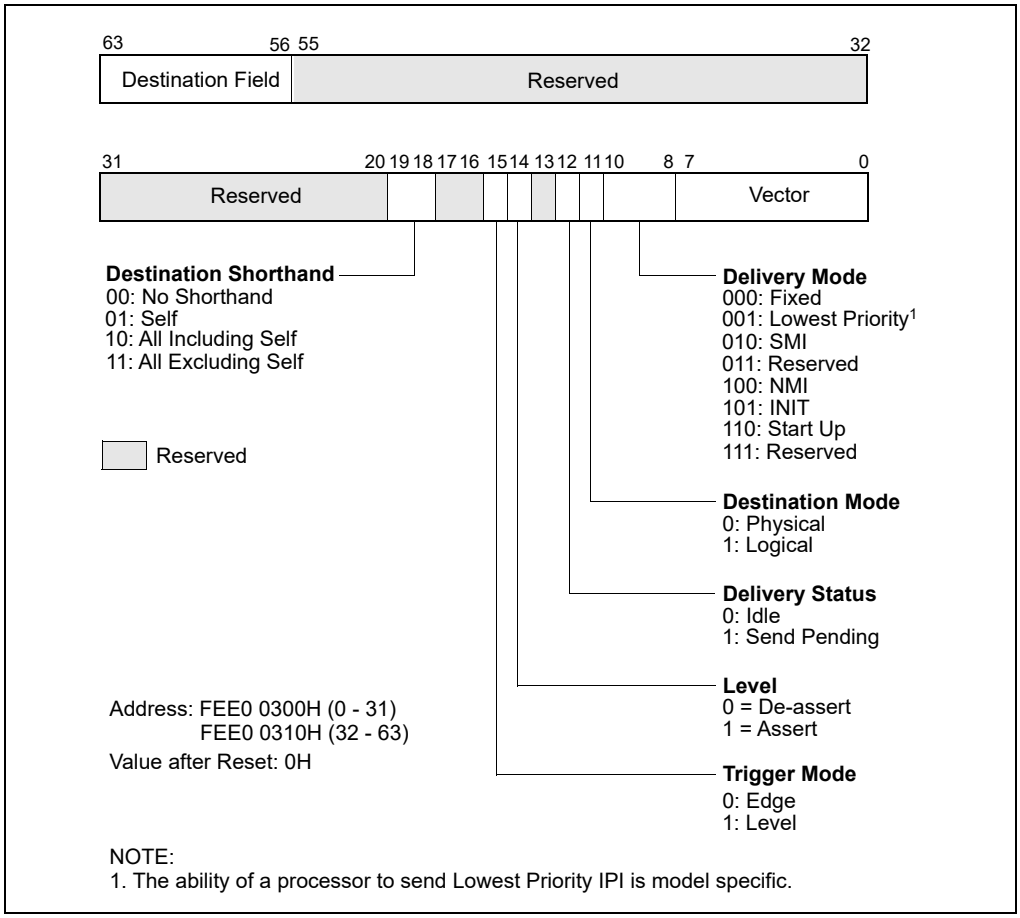


Figure 11-12. Interrupt Command Register (ICR)

1. In XAPIC mode the ICR is addressed as two 32-bit registers, ICR_LOW (FFE0 0300H) and ICR_HIGH (FFE0 0310H). In x2APIC mode, the ICR uses MSR 830H.

To send an IPI, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. (All fields of the ICR are read-write by software with the exception of the delivery status field, which is read-only.) The act of writing to the low doubleword of the ICR causes the IPI to be sent.

The ICR consists of the following fields.

Vector The vector number of the interrupt being sent.

Delivery Mode Specifies the type of IPI to be sent. This field is also know as the IPI message type field.

- **000 (Fixed)** Delivers the interrupt specified in the vector field to the target processor or processors.
- **001 (Lowest Priority)** Same as fixed mode, except that the interrupt is delivered to the processor executing at the lowest priority among the set of processors specified in the destination field. The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.
- **010 (SMI)** Delivers an SMI interrupt to the target processor or processors. The vector field must be programmed to 00H for future compatibility.
- **011 (Reserved)**
- **100 (NMI)** Delivers an NMI interrupt to the target processor or processors. The vector information is ignored.
- **101 (INIT)** Delivers an INIT request to the target processor or processors, which causes them to perform an INIT. As a result of this IPI message, all the target processors perform an INIT. The vector field must be programmed to 00H for future compatibility.
- **101 (INIT Level De-assert)** (Not supported in the Pentium 4 and Intel Xeon processors.) Sends a synchronization message to all the local APICs in the system to set their arbitration IDs (stored in their Arb ID registers) to the values of their APIC IDs (see Section 11.7, "System and APIC Bus Arbitration"). For this delivery mode, the level flag must be set to 0 and trigger mode flag to 1. This IPI is sent to all processors, regardless of the value in the destination field or the destination shorthand field; however, software should specify the "all including self" shorthand.
- **110 (Start-Up)** Sends a special "start-up" IPI (called a SIPI) to the target processor or processors. The vector typically points to a start-up routine that is part of the BIOS boot-strap code (see Section 9.4, "Multiple-Processor (MP) Initialization"). IPIs sent with this delivery mode are not automatically retried if the source APIC is unable to deliver it. It is up to the software to determine if the SIPI was not successfully delivered and to reissue the SIPI if necessary.

Destination Mode Selects either physical (0) or logical (1) destination mode (see Section 11.6.2, "Determining IPI Destination").

Delivery Status (Read Only) Indicates the IPI delivery status, as follows:

- **0 (Idle)** Indicates that this local APIC has completed sending any previous IPIs.
- **1 (Send Pending)** Indicates that this local APIC has not completed sending the last IPI.

Level For the INIT level de-assert delivery mode this flag must be set to 0; for all other delivery modes it must be set to 1. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 1.)

Trigger Mode Selects the trigger mode when using the INIT level de-assert delivery mode: edge (0) or level (1). It is ignored for all other delivery modes. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 0.)

Destination Shorthand

Indicates whether a shorthand notation is used to specify the destination of the interrupt and, if so, which shorthand is used. Destination shorthands are used in place of the 8-bit destination field, and can be sent by software using a single write to the low doubleword of the ICR. Shorthands are defined for the following cases: software self interrupt, IPIs to all processors in the system including the sender, IPIs to all processors in the system excluding the sender.

00: (No Shorthand)
The destination is specified in the destination field.

01: (Self) The issuing APIC is the one and only destination of the IPI. This destination shorthand allows software to interrupt the processor on which it is executing. An APIC implementation is free to deliver the self-interrupt message internally or to issue the message to the bus and "snoop" it as with any other IPI message.

10: (All Including Self)
The IPI is sent to all processors in the system including the processor sending the IPI. The APIC will broadcast an IPI message with the destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors.

11: (All Excluding Self)
The IPI is sent to all processors in a system with the exception of the processor sending the IPI. The APIC broadcasts a message with the physical destination mode and destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors. Support for this destination shorthand in conjunction with the lowest-priority delivery mode is model specific. For Pentium 4 and Intel Xeon processors, when this shorthand is used together with lowest priority delivery mode, the IPI may be redirected back to the issuing processor.

Destination Specifies the target processor or processors. This field is only used when the destination shorthand field is set to 00B. If the destination mode is set to physical, then bits 56 through 59 contain the APIC ID of the target processor for Pentium and P6 family processors and bits 56 through 63 contain the APIC ID of the target processor the for Pentium 4 and Intel Xeon processors. If the destination mode is set to logical, the interpretation of the 8-bit destination field depends on the settings of the DFR and LDR registers of the local APICs in all the processors in the system (see Section 11.6.2, "Determining IPI Destination").

Not all combinations of options for the ICR are valid. Table 11-3 shows the valid combinations for the fields in the ICR for the Pentium 4 and Intel Xeon processors; Table 11-4 shows the valid combinations for the fields in the ICR for the P6 family processors. Also note that the lower half of the ICR may not be preserved over transitions to the deepest C-States.

ICR operation in x2APIC mode is discussed in Section 11.12.9.

Table 11-3. Valid Combinations for Pentium 4 and Intel Xeon Processors Local xAPIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes[1]	Physical or Logical
No Shorthand	Invalid[2]	Level	All Modes	Physical or Logical
Self	Valid	Edge	Fixed	X[3]
Self	Invalid[2]	Level	Fixed	X
Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Including Self	Valid	Edge	Fixed	X
All Including Self	Invalid[2]	Level	Fixed	X

Table 11-3. Valid Combinations for Pentium 4 and Intel Xeon Processors Local xAPIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
All Including Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Excluding Self	Valid	Edge	Fixed, Lowest Priority[1, 4], NMI, INIT, SMI, Start-Up	X
All Excluding Self	Invalid[2]	Level	FIxed, Lowest Priority[4], NMI, INIT, SMI, Start-Up	X

NOTES:

1. The ability of a processor to send a lowest priority IPI is model specific.
2. For these interrupts, if the trigger mode bit is 1 (Level), the local xAPIC will override the bit setting and issue the interrupt as an edge triggered interrupt.
3. X means the setting is ignored.
4. When using the "lowest priority" delivery mode and the "all excluding self" destination, the IPI can be redirected back to the issuing APIC, which is essentially the same as the "all including self" destination mode.

Table 11-4. Valid Combinations for the P6 Family Processor Local APIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes[1]	Physical or Logical
No Shorthand	Valid[2]	Level	Fixed, Lowest Priority[1], NMI	Physical or Logical
No Shorthand	Valid[3]	Level	INIT	Physical or Logical
Self	Valid	Edge	Fixed	X[4]
Self	Valid[2]	Level	Fixed	X
Self	Invalid[5]	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All including Self	Valid	Edge	Fixed	X
All including Self	Valid[2]	Level	Fixed	X
All including Self	Invalid[5]	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All excluding Self	Valid	Edge	All Modes[1]	X
All excluding Self	Valid[2]	Level	Fixed, Lowest Priority[1], NMI	X
All excluding Self	Invalid[5]	Level	SMI, Start-Up	X
All excluding Self	Valid[3]	Level	INIT	X
X	Invalid[5]	Level	SMI, Start-Up	X

NOTES:

1. The ability of a processor to send a lowest priority IPI is model specific.
2. Treated as edge triggered if level bit is set to 1, otherwise ignored.
3. Treated as edge triggered when Level bit is set to 1; treated as "INIT Level Deassert" message when level bit is set to 0 (deassert). Only INIT level deassert messages are allowed to have the level bit set to 0. For all other messages the level bit must be set to 1.
4. X means the setting is ignored.
5. The behavior of the APIC is undefined.

11.6.2 Determining IPI Destination

The destination of an IPI[1] can be one, all, or a subset (group) of the processors on the system bus. The sender of the IPI specifies the destination of an IPI with the following APIC registers and fields within the registers:

- **ICR Register** — The following fields in the ICR register are used to specify the destination of an IPI.

1. Determination of IPI destinations in x2APIC mode is discussed in Section 10.12.10.

  - **Destination Mode** — Selects one of two destination modes (physical or logical).
  - **Destination Field** — In physical destination mode, used to specify the APIC ID of the destination processor; in logical destination mode, used to specify a message destination address (MDA) that can be used to select specific processors in clusters.
  - **Destination Shorthand** — A quick method of specifying all processors, all excluding self, or self as the destination.
  - **Delivery mode, Lowest Priority** — Architecturally specifies that a lowest-priority arbitration mechanism be used to select a destination processor from a specified group of processors. The ability of a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.
- **Local destination register (LDR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.
- **Destination format register (DFR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.

How the ICR, LDR, and DFR are used to select an IPI destination depends on the destination mode used: physical, logical, broadcast/self, or lowest-priority delivery mode. These destination modes are described in the following sections.

11.6.2.1 Physical Destination Mode

In physical destination mode, the destination processor is specified by its local APIC ID (see Section 11.4.6, "Local APIC ID"). For Pentium 4 and Intel Xeon processors, either a single destination (local APIC IDs 00H through FEH) or a broadcast to all APICs (the APIC ID is FFH) may be specified in physical destination mode.

A broadcast IPI (bits 28-31 of the MDA are 1's) or I/O subsystem initiated interrupt with lowest priority delivery mode is not supported in physical destination mode and must not be configured by software. Also, for any non-broadcast IPI or I/O subsystem initiated interrupt with lowest priority delivery mode, software must ensure that APICs defined in the interrupt address are present and enabled to receive interrupts.

For the P6 family and Pentium processors, a single destination is specified in physical destination mode with a local APIC ID of 0H through 0EH, allowing up to 15 local APICs to be addressed on the APIC bus. A broadcast to all local APICs is specified with 0FH.

NOTE

The number of local APICs that can be addressed on the system bus may be restricted by hardware.

11.6.2.2 Logical Destination Mode

In logical destination mode, IPI destination is specified using an 8-bit message destination address (MDA), which is entered in the destination field of the ICR. Upon receiving an IPI message that was sent using logical destination mode, a local APIC compares the MDA in the message with the values in its LDR and DFR to determine if it should accept and handle the IPI. For both configurations of logical destination mode, when combined with lowest priority delivery mode, software is responsible for ensuring that all of the local APICs included in or addressed by the IPI or I/O subsystem interrupt are present and enabled to receive the interrupt.

Figure 11-13 shows the layout of the logical destination register (LDR). The 8-bit logical APIC ID field in this register is used to create an identifier that can be compared with the MDA.

NOTE

The logical APIC ID should not be confused with the local APIC ID that is contained in the local APIC ID register.

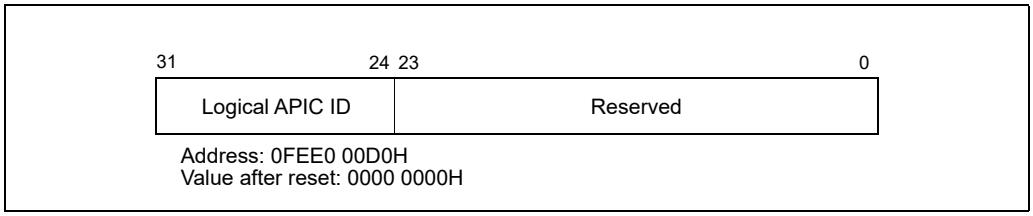


Figure 11-13. Logical Destination Register (LDR)

Figure 11-14 shows the layout of the destination format register (DFR). The 4-bit model field in this register selects one of two models (flat or cluster) that can be used to interpret the MDA when using logical destination mode.

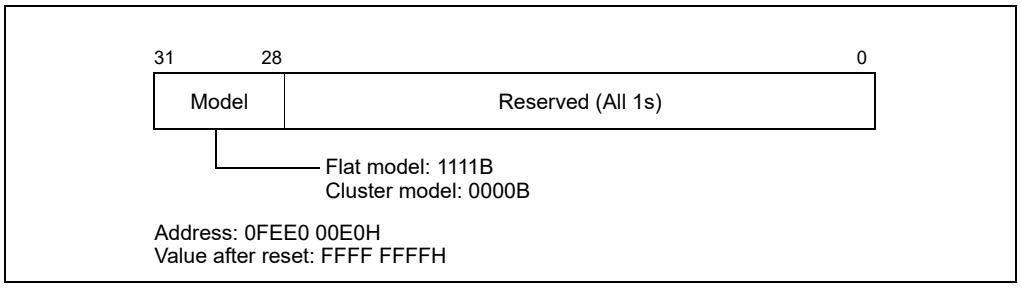


Figure 11-14. Destination Format Register (DFR)

The interpretation of MDA for the two models is described in the following paragraphs.

1. **Flat Model** — This model is selected by programming DFR bits 28 through 31 to 1111. Here, a unique logical APIC ID can be established for up to 8 local APICs by setting a different bit in the logical APIC ID field of the LDR for each local APIC. A group of local APICs can then be selected by setting one or more bits in the MDA.

   Each local APIC performs a bit-wise AND of the MDA and its logical APIC ID. If a true condition (non-zero) is detected, the local APIC accepts the IPI message. A broadcast to all APICs is achieved by setting the MDA to 1s.

2. **Cluster Model** — This model is selected by programming DFR bits 28 through 31 to 0000. This model supports two basic destination schemes: flat cluster and hierarchical cluster.

   The flat cluster destination model is only supported for P6 family and Pentium processors. Using this model, all APICs are assumed to be connected through the APIC bus. Bits 60 through 63 of the MDA contains the encoded address of the destination cluster and bits 56 through 59 identify up to four local APICs within the cluster (each bit is assigned to one local APIC in the cluster, as in the flat connection model). To identify one or more local APICs, bits 60 through 63 of the MDA are compared with bits 28 through 31 of the LDR to determine if a local APIC is part of the cluster. Bits 56 through 59 of the MDA are compared with Bits 24 through 27 of the LDR to identify a local APICs within the cluster.

   Sets of processors within a cluster can be specified by writing the target cluster address in bits 60 through 63 of the MDA and setting selected bits in bits 56 through 59 of the MDA, corresponding to the chosen members of the cluster. In this mode, 15 clusters (with cluster addresses of 0 through 14) each having 4 local APICs can be specified in the message. For the P6 and Pentium processor's local APICs, however, the APIC arbitration ID supports only 15 APIC agents. Therefore, the total number of processors and their local APICs supported in this mode is limited to 15. Broadcast to all local APICs is achieved by setting all destination bits to one. This guarantees a match on all clusters and selects all APICs in each cluster. A broadcast IPI or I/O subsystem broadcast interrupt with lowest priority delivery mode is not supported in cluster mode and must not be configured by software.

   The hierarchical cluster destination model can be used with Pentium 4, Intel Xeon, P6 family, or Pentium processors. With this model, a hierarchical network can be created by connecting different flat clusters via independent system or APIC buses. This scheme requires a cluster manager within each cluster, which is responsible for handling message passing between system or APIC buses. One cluster contains up to 4 agents. Thus 15 cluster managers, each with 4 agents, can form a network of up to 60 APIC agents. Note that hierar-

chical APIC networks requires a special cluster manager device, which is not part of the local or the I/O APIC units.

NOTES

All processors that have their APIC software enabled (using the spurious vector enable/disable bit) must have their DFRs (Destination Format Registers) programmed identically.

The default mode for DFR is flat mode. If you are using cluster mode, DFRs must be programmed before the APIC is software enabled. Since some chipsets do not accurately track a system view of the logical mode, program DFRs as soon as possible after starting the processor.

11.6.2.3 Broadcast/Self Delivery Mode

The destination shorthand field of the ICR allows the delivery mode to be by-passed in favor of broadcasting the IPI to all the processors on the system bus and/or back to itself (see Section 11.6.1, "Interrupt Command Register (ICR)"). Three destination shorthands are supported: self, all excluding self, and all including self. The destination mode is ignored when a destination shorthand is used.

11.6.2.4 Lowest Priority Delivery Mode

With lowest priority delivery mode, the ICR is programmed to send an IPI to several processors on the system bus, using the logical or shorthand destination mechanism for selecting the processor. The selected processors then arbitrate with one another over the system bus or the APIC bus, with the lowest-priority processor accepting the IPI.

For systems based on the Intel Xeon processor, the chipset bus controller accepts messages from the I/O APIC agents in the system and directs interrupts to the processors on the system bus. When using the lowest priority delivery mode, the chipset chooses a target processor to receive the interrupt out of the set of possible targets. The Pentium 4 processor provides a special bus cycle on the system bus that informs the chipset of the current task priority for each logical processor in the system. The chipset saves this information and uses it to choose the lowest priority processor when an interrupt is received.

For systems based on P6 family processors, the processor priority used in lowest-priority arbitration is contained in the arbitration priority register (APR) in each local APIC. Figure 11-15 shows the layout of the APR.

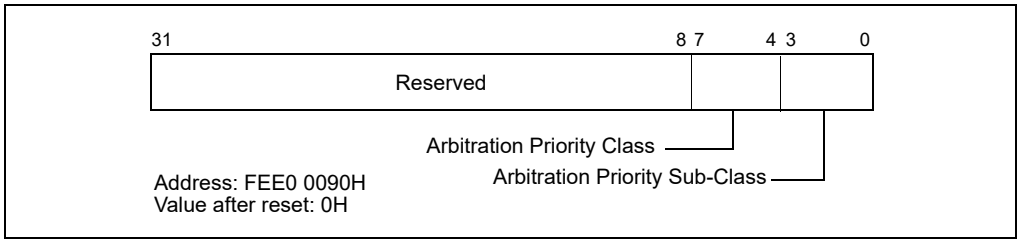


Figure 11-15. Arbitration Priority Register (APR)

The APR value is computed as follows:

```
IF (TPR[7:4] ≥ IRRV[7:4]) AND (TPR[7:4] > ISRV[7:4])
    THEN
        APR[7:0] ← TPR[7:0]
    ELSE
        APR[7:4] ← max(TPR[7:4] AND ISRV[7:4], IRRV[7:4])
        APR[3:0] ← 0.
```

Here, the TPR value is the task priority value in the TPR (see Figure 11-18), the IRRV value is the vector number for the highest priority bit that is set in the IRR (see Figure 11-20) or 00H (if no IRR bit is set), and the ISRV value is the vector number for the highest priority bit that is set in the ISR (see Figure 11-20). Following arbitration

among the destination processors, the processor with the lowest value in its APR handles the IPI and the other processors ignore it.

(P6 family and Pentium processors.) For these processors, if a **focus processor** exists**,** it may accept the interrupt, regardless of its priority. A processor is said to be the focus of an interrupt if it is currently servicing that interrupt or if it has a pending request for that interrupt. For Intel Xeon processors, the concept of a focus processor is not supported.

In operating systems that use the lowest priority delivery mode but do not update the TPR, the TPR information saved in the chipset will potentially cause the interrupt to be always delivered to the same processor from the logical set. This behavior is functionally backward compatible with the P6 family processor but may result in unexpected performance implications.

11.6.3 IPI Delivery and Acceptance

When the low double-word of the ICR is written to, the local APIC creates an IPI message from the information contained in the ICR and sends the message out on the system bus (Pentium 4 and Intel Xeon processors) or the APIC bus (P6 family and Pentium processors). The manner in which these IPIs are handled after being issued is described in Section 11.8, "Handling Interrupts."

11.7 SYSTEM AND APIC BUS ARBITRATION

When several local APICs and the I/O APIC are sending IPI and interrupt messages on the system bus (or APIC bus), the order in which the messages are sent and handled is determined through bus arbitration.

For the Pentium 4 and Intel Xeon processors, the local and I/O APICs use the arbitration mechanism defined for the system bus to determine the order in which IPIs are handled. This mechanism is non-architectural and cannot be controlled by software.

For the P6 family and Pentium processors, the local and I/O APICs use an APIC-based arbitration mechanism to determine the order in which IPIs are handled. Here, each local APIC is given an arbitration priority of from 0 to 15, which the I/O APIC uses during arbitration to determine which local APIC should be given access to the APIC bus. The local APIC with the highest arbitration priority always wins bus access. Upon completion of an arbitration round, the winning local APIC lowers its arbitration priority to 0 and the losing local APICs each raise theirs by 1.

The current arbitration priority for a local APIC is stored in a 4-bit, software-transparent arbitration ID (Arb ID) register. During reset, this register is initialized to the APIC ID number (stored in the local APIC ID register). The INIT level-deassert IPI, which is issued with and ICR command, can be used to resynchronize the arbitration priorities of the local APICs by resetting Arb ID register of each agent to its current APIC ID value. (The Pentium 4 and Intel Xeon processors do not implement the Arb ID register.)

Section 11.10, "APIC Bus Message Passing Mechanism and Protocol (P6 Family, Pentium Processors)," describes the APIC bus arbitration protocols and bus message formats, while Section 11.6.1, "Interrupt Command Register (ICR)," describes the INIT level de-assert IPI message.

Note that except for the SIPI IPI (see Section 11.6.1, "Interrupt Command Register (ICR)"), all bus messages that fail to be delivered to their specified destination or destinations are automatically retried. Software should avoid situations in which IPIs are sent to disabled or nonexistent local APICs, causing the messages to be resent repeatedly. Additionally, interrupt sources that target the APIC should be masked or changed to no longer target the APIC.

11.8 HANDLING INTERRUPTS

When a local APIC receives an interrupt from a local source, an interrupt message from an I/O APIC, or an IPI, the manner in which it handles the message depends on processor implementation, as described in the following sections.

11.8.1 Interrupt Handling with the Pentium 4 and Intel Xeon Processors

With the Pentium 4 and Intel Xeon processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows:

1. It determines if it is the specified destination or not (see Figure 11-16). If it is the specified destination, it accepts the message; if it is not, it discards the message.

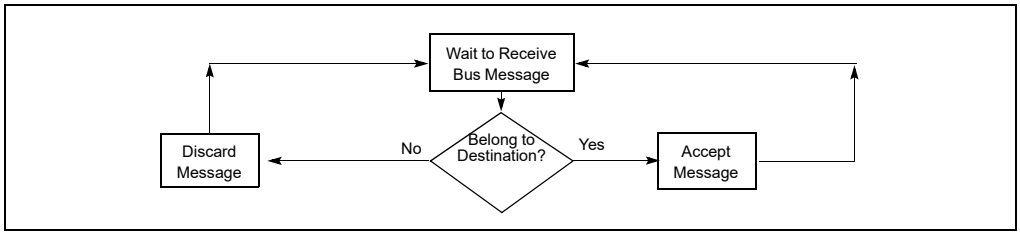


Figure 11-16. Interrupt Acceptance Flow Chart for the Local APIC (Pentium 4 and Intel Xeon Processors)

2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or SIPI, the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC sets the appropriate bit in the IRR.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 11.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI) register in the local APIC (see Section 11.8.5, "Signaling Interrupt Servicing Completion"). The act of writing to the EOI register causes the local APIC to delete the interrupt from its ISR queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

11.8.2 Interrupt Handling with the P6 Family and Pentium Processors

With the P6 family and Pentium processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows (see Figure 11-17).

1. (IPIs only) The local APIC examines the IPI message to determines if it is the specified destination for the IPI as described in Section 11.6.2, "Determining IPI Destination." If it is the specified destination, it continues its acceptance procedure; if it is not the destination, it discards the IPI message. When the message specifies lowest-priority delivery mode, the local APIC will arbitrate with the other processors that were designated as recipients of the IPI message (see Section 11.6.2.4, "Lowest Priority Delivery Mode").
2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or INIT-deassert interrupt, or one of the MP protocol IPI messages (BIPI, FIPI, and SIPI), the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC looks for an open slot in one of its two pending interrupt queues contained in the IRR and ISR registers (see Figure 11-20). If a slot is available (see Section 11.8.4, "Interrupt Acceptance for Fixed Interrupts"), places the interrupt in the slot. If a slot is not available, it rejects the interrupt request and sends it back to the sender with a retry message.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 11.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI)

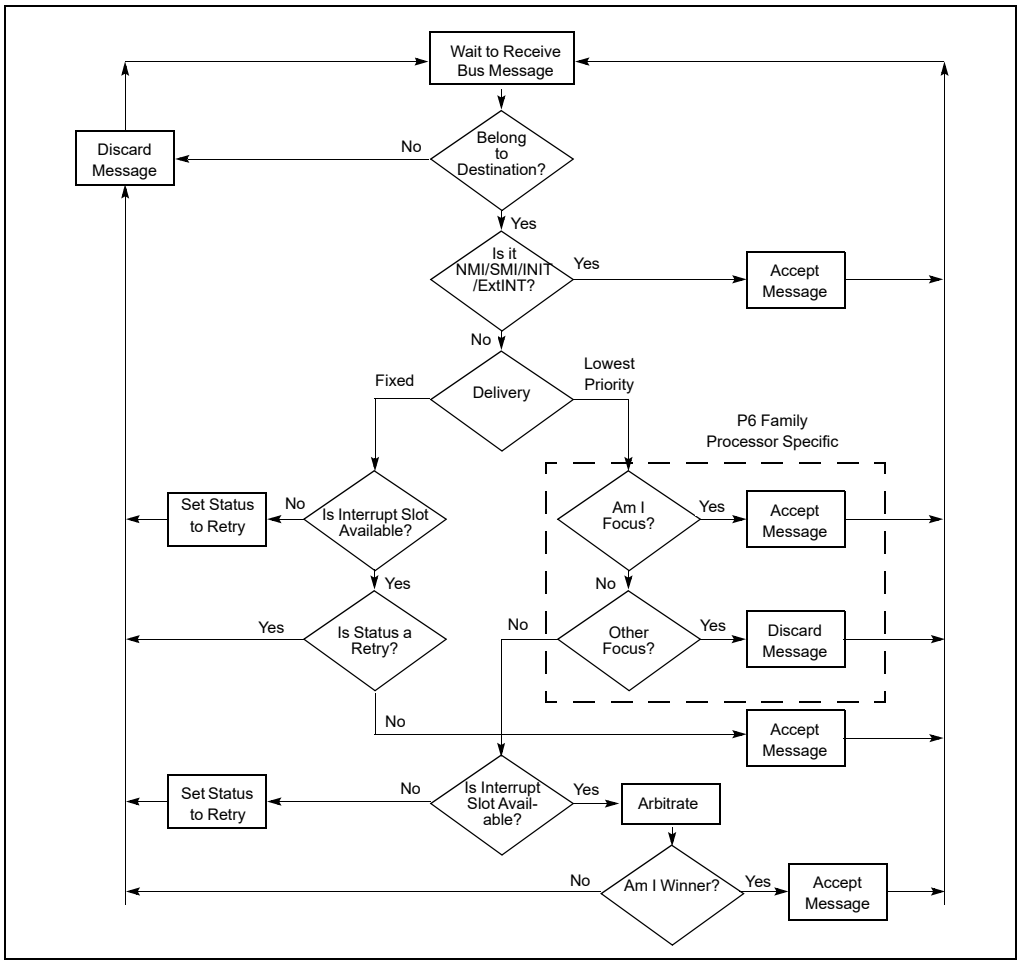


Figure 11-17. Interrupt Acceptance Flow Chart for the Local APIC (P6 Family and Pentium Processors)

register in the local APIC (see Section 11.8.5, "Signaling Interrupt Servicing Completion"). The act of writing to the EOI register causes the local APIC to delete the interrupt from its queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

The following sections describe the acceptance of interrupts and their handling by the local APIC and processor in greater detail.

11.8.3 Interrupt, Task, and Processor Priority

Each interrupt delivered to the processor through the local APIC has a priority based on its vector number. The local APIC uses this priority to determine when to service the interrupt relative to the other activities of the processor, including the servicing of other interrupts.

Each interrupt vector is an 8-bit value. The **interrupt-priority class** is the value of bits 7:4 of the interrupt vector. The lowest interrupt-priority class is 1 and the highest is 15; interrupts with vectors in the range 0–15 (with interrupt-priority class 0) are illegal and are never delivered. Because vectors 0–31 are reserved for dedicated uses by the Intel 64 and IA-32 architectures, software should configure interrupt vectors to use interrupt-priority classes in the range 2–15.

Each interrupt-priority class encompasses 16 vectors. The relative priority of interrupts within an interrupt-priority class is determined by the value of bits 3:0 of the vector number. The higher the value of those bits, the higher the

priority within that interrupt-priority class. Thus, each interrupt vector comprises two parts, with the high 4 bits indicating its interrupt-priority class and the low 4 bits indicating its ranking within the interrupt-priority class.

11.8.3.1 Task and Processor Priorities

The local APIC also defines a **task priority** and a **processor priority** that determine the order in which interrupts are handled. The **task-priority class** is the value of bits 7:4 of the task-priority register (TPR), which can be written by software (TPR is a read/write register); see Figure 11-18.

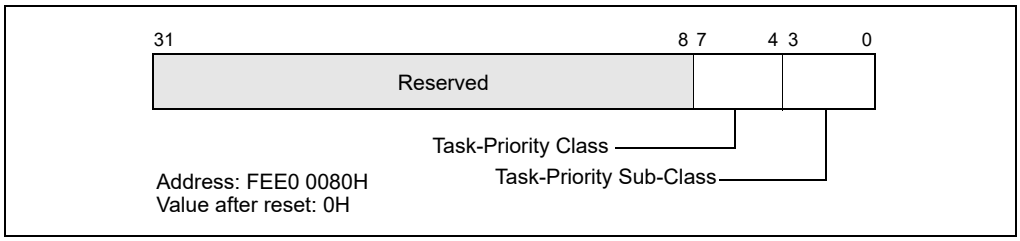


Figure 11-18. Task-Priority Register (TPR)

NOTE

In this discussion, the term "task" refers to a software defined task, process, thread, program, or routine that is dispatched to run on the processor by the operating system. It does not refer to an IA-32 architecture defined task as described in Chapter 8, "Task Management."

The task priority allows software to set a priority threshold for interrupting the processor. This mechanism enables the operating system to temporarily block low priority interrupts from disturbing high-priority work that the processor is doing. The ability to block such interrupts using task priority results from the way that the TPR controls the value of the processor-priority register (PPR).[1]

The **processor-priority class** is a value in the range 0–15 that is maintained in bits 7:4 of the processor-priority register (PPR); see Figure 11-19. The PPR is a read-only register. The processor-priority class represents the current priority at which the processor is executing.

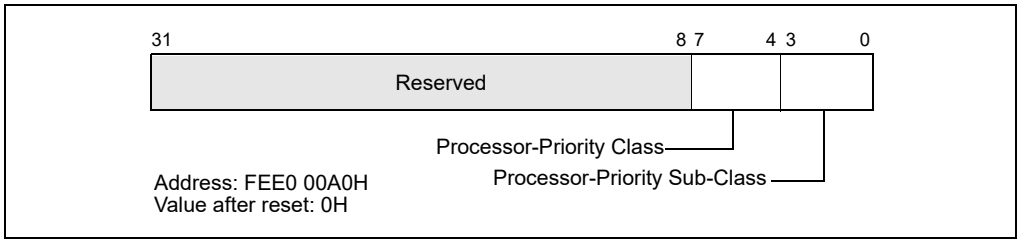


Figure 11-19. Processor-Priority Register (PPR)

The value of the PPR is based on the value of TPR and the value ISRV; ISRV is the vector number of the highest priority bit that is set in the ISR or 00H if no bit is set in the ISR. (See Section 11.8.4 for more details on the ISR.) The value of PPR is determined as follows:

- PPR[7:4] (the processor-priority class) the maximum of TPR[7:4] (the task- priority class) and ISRV[7:4] (the priority of the highest priority interrupt in service).
- PPR[3:0] (the processor-priority sub-class) is determined as follows:
  - — If TPR[7:4] > ISRV[7:4], PPR[3:0] is TPR[3:0] (the task-priority sub-class).
  - — If TPR[7:4] < ISRV[7:4], PPR[3:0] is 0.
  - — If TPR[7:4] = ISRV[7:4], PPR[3:0] may be either TPR[3:0] or 0. The actual behavior is model-specific.

1. The TPR also determines the arbitration priority of the local processor; see Section 11.6.2.4, "Lowest Priority Delivery Mode."

The processor-priority class determines the priority threshold for interrupting the processor. The processor will deliver only those interrupts that have an interrupt-priority class higher than the processor-priority class in the PPR. If the processor-priority class is 0, the PPR does not inhibit the delivery any interrupt; if it is 15, the processor inhibits the delivery of all interrupts. (The processor-priority mechanism does not affect the delivery of interrupts with the NMI, SMI, INIT, ExtINT, INIT-deassert, and start-up delivery modes.)

The processor does not use the processor-priority sub-class to determine which interrupts to deliver and which to inhibit. (The processor uses the processor-priority sub-class only to satisfy reads of the PPR.)

11.8.4 Interrupt Acceptance for Fixed Interrupts

The local APIC queues the fixed interrupts that it accepts in one of two interrupt pending registers: the interrupt request register (IRR) or in-service register (ISR). These two 256-bit read-only registers are shown in Figure 11-20. The 256 bits in these registers represent the 256 possible vectors; vectors 0 through 15 are reserved by the APIC (see also: Section 11.5.2, "Valid Interrupt Vectors").

NOTE

All interrupts with an NMI, SMI, INIT, ExtINT, start-up, or INIT-deassert delivery mode bypass the IRR and ISR registers and are sent directly to the processor core for servicing.

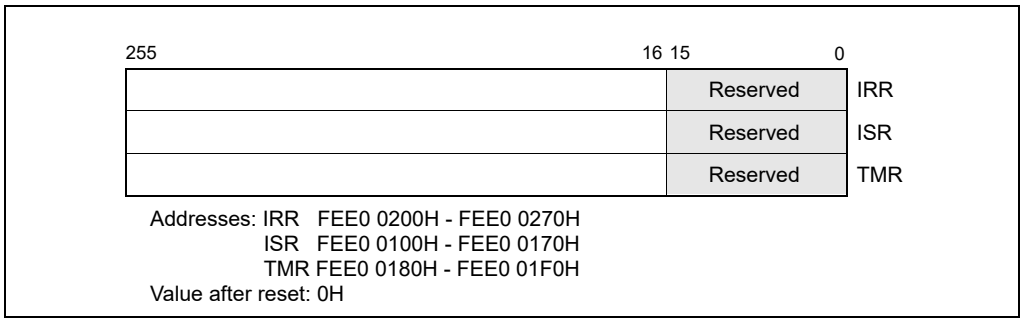


Figure 11-20. IRR, ISR, and TMR Registers

The IRR contains the active interrupt requests that have been accepted, but not yet dispatched to the processor for servicing. When the local APIC accepts an interrupt, it sets the bit in the IRR that corresponds the vector of the accepted interrupt. When the processor core is ready to handle the next interrupt, the local APIC clears the highest priority IRR bit that is set and sets the corresponding ISR bit. The vector for the highest priority bit set in the ISR is then dispatched to the processor core for servicing.

While the processor is servicing the highest priority interrupt, the local APIC can send additional fixed interrupts by setting bits in the IRR. When the interrupt service routine issues a write to the EOI register (see Section 11.8.5, "Signaling Interrupt Servicing Completion"), the local APIC responds by clearing the highest priority ISR bit that is set. It then repeats the process of clearing the highest priority bit in the IRR and setting the corresponding bit in the ISR. The processor core then begins executing the service routing for the highest priority bit set in the ISR.

If more than one interrupt is generated with the same vector number, the local APIC can set the bit for the vector both in the IRR and the ISR. This means that for the Pentium 4 and Intel Xeon processors, the IRR and ISR can queue two interrupts for each interrupt vector: one in the IRR and one in the ISR. Any additional interrupts issued for the same interrupt vector are collapsed into the single bit in the IRR.

For the P6 family and Pentium processors, the IRR and ISR registers can queue no more than two interrupts per interrupt vector and will reject other interrupts that are received within the same vector.

If the local APIC receives an interrupt with an interrupt-priority class higher than that of the interrupt currently in service, and interrupts are enabled in the processor core, the local APIC dispatches the higher priority interrupt to the processor immediately (without waiting for a write to the EOI register). The currently executing interrupt handler is then interrupted so the higher-priority interrupt can be handled. When the handling of the higher-priority interrupt has been completed, the servicing of the interrupted interrupt is resumed.

The trigger mode register (TMR) indicates the trigger mode of the interrupt (see Figure 11-20). Upon acceptance of an interrupt into the IRR, the corresponding TMR bit is cleared for edge-triggered interrupts and set for level-triggered interrupts. If a TMR bit is set when an EOI cycle for its corresponding interrupt vector is generated, an EOI message is sent to all I/O APICs.

11.8.5 Signaling Interrupt Servicing Completion

For all interrupts except those delivered with the NMI, SMI, INIT, ExtINT, the start-up, or INIT-Deassert delivery mode, the interrupt handler must include a write to the end-of-interrupt (EOI) register (see Figure 11-21). This write must occur at the end of the handler routine, sometime before the IRET instruction. This action indicates that the servicing of the current interrupt is complete and the local APIC can issue the next interrupt from the ISR.

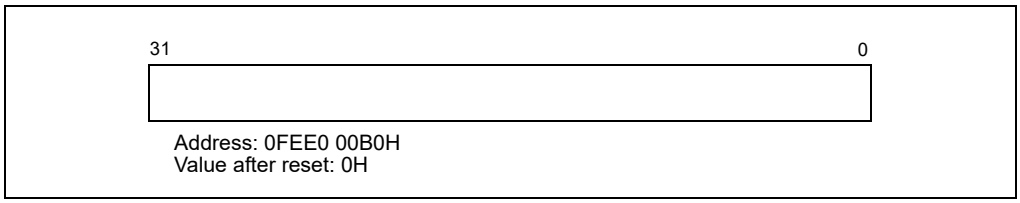


Figure 11-21. EOI Register

Upon receiving an EOI, the APIC clears the highest priority bit in the ISR and dispatches the next highest priority interrupt to the processor. If the terminated interrupt was a level-triggered interrupt, the local APIC also sends an end-of-interrupt message to all I/O APICs.

System software may prefer to direct EOIs to specific I/O APICs rather than having the local APIC send end-of-interrupt messages to all I/O APICs.

Software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register (see Section 11.9). If this bit is set, a broadcast EOI is not generated on an EOI cycle even if the associated TMR bit indicates that the current interrupt was level-triggered. The default value for the bit is 0, indicating that EOI broadcasts are performed.

Bit 12 of the Spurious Interrupt Vector Register is reserved to 0 if the processor does not support suppression of EOI broadcasts. Support for EOI-broadcast suppression is reported in bit 24 in the Local APIC Version Register (see Section 11.4.8); the feature is supported if that bit is set to 1. When supported, the feature is available in both xAPIC mode and x2APIC mode.

System software desiring to perform directed EOIs for level-triggered interrupts should set bit 12 of the Spurious Interrupt Vector Register and follow each the EOI to the local xAPIC for a level triggered interrupt with a directed EOI to the I/O APIC generating the interrupt (this is done by writing to the I/O APIC's EOI register). System software performing directed EOIs must retain a mapping associating level-triggered interrupts with the I/O APICs in the system.

11.8.6 Task Priority in IA-32e Mode

In IA-32e mode, operating systems can manage the 16 interrupt-priority classes (see Section 11.8.3, "Interrupt, Task, and Processor Priority") explicitly using the task priority register (TPR). Operating systems can use the TPR to temporarily block specific (low-priority) interrupts from interrupting a high-priority task. This is done by loading TPR with a value in which the task-priority class corresponds to the highest interrupt-priority class that is to be blocked. For example:

- Loading the TPR with a task-priority class of 8 (01000B) blocks all interrupts with an interrupt-priority class of 8 or less while allowing all interrupts with an interrupt-priority class of 9 or more to be recognized.
- Loading the TPR with a task-priority class of 0 enables all external interrupts.
- Loading the TPR with a task-priority class of 0FH (01111B) disables all external interrupts.

The TPR (shown in Figure 11-18) is cleared to 0 on reset. In 64-bit mode, software can read and write the TPR using an alternate interface, MOV CR8 instruction. The new task-priority class is established when the MOV CR8

instruction completes execution. Software does not need to force serialization after loading the TPR using MOV CR8.

Use of the MOV CRn instruction requires a privilege level of 0. Programs running at privilege level greater than 0 cannot read or write the TPR. An attempt to do so causes a general-protection exception. The TPR is abstracted from the interrupt controller (IC), which prioritizes and manages external interrupt delivery to the processor. The IC can be an external device, such as an APIC or 8259. Typically, the IC provides a priority mechanism similar or identical to the TPR. The IC, however, is considered implementation-dependent with the under-lying priority mechanisms subject to change. CR8, by contrast, is part of the Intel 64 architecture. Software can depend on this definition remaining unchanged.

Figure 11-22 shows the layout of CR8; only the low four bits are used. The remaining 60 bits are reserved and must be written with zeros. Failure to do this causes a general-protection exception.

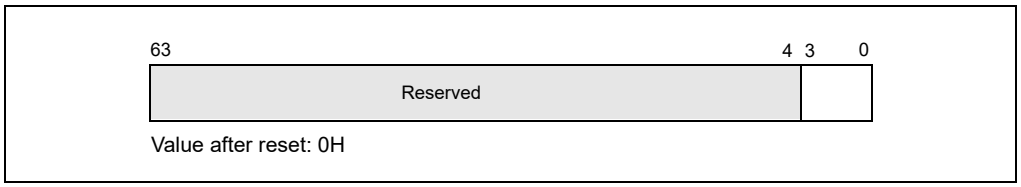


Figure 11-22. CR8 Register

11.8.6.1 Interaction of Task Priorities between CR8 and APIC

The first implementation of Intel 64 architecture includes a local advanced programmable interrupt controller (APIC) that is similar to the APIC used with previous IA-32 processors. Some aspects of the local APIC affect the operation of the architecturally defined task priority register and the programming interface using CR8.

Notable CR8 and APIC interactions are:

- The processor powers up with the local APIC enabled.
- The APIC must be enabled for CR8 to function as the TPR. Writes to CR8 are reflected into the APIC Task Priority Register.
- APIC.TPR[bits 7:4] = CR8[bits 3:0], APIC.TPR[bits 3:0] = 0. A read of CR8 returns a 64-bit value which is the value of TPR[bits 7:4], zero extended to 64 bits.

There are no ordering mechanisms between direct updates of the APIC.TPR and CR8. Operating software should implement either direct APIC TPR updates or CR8 style TPR updates but not mix them. Software can use a serializing instruction (for example, CPUID) to serialize updates between MOV CR8 and stores to the APIC.

11.9 SPURIOUS INTERRUPT

A special situation may occur when a processor raises its task priority to be greater than or equal to the level of the interrupt for which the processor INTR signal is currently being asserted. If at the time the INTA cycle is issued, the interrupt that was to be dispensed has become masked (programmed by software), the local APIC will deliver a spurious-interrupt vector. Dispensing the spurious-interrupt vector does not affect the ISR, so the handler for this vector should return without an EOI.

The vector number for the spurious-interrupt vector is specified in the spurious-interrupt vector register (see Figure 11-23). The functions of the fields in this register are as follows:

Spurious Vector Determines the vector number to be delivered to the processor when the local APIC generates a spurious vector.

(Pentium 4 and Intel Xeon processors.) Bits 0 through 7 of the this field are programmable by software.

(P6 family and Pentium processors). Bits 4 through 7 of the this field are programmable by software, and bits 0 through 3 are hardwired to logical ones. Software writes to bits 0 through 3 have no effect.

APIC Software Enable/Disable

Allows software to temporarily enable (1) or disable (0) the local APIC (see Section 11.4.3, "Enabling or Disabling the Local APIC").

Focus Processor Checking

Determines if focus processor checking is enabled (0) or disabled (1) when using the lowest-priority delivery mode. In Pentium 4 and Intel Xeon processors, this bit is reserved and should be cleared to 0.

Suppress EOI Broadcasts

Determines whether an EOI for a level-triggered interrupt causes EOI messages to be broadcast to the I/O APICs (0) or not (1). See Section 11.8.5. The default value for this bit is 0, indicating that EOI broadcasts are performed. This bit is reserved to 0 if the processor does not support EOI-broadcast suppression.

NOTE

Do not program an LVT or IOAPIC RTE with a spurious vector even if you set the mask bit. A spurious vector ISR does not do an EOI. If for some reason an interrupt is generated by an LVT or RTE entry, the bit in the in-service register will be left set for the spurious vector. This will mask all interrupts at the same or lower priority

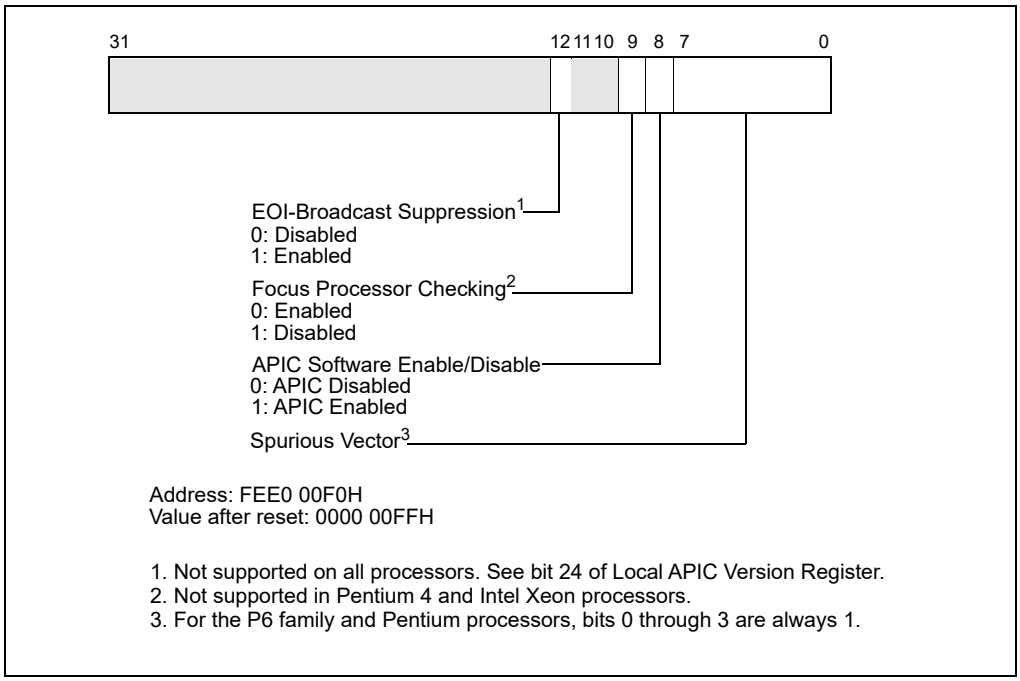


Figure 11-23. Spurious-Interrupt Vector Register (SVR)

11.10 APIC BUS MESSAGE PASSING MECHANISM AND PROTOCOL (P6 FAMILY, PENTIUM PROCESSORS)

The Pentium 4 and Intel Xeon processors pass messages among the local and I/O APICs on the system bus, using the system bus message passing mechanism and protocol.

The P6 family and Pentium processors, pass messages among the local and I/O APICs on the serial APIC bus, as follows. Because only one message can be sent at a time on the APIC bus, the I/O APIC and local APICs employ a "rotating priority" arbitration protocol to gain permission to send a message on the APIC bus. One or more APICs may start sending their messages simultaneously. At the beginning of every message, each APIC presents the type of the message it is sending and its current arbitration priority on the APIC bus. This information is used for arbitration. After each arbitration cycle (within an arbitration round), only the potential winners keep driving the bus.

By the time all arbitration cycles are completed, there will be only one APIC left driving the bus. Once a winner is selected, it is granted exclusive use of the bus, and will continue driving the bus to send its actual message.

After each successfully transmitted message, all APICs increase their arbitration priority by 1. The previous winner (that is, the one that has just successfully transmitted its message) assumes a priority of 0 (lowest). An agent whose arbitration priority was 15 (highest) during arbitration, but did not send a message, adopts the previous winner's arbitration priority, incremented by 1.

Note that the arbitration protocol described above is slightly different if one of the APICs issues a special End-Of-Interrupt (EOI). This high-priority message is granted the bus regardless of its sender's arbitration priority, unless more than one APIC issues an EOI message simultaneously. In the latter case, the APICs sending the EOI messages arbitrate using their arbitration priorities.

If the APICs are set up to use "lowest priority" arbitration (see Section 11.6.2.4, "Lowest Priority Delivery Mode") and multiple APICs are currently executing at the lowest priority (the value in the APR register), the arbitration priorities (unique values in the Arb ID register) are used to break ties. All 8 bits of the APR are used for the lowest priority arbitration.

11.10.1 Bus Message Formats

See Section 11.13, "APIC Bus Message Formats," for a description of bus message formats used to transmit messages on the serial APIC bus.

11.11 MESSAGE SIGNALLED INTERRUPTS

The PCI Local Bus Specification, Rev 2.2 (www.pcisig.com) introduces the concept of message signalled interrupts. As the specification indicates:

> "Message signalled interrupts (MSI) is an optional feature that enables PCI devices to request service by writing a system-specified message to a system-specified address (PCI DWORD memory write transaction). The transaction address specifies the message destination while the transaction data specifies the message. System software is expected to initialize the message destination and message during device configuration, allocating one or more non-shared messages to each MSI capable function."

The capabilities mechanism provided by the PCI Local Bus Specification is used to identify and configure MSI capable PCI devices. Among other fields, this structure contains a Message Data Register and a Message Address Register. To request service, the PCI device function writes the contents of the Message Data Register to the address contained in the Message Address Register (and the Message Upper Address register for 64-bit message addresses).

Section 11.11.1 and Section 11.11.2 provide layout details for the Message Address Register and the Message Data Register. The operation issued by the device is a PCI write command to the Message Address Register with the Message Data Register contents. The operation follows semantic rules as defined for PCI write operations and is a DWORD operation.

11.11.1 Message Address Register Format

The format of the Message Address Register (lower 32-bits) is shown in Figure 11-24.

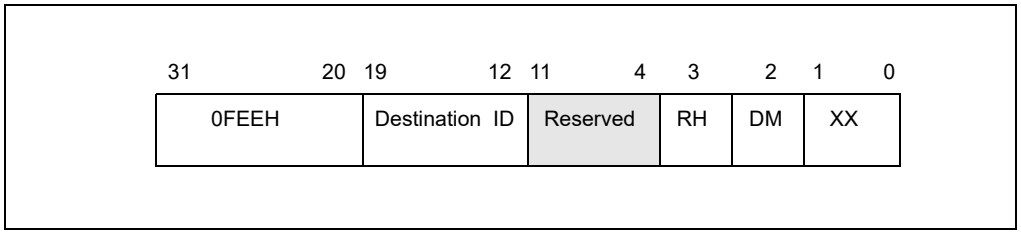


Figure 11-24. Layout of the MSI Message Address Register

Fields in the Message Address Register are as follows:

1. **Bits 31-20** — These bits contain a fixed value for interrupt messages (0FEEH). This value locates interrupts at the 1-MByte area with a base address of 4G – 18M. All accesses to this region are directed as interrupt messages. Care must to be taken to ensure that no other device claims the region as I/O space.
2. **Destination ID** — This field contains an 8-bit destination ID. It identifies the message's target processor(s). The destination ID corresponds to bits 63:56 of the I/O APIC Redirection Table Entry if the IOAPIC is used to dispatch the interrupt to the processor(s).
3. **Redirection hint indication** (RH) — When this bit is set, the message is directed to the processor with the lowest interrupt priority among processors that can receive the interrupt.
    - When RH is 0, the interrupt is directed to the processor listed in the Destination ID field.
    - When RH is 1 and the physical destination mode is used, the Destination ID field must not be set to FFH; it must point to a processor that is present and enabled to receive the interrupt.
    - When RH is 1 and the logical destination mode is active in a system using a flat addressing model, the Destination ID field must be set so that bits set to 1 identify processors that are present and enabled to receive the interrupt.
    - If RH is set to 1 and the logical destination mode is active in a system using cluster addressing model, then Destination ID field must not be set to FFH; the processors identified with this field must be present and enabled to receive the interrupt.
4. **Destination mode (DM)** — This bit indicates whether the Destination ID field should be interpreted as logical or physical APIC ID for delivery of the lowest priority interrupt.
    - If RH is 1 and DM is 0, the Destination ID field is in physical destination mode and only the processor in the system that has the matching APIC ID is considered for delivery of that interrupt (this means no re-direction).
    - If RH is 1 and DM is 1, the Destination ID Field is interpreted as in logical destination mode and the redirection is limited to only those processors that are part of the logical group of processors based on the processor's logical APIC ID and the Destination ID field in the message. The logical group of processors consists of those identified by matching the 8-bit Destination ID with the logical destination identified by the Destination Format Register and the Logical Destination Register in each local APIC. The details are similar to those described in Section 11.6.2, "Determining IPI Destination."
    - If RH is 0, then the DM bit is ignored and the message is sent ahead independent of whether the physical or logical destination mode is used.

11.11.2 Message Data Register Format

The layout of the Message Data Register is shown in Figure 11-25.

Reserved fields are not assumed to be any value. Software must preserve their contents on writes. Other fields in the Message Data Register are described below.

1. **Vector** — This 8-bit field contains the interrupt vector associated with the message. Values range from 010H to 0FEH. Software must guarantee that the field is not programmed with vector 00H to 0FH.
2. **Delivery Mode** — This 3-bit field specifies how the interrupt receipt is handled. Delivery Modes operate only in conjunction with specified Trigger Modes. Correct Trigger Modes must be guaranteed by software. Restrictions are indicated below:
    a. **000B (Fixed Mode)** — Deliver the signal to all the agents listed in the destination. The Trigger Mode for fixed delivery mode can be edge or level.
    b. **001B (Lowest Priority)** — Deliver the signal to the agent that is executing at the lowest priority of all agents listed in the destination field. The trigger mode can be edge or level.
    c. **010B (System Management Interrupt or SMI)** — The delivery mode is edge only. For systems that rely on SMI semantics, the vector field is ignored but must be programmed to all zeroes for future compatibility.

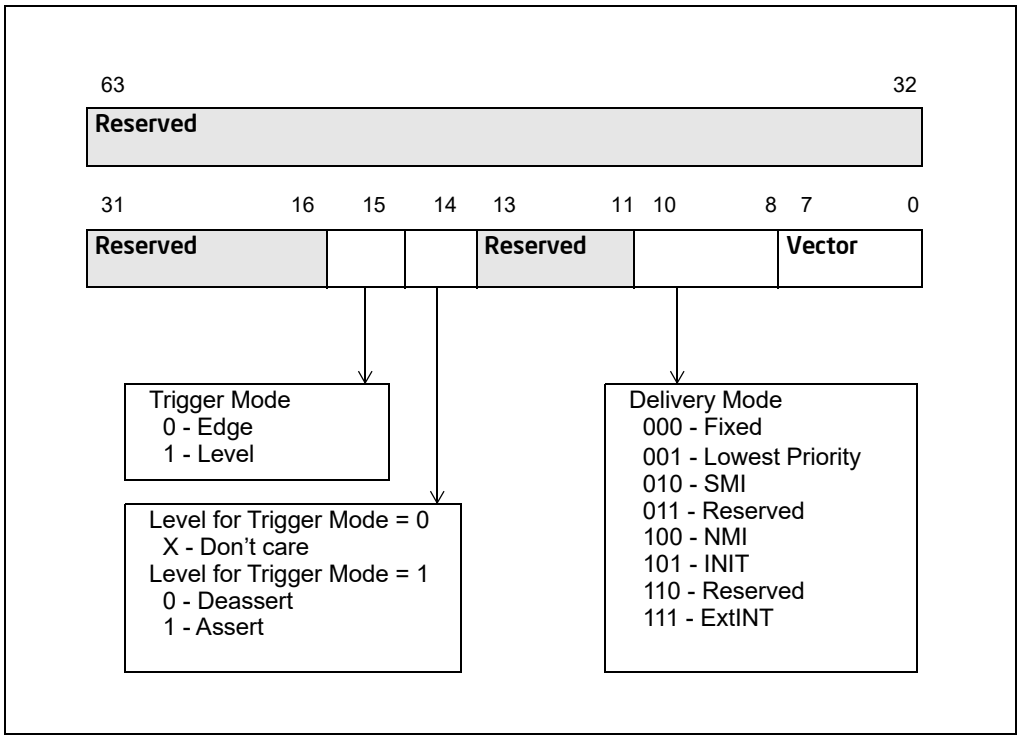


Figure 11-25. Layout of the MSI Message Data Register

d. **100B (NMI)** — Deliver the signal to all the agents listed in the destination field. The vector information is ignored. NMI is an edge triggered interrupt regardless of the Trigger Mode Setting.

e. **101B (INIT)** — Deliver this signal to all the agents listed in the destination field. The vector information is ignored. INIT is an edge triggered interrupt regardless of the Trigger Mode Setting.

f. **111B (ExtINT)** — Deliver the signal to the INTR signal of all agents in the destination field (as an interrupt that originated from an 8259A compatible interrupt controller). The vector is supplied by the INTA cycle issued by the activation of the ExtINT. ExtINT is an edge triggered interrupt.

3. **Level** — Edge triggered interrupt messages are always interpreted as assert messages. For edge triggered interrupts this field is not used. For level triggered interrupts, this bit reflects the state of the interrupt input.

4. **Trigger Mode** — This field indicates the signal type that will trigger a message.

   a. **0** — Indicates edge sensitive.

   b. **1** — Indicates level sensitive.

11.12 EXTENDED XAPIC (X2APIC)

The x2APIC architecture extends the xAPIC architecture (described in Section 11.4) in a backward compatible manner and provides forward extendability for future Intel platform innovations. Specifically, the x2APIC architecture does the following.

- Retains all key elements of compatibility to the xAPIC architecture.
  - — Delivery modes.
  - — Interrupt and processor priorities.
  - — Interrupt sources.
  - — Interrupt destination types.
- Provides extensions to scale processor addressability for both the logical and physical destination modes.

- Adds new features to enhance performance of interrupt delivery.
- Reduces complexity of logical destination mode interrupt delivery on link based platform architectures.
- Uses MSR programming interface to access APIC registers in x2APIC mode instead of memory-mapped interfaces. Memory-mapped interface is supported when operating in xAPIC mode.

11.12.1 Detecting and Enabling x2APIC Mode

Processor support for x2APIC mode can be detected by executing CPUID with EAX=1 and then checking ECX, bit 21 ECX. If CPUID.(EAX=1):ECX.21 is set , the processor supports the x2APIC capability and can be placed into the x2APIC mode.

System software can place the local APIC in the x2APIC mode by setting the x2APIC mode enable bit (bit 10) in the IA32_APIC_BASE MSR at MSR address 01BH. The layout for the IA32_APIC_BASE MSR is shown in Figure 11-26.

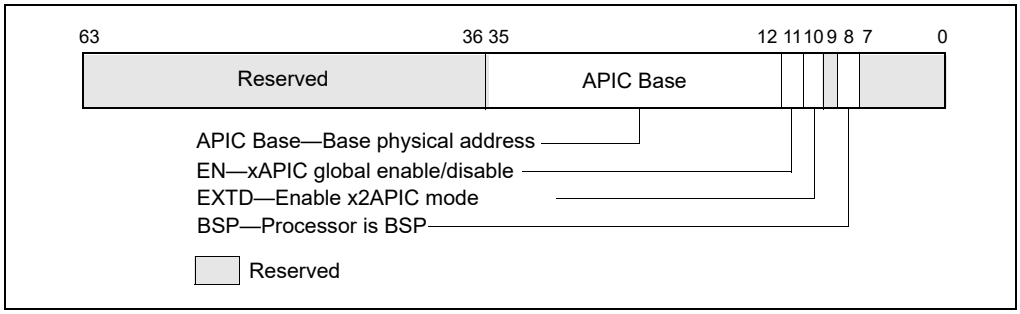


Figure 11-26. IA32_APIC_BASE MSR Supporting x2APIC

Table 11-5, "x2APIC operating mode configurations" describe the possible combinations of the enable bit (EN - bit 11) and the extended mode bit (EXTD - bit 10) in the IA32_APIC_BASE MSR.

Table 11-5. x2APIC Operating Mode Configurations

xAPIC global enable (IA32_APIC_BASE[11])	x2APIC enable (IA32_APIC_BASE[10])	Description
0	0	local APIC is disabled
0	1	Invalid
1	0	local APIC is enabled in xAPIC mode
1	1	local APIC is enabled in x2APIC mode

Once the local APIC has been switched to x2APIC mode (EN = 1, EXTD = 1), switching back to xAPIC mode would require system software to disable the local APIC unit. Specifically, attempting to write a value to the IA32_APIC_BASE MSR that has (EN= 1, EXTD = 0) when the local APIC is enabled and in x2APIC mode causes a general-protection exception. Once bit 10 in IA32_APIC_BASE MSR is set, the only way to leave x2APIC mode using IA32_APIC_BASE would require a WRMSR to set both bit 11 and bit 10 to zero. Section 11.12.5, "x2APIC State Transitions," provides a detailed state diagram for the state transitions allowed for the local APIC.

11.12.1.1 Instructions to Access APIC Registers

In x2APIC mode, system software uses RDMSR and WRMSR to access the APIC registers. The MSR addresses for accessing the x2APIC registers are architecturally defined and specified in Section 11.12.1.2, "x2APIC Register Address Space." Executing the RDMSR instruction with the APIC register address specified in ECX returns the content of bits 0 through 31 of the APIC registers in EAX. Bits 32 through 63 are returned in register EDX - these bits are reserved if the APIC register being read is a 32-bit register. Similarly executing the WRMSR instruction with the APIC register address in ECX, writes bits 0 to 31 of register EAX to bits 0 to 31 of the specified APIC register. If the register is a 64-bit register then bits 0 to 31 of register EDX are written to bits 32 to 63 of the APIC register. The

Interrupt Command Register is the only APIC register that is implemented as a 64-bit MSR. The semantics of handling reserved bits are defined in Section 11.12.1.3, "Reserved Bit Checking."

11.12.1.2 x2APIC Register Address Space

The MSR address range 800H through 8FFH is architecturally reserved and dedicated for accessing APIC registers in x2APIC mode. Table 11-6 lists the APIC registers that are available in x2APIC mode. When appropriate, the table also gives the offset at which each register is available on the page referenced by IA32_APIC_BASE[35:12] in xAPIC mode.

There is a one-to-one mapping between the x2APIC MSRs and the legacy xAPIC register offsets with the following exceptions:

- The Destination Format Register (DFR): The DFR, supported at offset 0E0H in xAPIC mode, is not supported in x2APIC mode. There is no MSR with address 80EH.
- The Interrupt Command Register (ICR): The two 32-bit registers in xAPIC mode (at offsets 300H and 310H) are merged into a single 64-bit MSR in x2APIC mode (with MSR address 830H). There is no MSR with address 831H.
- The SELF IPI register. This register is available only in x2APIC mode at address 83FH. In xAPIC mode, there is no register defined at offset 3F0H.

MSR addresses in the range 800H–8FFH that are not listed in Table 11-6 (including 80EH and 831H) are reserved. Executions of RDMSR and WRMSR that attempt to access such addresses cause general-protection exceptions.

The MSR address space is compressed to allow for future growth. Every 32 bit register on a 128-bit boundary in the legacy MMIO space is mapped to a single MSR in the local x2APIC MSR address space. The upper 32-bits of all x2APIC MSRs (except for the ICR) are reserved.

Table 11-6. Local APIC Register Address Map Supported by x2APIC

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
802H	020H	Local APIC ID register	Read-only[1]	See Section 11.12.5.1 for initial values.
803H	030H	Local APIC Version register	Read-only	Same version used in xAPIC mode and x2APIC mode.
808H	080H	Task Priority Register (TPR)	Read/write	Bits 31:8 are reserved.[2]
80AH	0A0H	Processor Priority Register (PPR)	Read-only	
80BH	0B0H	EOI register	Write-only[3]	WRMSR of a non-zero value causes #GP(0).
80DH	0D0H	Logical Destination Register (LDR)	Read-only	Read/write in xAPIC mode.
80FH	0F0H	Spurious Interrupt Vector Register (SVR)	Read/write	See Section 11.9 for reserved bits.
810H	100H	In-Service Register (ISR); bits 31:0	Read-only	
811H	110H	ISR bits 63:32	Read-only	
812H	120H	ISR bits 95:64	Read-only	
813H	130H	ISR bits 127:96	Read-only	
814H	140H	ISR bits 159:128	Read-only	
815H	150H	ISR bits 191:160	Read-only	
816H	160H	ISR bits 223:192	Read-only	

Table 11-6. Local APIC Register Address Map Supported by x2APIC (Contd.)

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
817H	170H	ISR bits 255:224	Read-only	
818H	180H	Trigger Mode Register (TMR); bits 31:0	Read-only	
819H	190H	TMR bits 63:32	Read-only	
81AH	1A0H	TMR bits 95:64	Read-only	
81BH	1B0H	TMR bits 127:96	Read-only	
81CH	1C0H	TMR bits 159:128	Read-only	
81DH	1D0H	TMR bits 191:160	Read-only	
81EH	1E0H	TMR bits 223:192	Read-only	
81FH	1F0H	TMR bits 255:224	Read-only	
820H	200H	Interrupt Request Register (IRR); bits 31:0	Read-only	
821H	210H	IRR bits 63:32	Read-only	
822H	220H	IRR bits 95:64	Read-only	
823H	230H	IRR bits 127:96	Read-only	
824H	240H	IRR bits 159:128	Read-only	
825H	250H	IRR bits 191:160	Read-only	
826H	260H	IRR bits 223:192	Read-only	
827H	270H	IRR bits 255:224	Read-only	
828H	280H	Error Status Register (ESR)	Read/write	WRMSR of a non-zero value causes #GP(0). See Section 11.5.3.
82FH	2F0H	LVT CMCI register	Read/write	See Figure 11-8 for reserved bits.
830H[4]	300H and 310H	Interrupt Command Register (ICR)	Read/write	See Figure 11-28 for reserved bits
832H	320H	LVT Timer register	Read/write	See Figure 11-8 for reserved bits.
833H	330H	LVT Thermal Sensor register	Read/write	See Figure 11-8 for reserved bits.
834H	340H	LVT Performance Monitoring register	Read/write	See Figure 11-8 for reserved bits.
835H	350H	LVT LINT0 register	Read/write	See Figure 11-8 for reserved bits.
836H	360H	LVT LINT1 register	Read/write	See Figure 11-8 for reserved bits.
837H	370H	LVT Error register	Read/write	See Figure 11-8 for reserved bits.
838H	380H	Initial Count register (for Timer)	Read/write	
839H	390H	Current Count register (for Timer)	Read-only	
83EH	3E0H	Divide Configuration Register (DCR; for Timer)	Read/write	See Figure 11-10 for reserved bits.
83FH	Not available	SELF IPI[5]	Write-only	Available only in x2APIC mode.

NOTES:

1. WRMSR causes #GP(0) for read-only registers.

2. WRMSR causes #GP(0) for attempts to set a reserved bit to 1 in a read/write register (including bits 63:32 of each register).
3. RDMSR causes #GP(0) for write-only registers.
4. MSR 831H is reserved; read/write operations cause general-protection exceptions. The contents of the APIC register at MMIO offset 310H are accessible in x2APIC mode through the MSR at address 830H.
5. SELF IPI register is supported only in x2APIC mode.

11.12.1.3 Reserved Bit Checking

Section 11.12.1.2 and Table 11-6 specifies the reserved bit definitions for the APIC registers in x2APIC mode. Non-zero writes (by WRMSR instruction) to reserved bits to these registers will raise a general protection fault exception while reads return zeros (RsvdZ semantics).

In x2APIC mode, the local APIC ID register is increased to 32 bits wide. This enables $2^{32}-1$ processors to be addressable in physical destination mode. This 32-bit value is referred to as "x2APIC ID". A processor implementation may choose to support less than 32 bits in its hardware. System software should be agnostic to the actual number of bits that are implemented. All non-implemented bits will return zeros on reads by software.

The APIC ID value of FFFF_FFFFH and the highest value corresponding to the implemented bit-width of the local APIC ID register in the system are reserved and cannot be assigned to any logical processor.

In x2APIC mode, the local APIC ID register is a read-only register to system software and will be initialized by hardware. It is accessed via the RDMSR instruction reading the MSR at address 0802H.

Each logical processor in the system (including clusters with a communication fabric) must be configured with an unique x2APIC ID to avoid collisions of x2APIC IDs. On DP and high-end MP processors targeted to specific market segments and depending on the system configuration, it is possible that logical processors in different and "unconnected" clusters power up initialized with overlapping x2APIC IDs. In these configurations, a model-specific means may be provided in those product segments to enable BIOS and/or platform firmware to re-configure the x2APIC IDs in some clusters to provide for unique and non-overlapping system wide IDs before configuring the disconnected components into a single system.

11.12.2 x2APIC Register Availability

The local APIC registers can be accessed via the MSR interface only when the local APIC has been switched to the x2APIC mode as described in Section 11.12.1. Accessing any APIC register in the MSR address range 0800H through 08FFH via RDMSR or WRMSR when the local APIC is not in x2APIC mode causes a general-protection exception. In x2APIC mode, the memory mapped interface is not available and any access to the MMIO interface will behave similar to that of a legacy xAPIC in globally disabled state. Table 11-7 provides the interactions between the legacy & extended modes and the legacy and register interfaces.

Table 11-7. MSR/MMIO Interface of a Local x2APIC in Different Modes of Operation

	MMIO Interface	MSR Interface
xAPIC mode	Available	General-protection exception
x2APIC mode	Behavior identical to xAPIC in globally disabled state	Available

11.12.3 MSR Access in x2APIC Mode

To allow for efficient access to the APIC registers in x2APIC mode, the serializing semantics of WRMSR are relaxed when writing to the APIC registers. Thus, system software should not use "WRMSR to APIC registers in x2APIC mode" as a serializing instruction. Read and write accesses to the APIC registers will occur in program order. A WRMSR to an APIC register may complete before all preceding stores are globally visible; software can prevent this by inserting a serializing instruction or the sequence MFENCE;LFENCE before the WRMSR.

The RDMSR instruction is not serializing and this behavior is unchanged when reading APIC registers in x2APIC mode. System software accessing the APIC registers using the RDMSR instruction should not expect a serializing behavior. (Note: The MMIO-based xAPIC interface is mapped by system software as an un-cached region. Consequently, read/writes to the xAPIC-MMIO interface have serializing semantics in the xAPIC mode.)

11.12.4 VM-Exit Controls for MSRs and x2APIC Registers

The VMX architecture allows a VMM to specify lists of MSRs to be loaded or stored on VMX transitions using the VMX-transition MSR areas (see VM-exit MSR-store address field, VM-exit MSR-load address field, and VM-entry MSR-load address field in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C).

The X2APIC MSRs cannot to be loaded and stored on VMX transitions. A VMX transition fails if the VMM has specified that the transition should access any MSRs in the address range from 0000_0800H to 0000_08FFH (the range used for accessing the X2APIC registers). Specifically, processing of an 128-bit entry in any of the VMX-transition MSR areas fails if bits 31:0 of that entry (represented as ENTRY_LOW_DW) satisfies the expression: "ENTRY_LOW_DW & FFFFF800H = 00000800H". Such a failure causes an associated VM entry to fail (by reloading host state) and causes an associated VM exit to lead to VMX abort.

11.12.5 x2APIC State Transitions

This section provides a detailed description of the x2APIC states of a local x2APIC unit, transitions between these states as well as interactions of these states with INIT and reset.

11.12.5.1 x2APIC States

The valid states for a local x2APIC unit are listed in Table 11-5.

- APIC disabled: IA32_APIC_BASE[EN]=0 and IA32_APIC_BASE[EXTD]=0.
- xAPIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=0.
- x2APIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=1.
- Invalid: IA32_APIC_BASE[EN]=0 and IA32_APIC_BASE[EXTD]=1.

The state corresponding to EXTD=1 and EN=0 is not valid and it is not possible to get into this state. An execution of WRMSR to the IA32_APIC_BASE_MSR that attempts a transition from a valid state to this invalid state causes a general-protection exception. Figure 11-27 shows the comprehensive state transition diagram for a local x2APIC unit.

On coming out of reset, the local APIC unit is enabled and is in the xAPIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=0. The APIC registers are initialized as follows.

- The local APIC ID is initialized by hardware with a 32 bit ID (x2APIC ID). The lowest 8 bits of the x2APIC ID are the legacy local xAPIC ID, and are stored in the upper 8 bits of the APIC register for access in xAPIC mode.
- The following APIC registers are reset to all zeros for those fields that are defined in the xAPIC mode.
  - IRR, ISR, TMR, ICR, LDR, TPR, Divide Configuration Register (See Section 11.4 through Section 11.6 for details of individual APIC registers).
  - Timer initial count and timer current count registers.
- The LVT registers are reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The Spurious Interrupt Vector Register is initialized to 000000FFH.
- The DFR (available only in xAPIC mode) is reset to all 1s.
- SELF IPI register is reset to zero.

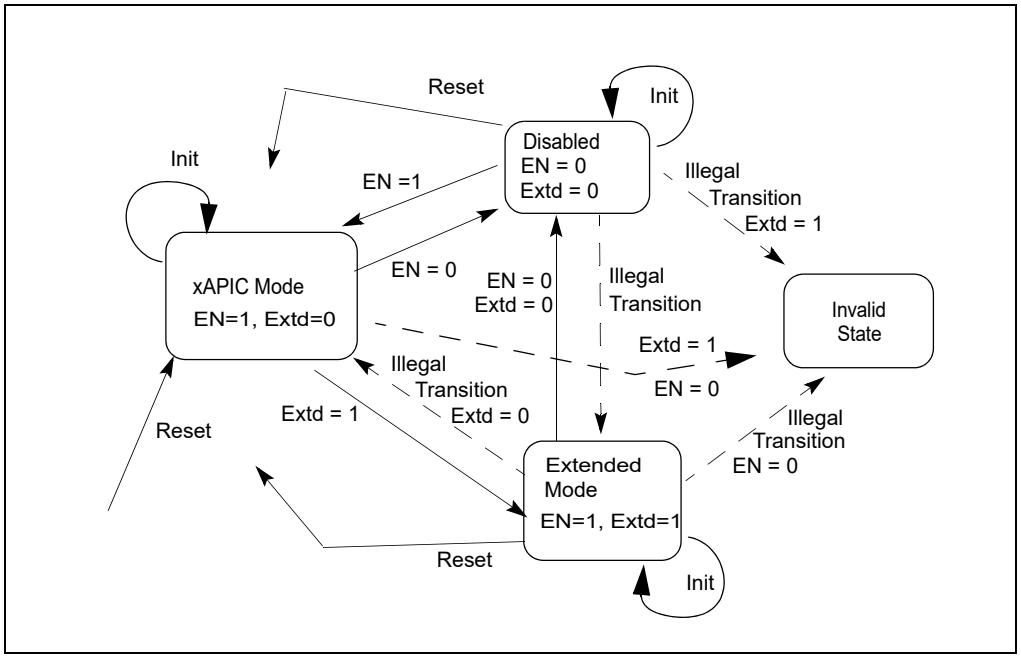


Figure 11-27. Local x2APIC State Transitions with IA32_APIC_BASE, INIT, and Reset

x2APIC After Reset

The valid transitions from the xAPIC mode state are:

- to the x2APIC mode by setting EXT to 1 (resulting EN=1, EXTD= 1). The physical x2APIC ID (see Figure 11-6) is preserved across this transition and the logical x2APIC ID (see Figure 11-29) is initialized by hardware during this transition as documented in Section 11.12.10.2. The state of the extended fields in other APIC registers, which was not initialized at reset, is not architecturally defined across this transition and system software should explicitly initialize those programmable APIC registers.
- to the disabled state by setting EN to 0 (resulting EN=0, EXTD= 0).

The result of an INIT in the xAPIC state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is preserved (the 8-bit xAPIC ID is in the upper 8 bits of the APIC ID register). All the other APIC registers are initialized as a result of INIT.

A reset in this state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is initialized as described in Section 11.12.5.1. All the other APIC registers are initialized described in Section 11.12.5.1.

x2APIC Transitions From x2APIC Mode

From the x2APIC mode, the only valid x2APIC transition using IA32_APIC_BASE is to the state where the x2APIC is disabled by setting EN to 0 and EXTD to 0. The x2APIC ID (32 bits) and the legacy local xAPIC ID (8 bits) are preserved across this transition. A transition from the x2APIC mode to xAPIC mode is not valid, and the corresponding WRMSR to the IA32_APIC_BASE MSR causes a general-protection exception.

A reset in this state places the x2APIC in xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 11.12.5.1.

An INIT in this state keeps the x2APIC in the x2APIC mode. The state of the local APIC ID register is preserved (all 32 bits). However, all the other APIC registers are initialized as a result of the INIT transition.

x2APIC Transitions From Disabled Mode

From the disabled state, the only valid x2APIC transition using IA32_APIC_BASE is to the xAPIC mode (EN= 1, EXTD = 0). Thus the only means to transition from x2APIC mode to xAPIC mode is a two-step process:

- first transition from x2APIC mode to local APIC disabled mode (EN= 0, EXTD = 0),
- followed by another transition from disabled mode to xAPIC mode (EN= 1, EXTD= 0).

Consequently, all the APIC register states in the x2APIC, except for the x2APIC ID (32 bits), are not preserved across mode transitions.

A reset in the disabled state places the x2APIC in the xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 11.12.5.1.

An INIT in the disabled state keeps the x2APIC in the disabled state.

State Changes From xAPIC Mode to x2APIC Mode

After APIC register states have been initialized by software in xAPIC mode, a transition from xAPIC mode to x2APIC mode does not affect most of the APIC register states, except the following:

- The Logical Destination Register is not preserved.
- Any APIC ID value written to the memory-mapped local APIC ID register is not preserved.
- The high half of the Interrupt Command Register is not preserved.

11.12.6 Routing of Device Interrupts in x2APIC Mode

The x2APIC architecture is intended to work with all existing IOxAPIC units as well as all PCI and PCI Express (PCIe) devices that support the capability for message-signaled interrupts (MSI). Support for x2APIC modifies only the following:

- the local APIC units;
- the interconnects joining IOxAPIC units to the local APIC units; and
- the interconnects joining MSI-capable PCI and PCIe devices to the local APIC units.

No modifications are required to MSI-capable PCI and PCIe devices. Similarly, no modifications are required to IOxAPIC units. This is made possible through use of the interrupt-remapping architecture specified in the Intel® Virtualization Technology for Directed I/O Specification, Revision 1.3 and/or later versions, for the routing of interrupts from MSI-capable devices to local APIC units operating in x2APIC mode.

11.12.7 Initialization by System Software

Routing of device interrupts to local APIC units operating in x2APIC mode requires use of the interrupt-remapping architecture specified in the Intel® Virtualization Technology for Directed I/O Specification (Revision 1.3 and/or later versions). Because of this, BIOS must enumerate support for and software must enable this interrupt remapping with Extended Interrupt Mode Enabled before it enabling x2APIC mode in the local APIC units.

The ACPI interfaces for the x2APIC are described in Section 5.2, "ACPI System Description Tables," of the Advanced Configuration and Power Interface Specification, Revision 4.0a (http://www.acpi.info/spec.htm). The default behavior for BIOS is to pass the control to the operating system with the local x2APICs in xAPIC mode if all APIC IDs reported by CPUID.0BH:EDX are less than 255, and in x2APIC mode if there are any logical processor reporting an APIC ID of 255 or greater.

11.12.8 CPUID Extensions And Topology Enumeration

For Intel 64 and IA-32 processors that support x2APIC, a value of 1 reported by CPUID.01H:ECX[21] indicates that the processor supports x2APIC and the extended topology enumeration leaf (CPUID.0BH).

The extended topology enumeration leaf can be accessed by executing CPUID with EAX = 0BH. Processors that do not support x2APIC may support CPUID leaf 0BH. Software can detect the availability of the 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 or 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. If available, the extended topology enumeration leaf is the preferred mechanism for enumerating topology. The presence of CPUID leaf 0BH in a processor does not guarantee support for 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.

The extended topology enumeration leaf is intended to assist software with enumerating processor topology on systems that requires 32-bit x2APIC IDs to address individual logical processors. Details of CPUID leaf 0BH can be found in the reference pages of CPUID in Chapter 3 of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

Processor topology enumeration algorithm for processors supporting the extended topology enumeration leaf of CPUID and processors that do not support CPUID leaf 0BH are treated in Section 9.9.4, "Algorithm for Three-Domain Mappings of APIC_ID."

11.12.8.1 Consistency of APIC IDs and CPUID

The consistency of physical x2APIC ID in MSR 802H in x2APIC mode and the 32-bit value returned in CPUID.0BH:EDX is facilitated by processor hardware.

CPUID.0BH:EDX will report the full 32 bit ID, in xAPIC and x2APIC mode. This allows BIOS to determine if a system has processors with IDs exceeding the 8-bit initial APIC ID limit (CPUID.01H:EBX[31:24]). Initial APIC ID (CPUID.01H:EBX[31:24]) is always equal to CPUID.0BH:EDX[7:0].

If the values of CPUID.0BH:EDX reported by all logical processors in a system are less than 255, BIOS can transfer control to OS in xAPIC mode.

If the values of CPUID.0BH:EDX reported by some logical processors in a system are greater than or equal to 255, BIOS must support two options to hand off to OS.

- If BIOS enables logical processors with x2APIC IDs greater than 255, then it should enable x2APIC in the Boot Strap Processor (BSP) and all Application Processors (AP) before passing control to the OS. Applications requiring processor topology information must use OS provided services based on x2APIC IDs or CPUID.0BH leaf.
- If a BIOS transfers control to OS in xAPIC mode, then the BIOS must ensure that only logical processors with CPUID.0BH.EDX value less than 255 are enabled. BIOS initialization on all logical processors with CPUID.0B.EDX values greater than or equal to 255 must (a) disable APIC and execute CLI in each logical processor, and (b) leave these logical processor in the lowest power state so that these processors do not respond to INIT IPI during OS boot. The BSP and all the enabled logical processor operate in xAPIC mode after BIOS passed control to OS. Application requiring processor topology information can use OS provided legacy services based on 8-bit initial APIC IDs or legacy topology information from CPUID.01H and CPUID 04H leaves. Even if the BIOS passes control in xAPIC mode, an OS can switch the processors to x2APIC mode later. BIOS SMM handler should always read the APIC_BASE_MSR, determine the APIC mode and use the corresponding access method.

11.12.9 ICR Operation in x2APIC Mode

In x2APIC mode, the layout of the Interrupt Command Register is shown in Figure 11-28. The lower 32 bits of ICR in x2APIC mode is identical to the lower half of the ICR in xAPIC mode, except the Delivery Status bit is removed since it is not needed in x2APIC mode. The destination ID field is expanded to 32 bits in x2APIC mode.

To send an IPI using the ICR, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. Self IPIs can also be sent using the SELF IPI register (see Section 11.12.11).

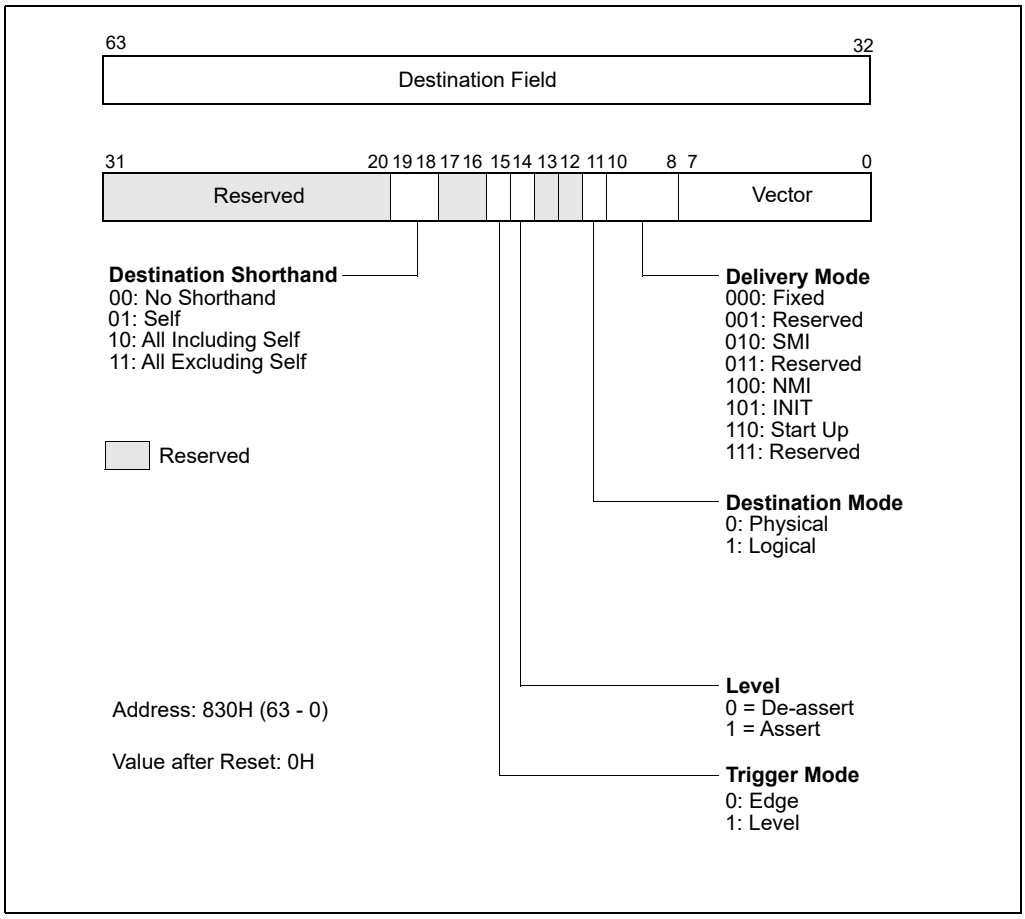


Figure 11-28. Interrupt Command Register (ICR) in x2APIC Mode

A single MSR write to the Interrupt Command Register is required for dispatching an interrupt in x2APIC mode. With the removal of the Delivery Status bit, system software no longer has a reason to read the ICR. It remains readable only to aid in debugging; however, software should not assume the value returned by reading the ICR is the last written value.

A destination ID value of FFFF_FFFFH is used for broadcast of interrupts in both logical destination and physical destination modes.

11.12.10 Determining IPI Destination in x2APIC Mode

11.12.10.1 Logical Destination Mode in x2APIC Mode

In x2APIC mode, the Logical Destination Register (LDR) is increased to 32 bits wide. It is a read-only register to system software. This 32-bit value is referred to as "logical x2APIC ID". System software accesses this register via the RDMSR instruction reading the MSR at address 80DH. Figure 11-29 provides the layout of the Logical Destination Register in x2APIC mode.

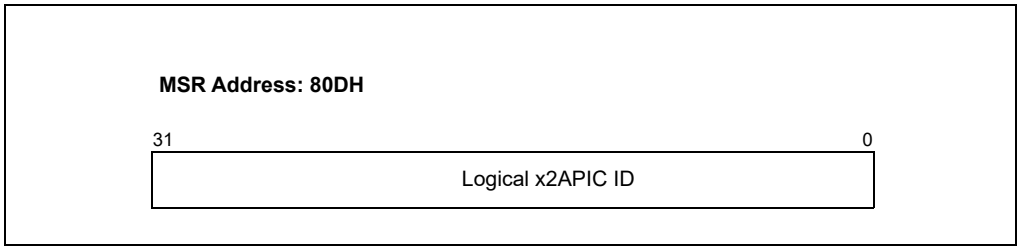


Figure 11-29. Logical Destination Register in x2APIC Mode

In the xAPIC mode, the Destination Format Register (DFR) through the MMIO interface determines the choice of a flat logical mode or a clustered logical mode. Flat logical mode is not supported in the x2APIC mode. Hence the Destination Format Register (DFR) is eliminated in x2APIC mode.

The 32-bit logical x2APIC ID field of LDR is partitioned into two sub-fields:

- Cluster ID (LDR[31:16]): is the address of the destination cluster
- Logical ID (LDR[15:0]): defines a logical ID of the individual local x2APIC within the cluster specified by LDR[31:16].

This layout enables 2^16-1 clusters each with up to 16 unique logical IDs - effectively providing an addressability of ((2^20) - 16) processors in logical destination mode.

It is likely that processor implementations may choose to support less than 16 bits of the cluster ID or less than 16-bits of the Logical ID in the Logical Destination Register. However system software should be agnostic to the number of bits implemented in the cluster ID and logical ID sub-fields. The x2APIC hardware initialization will ensure that the appropriately initialized logical x2APIC IDs are available to system software and reads of non-implemented bits return zero. This is a read-only register that software must read to determine the logical x2APIC ID of the processor. Specifically, software can apply a 16-bit mask to the lowest 16 bits of the logical x2APIC ID to identify the logical address of a processor within a cluster without needing to know the number of implemented bits in cluster ID and Logical ID sub-fields. Similarly, software can create a message destination address for cluster model, by bit-Oring the Logical X2APIC ID (31:0) of processors that have matching Cluster ID(31:16).

To enable cluster ID assignment in a fashion that matches the system topology characteristics and to enable efficient routing of logical mode lowest priority device interrupts in link based platform interconnects, the LDR are initialized by hardware based on the value of x2APIC ID upon x2APIC state transitions. Details of this initialization are provided in Section 11.12.10.2.

11.12.10.2 Deriving Logical x2APIC ID from the Local x2APIC ID

In x2APIC mode, the 32-bit logical x2APIC ID, which can be read from LDR, is derived from the 32-bit local x2APIC ID. Specifically, the 16-bit logical ID sub-field is derived by shifting 1 by the lowest 4 bits of the x2APIC ID, i.e., Logical ID = 1 « x2APIC ID[3:0]. The remaining bits of the x2APIC ID then form the cluster ID portion of the logical x2APIC ID:

```
Logical x2APIC ID = [(x2APIC ID[19:4] « 16) | (1 « x2APIC ID[3:0])]
```

The use of the lowest 4 bits in the x2APIC ID implies that at least 16 APIC IDs are reserved for logical processors within a socket in multi-socket configurations. If more than 16 APIC IDS are reserved for logical processors in a socket/package then multiple cluster IDs can exist within the package.

The LDR initialization occurs whenever the x2APIC mode is enabled (see Section 11.12.5).

11.12.11 SELF IPI Register

SELF IPIs are used extensively by some system software. The x2APIC architecture introduces a new register interface. This new register is dedicated to the purpose of sending self-IPIs with the intent of enabling a highly optimized path for sending self-IPIs.

Figure 11-30 provides the layout of the SELF IPI register. System software only specifies the vector associated with the interrupt to be sent. The semantics of sending a self-IPI via the SELF IPI register are identical to sending a self targeted edge triggered fixed interrupt with the specified vector. Specifically the semantics are identical to the following settings for an inter-processor interrupt sent via the ICR - Destination Shorthand (ICR[19:18] = 01 (Self)), Trigger Mode (ICR[15] = 0 (Edge)), Delivery Mode (ICR[10:8] = 000 (Fixed)), Vector (ICR[7:0] = Vector).

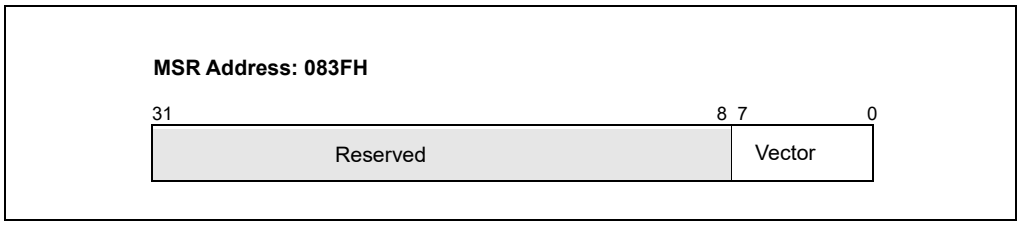


Figure 11-30. SELF IPI register

The SELF IPI register is a write-only register. A RDMSR instruction with address of the SELF IPI register causes a general-protection exception.

The handling and prioritization of a self-IPI sent via the SELF IPI register is architecturally identical to that for an IPI sent via the ICR from a legacy xAPIC unit. Specifically the state of the interrupt would be tracked via the Interrupt Request Register (IRR) and In Service Register (ISR) and Trigger Mode Register (TMR) as if it were received from the system bus. Also sending the IPI via the Self Interrupt Register ensures that interrupt is delivered to the processor core. Specifically completion of the WRMSR instruction to the SELF IPI register implies that the interrupt has been logged into the IRR. As expected for edge triggered interrupts, depending on the processor priority and readiness to accept interrupts, it is possible that interrupts sent via the SELF IPI register or via the ICR with identical vectors can be combined.

11.13 APIC BUS MESSAGE FORMATS

This section describes the message formats used when transmitting messages on the serial APIC bus. The information described here pertains only to the Pentium and P6 family processors.

11.13.1 Bus Message Formats

The local and I/O APICs transmit three types of messages on the serial APIC bus: EOI message, short message, and non-focused lowest priority message. The purpose of each type of message and its format are described below.

11.13.2 EOI Message

Local APICs send 14-cycle EOI messages to the I/O APIC to indicate that a level triggered interrupt has been accepted by the processor. This interrupt, in turn, is a result of software writing into the EOI register of the local APIC. Table 11-8 shows the cycles in an EOI message.

Table 11-8. EOI Message (14 Cycles)

Cycle	Bit1	Bit0	
1	1	1	11 = EOI
2	ArbID3	0	Arbitration ID bits 3 through 0

Table 11-8. EOI Message (14 Cycles) (Contd.)

Cycle	Bit1	Bit0	
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	V7	V6	Interrupt vector V7 - V0
7	V5	V4	
8	V3	V2	
9	V1	V0	
10	C	C	Checksum for cycles 6 - 9
11	0	0	
12	A	A	Status Cycle 0
13	A1	A1	Status Cycle 1
14	0	0	Idle

The checksum is computed for cycles 6 through 9. It is a cumulative sum of the 2-bit (Bit1:Bit0) logical data values. The carry out of all but the last addition is added to the sum. If any APIC computes a different checksum than the one appearing on the bus in cycle 10, it signals an error, driving 11 on the APIC bus during cycle 12. In this case, the APICs disregard the message. The sending APIC will receive an appropriate error indication (see Section 11.5.3, "Error Handling") and resend the message. The status cycles are defined in Table 11-11.

11.13.2.1 Short Message

Short messages (21-cycles) are used for sending fixed, NMI, SMI, INIT, start-up, ExtINT, and lowest-priority-with-focus interrupts. Table 11-9 shows the cycles in a short message.

Table 11-9. Short Message (21 Cycles)

Cycle	Bit1	Bit0	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination Mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	

Table 11-9. Short Message (21 Cycles) (Contd.)

Cycle	Bit1	Bit0	
17	C	C	Checksum for cycles 6-16
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	0	0	Idle

If the physical delivery mode is being used, then cycles 15 and 16 represent the APIC ID and cycles 13 and 14 are considered don't care by the receiver. If the logical delivery mode is being used, then cycles 13 through 16 are the 8-bit logical destination field.

For shorthands of "all-incl-self" and "all-excl-self," the physical delivery mode and an arbitration priority of 15 (D0:D3 = 1111) are used. The agent sending the message is the only one required to distinguish between the two cases. It does so using internal information.

When using lowest priority delivery with an existing focus processor, the focus processor identifies itself by driving 10 during cycle 19 and accepts the interrupt. This is an indication to other APICs to terminate arbitration. If the focus processor has not been found, the short message is extended on-the-fly to the non-focused lowest-priority message. Note that except for the EOI message, messages generating a checksum or an acceptance error (see Section 11.5.3, "Error Handling") terminate after cycle 21.

11.13.2.2 Non-focused Lowest Priority Message

These 34-cycle messages (see Table 11-10) are used in the lowest priority delivery mode when a focus processor is not present. Cycles 1 through 20 are same as for the short message. If during the status cycle (cycle 19) the state of the (A:A) flags is 10B, a focus processor has been identified, and the short message format is used (see Table 11-9). If the (A:A) flags are set to 00B, lowest priority arbitration is started and the 34-cycles of the non-focused lowest priority message are competed. For other combinations of status flags, refer to Section 11.13.2.3, "APIC Bus Status Cycles."

Table 11-10. Non-Focused Lowest Priority Message (34 Cycles)

Cycle	Bit0	Bit1	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	

Table 11-10. Non-Focused Lowest Priority Message (34 Cycles) (Contd.)

Cycle	Bit0	Bit1	
17	C	C	Checksum for cycles 6-16
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	P7	0	P7 - P0 = Inverted Processor Priority
22	P6	0	
23	P5	0	
24	P4	0	
25	P3	0	
26	P2	0	
27	P1	0	
28	P0	0	
29	ArbID3	0	Arbitration ID 3 -0
30	ArbID2	0	
31	ArbID1	0	
32	ArbID0	0	
33	A2	A2	Status Cycle
34	0	0	Idle

Cycles 21 through 28 are used to arbitrate for the lowest priority processor. The processors participating in the arbitration drive their inverted processor priority on the bus. Only the local APICs having free interrupt slots participate in the lowest priority arbitration. If no such APIC exists, the message will be rejected, requiring it to be tried at a later time.

Cycles 29 through 32 are also used for arbitration in case two or more processors have the same lowest priority. In the lowest priority delivery mode, all combinations of errors in cycle 33 (A2 A2) will set the "accept error" bit in the error status register (see Figure 11-9). Arbitration priority update is performed in cycle 20, and is not affected by errors detected in cycle 33. Only the local APIC that wins in the lowest priority arbitration, drives cycle 33. An error in cycle 33 will force the sender to resend the message.

11.13.2.3 APIC Bus Status Cycles

Certain cycles within an APIC bus message are status cycles. During these cycles the status flags (A:A) and (A1:A1) are examined. Table 11-11 shows how these status flags are interpreted, depending on the current delivery mode and existence of a focus processor.

Table 11-11. APIC Bus Status Cycles Interpretation

Delivery Mode	A Status	A1 Status	A2 Status	Update ArbID and Cycle#	Message Length	Retry
EOI	00: CS_OK	10: Accept	XX:	Yes, 13	14 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 13	14 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	14 Cycle	Yes
	11: CS_Error	XX:	XX:	No	14 Cycle	Yes
	10: Error	XX:	XX:	No	14 Cycle	Yes
	01: Error	XX:	XX:	No	14 Cycle	Yes
Fixed	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
NMI, SMI, INIT, ExtINT, Start-Up	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
Lowest	00: CS_OK, NoFocus	11: Do Lowest	10: Accept	Yes, 20	34 Cycle	No
	00: CS_OK, NoFocus	11: Do Lowest	11: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	11: Do Lowest	0X: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	10: End and Retry	XX:	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	0X: Error	XX:	No	34 Cycle	Yes
	10: CS_OK, Focus	XX:	XX:	Yes, 20	34 Cycle	No
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes

10. Updates to Chapter 16, Volume 3B

Change bars and violet text show changes to Chapter 16 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 16.9.2.4, "Request (RRRR) Sub-Field," to add two new entries to Table 16-13, "Encoding of Request (RRRR) Sub-Field," for page walk and EPT page walk request types.
- Updated Section 16.9.3.2, "Architecturally Defined SRAR Errors," to add four new architecturally defined SRAR errors. The details on the four new architecturally defined SRAR errors were added to Table 16-19, "MCA Compound Error Code Encoding for SRAR Errors." Table 16-20, "IA32_MCi_STATUS Values for All Defined SRAR Errors," was also updated.

CHAPTER 16
MACHINE-CHECK ARCHITECTURE

This chapter describes the machine-check architecture and machine-check exception mechanism found in the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors. See Chapter 6, "Interrupt 18—Machine-Check Exception (#MC)," for more information on machine-check exceptions. A brief description of the Pentium processor's machine check capability is also given.

Additionally, a signaling mechanism for software to respond to hardware corrected machine check error is covered.

16.1 MACHINE-CHECK ARCHITECTURE

The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors implement a machine-check architecture that provides a mechanism for detecting and reporting hardware (machine) errors, such as: system bus errors, ECC errors, parity errors, cache errors, and TLB errors. It consists of a set of model-specific registers (MSRs) that are used to set up machine checking and additional banks of MSRs used for recording errors that are detected.

The processor signals the detection of an uncorrected machine-check error by generating a machine-check exception (#MC), which is an abort class exception. The implementation of the machine-check architecture does not ordinarily permit the processor to be restarted reliably after generating a machine-check exception. However, the machine-check-exception handler can collect information about the machine-check error from the machine-check MSRs.

Starting with 45 nm Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH; 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 processor can report information on corrected machine-check errors and deliver a programmable interrupt for software to respond to MC errors, referred to as corrected machine-check error interrupt (CMCI). See Section 16.5 for details.

Intel 64 processors supporting machine-check architecture and CMCI may also support an additional enhancement, namely, support for software recovery from certain uncorrected recoverable machine check errors. See Section 16.6 for details.

16.2 COMPATIBILITY WITH PENTIUM PROCESSOR

The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors support and extend the machine-check exception mechanism introduced in the Pentium processor. The Pentium processor reports the following machine-check errors:

- Data parity errors during read cycles.
- Unsuccessful completion of a bus cycle.

The above errors are reported using the P5_MC_TYPE and P5_MC_ADDR MSRs (implementation specific for the Pentium processor). Use the RDMSR instruction to read these MSRs. See Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, for the addresses.

The machine-check error reporting mechanism that Pentium processors use is similar to that used in Pentium 4, Intel Xeon, Intel Atom, and P6 family processors. When an error is detected, it is recorded in P5_MC_TYPE and P5_MC_ADDR; the processor then generates a machine-check exception (#MC).

See Section 16.3.3, "Mapping of the Pentium Processor Machine-Check Errors to the Machine-Check Architecture," and Section 16.10.2, "Pentium Processor Machine-Check Exception Handling," for information on compatibility between machine-check code written to run on the Pentium processors and code written to run on P6 family processors.

16.3 MACHINE-CHECK MSRS

Machine check MSRs in the Pentium 4, Intel Atom, Intel Xeon, and P6 family processors consist of a set of global control and status registers and several error-reporting register banks. See Figure 16-1.

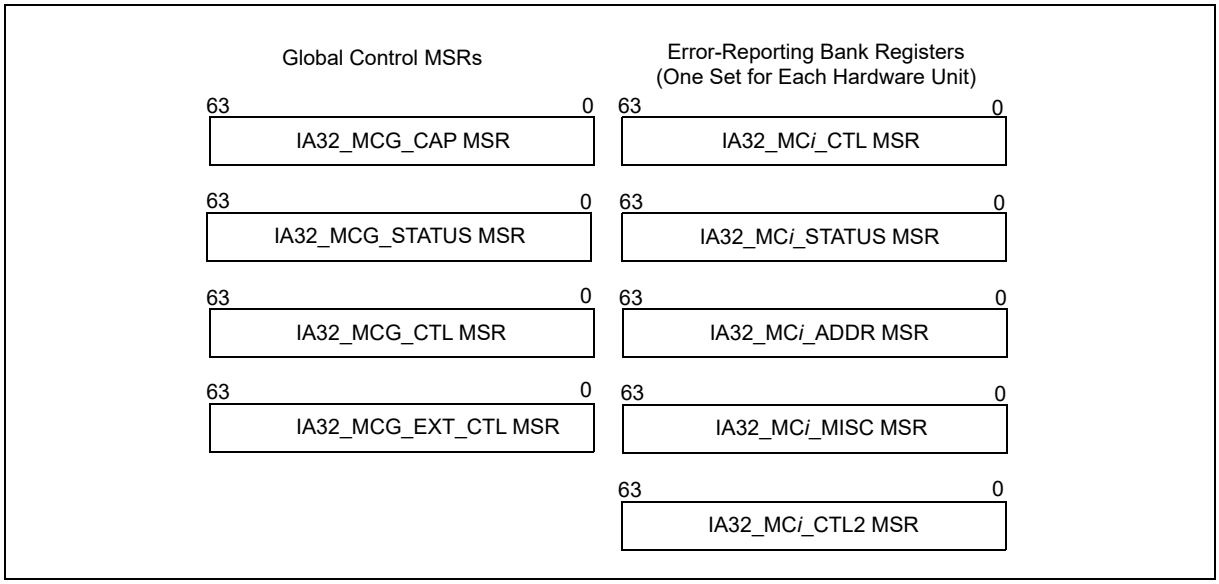


Figure 16-1. Machine-Check MSRs

Each error-reporting bank is associated with a specific hardware unit (or group of hardware units) in the processor. Use RDMSR and WRMSR to read and to write these registers.

16.3.1 Machine-Check Global Control MSRs

The machine-check global control MSRs include the IA32_MCG_CAP, IA32_MCG_STATUS, and optionally IA32_MCG_CTL and IA32_MCG_EXT_CTL. See Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, for the addresses of these registers.

16.3.1.1 IA32_MCG_CAP MSR

The IA32_MCG_CAP MSR is a read-only register that provides information about the machine-check architecture of the processor. Figure 16-2 shows the layout of the register.

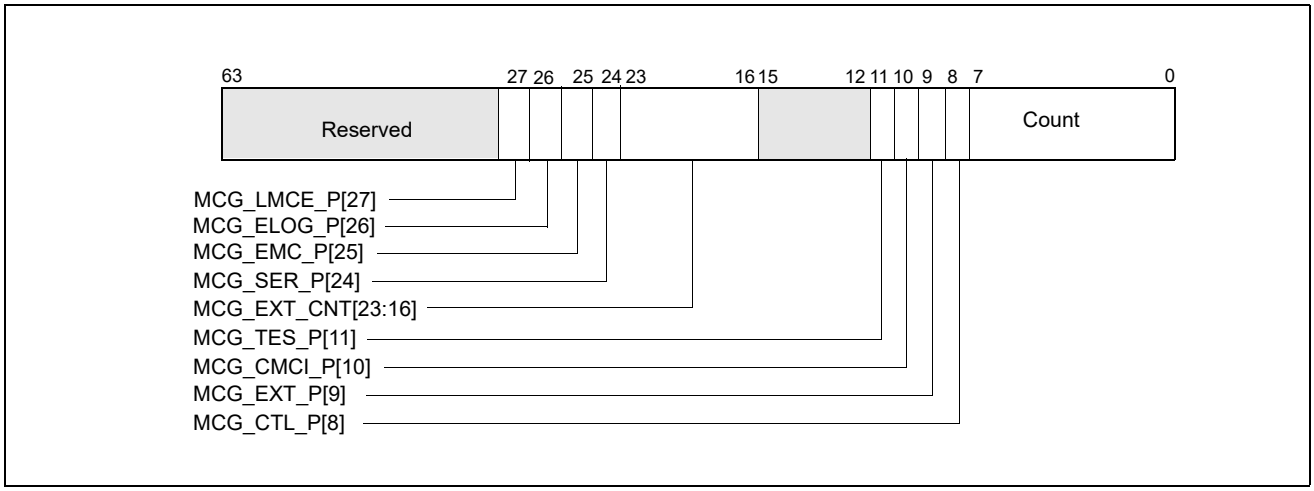


Figure 16-2. IA32_MCG_CAP Register

Where:

- **Count field, bits 7:0** — Indicates the number of hardware unit error-reporting banks available in a particular processor implementation.
- **MCG_CTL_P (control MSR present) flag, bit 8** — Indicates that the processor implements the IA32_MCG_CTL MSR when set; this register is absent when clear.
- **MCG_EXT_P (extended MSRs present) flag, bit 9** — Indicates that the processor implements the extended machine-check state registers found starting at MSR address 180H; these registers are absent when clear.
- **MCG_CMCI_P (Corrected MC error counting/signaling extension present) flag, bit 10** — Indicates (when set) that extended state and associated MSRs necessary to support the reporting of an interrupt on a corrected MC error event and/or count threshold of corrected MC errors, is present. When this bit is set, it does not imply this feature is supported across all banks. Software should check the availability of the necessary logic on a bank by bank basis when using this signaling capability (i.e., bit 30 settable in individual IA32_MCi_CTL2 register).
- **MCG_TES_P (threshold-based error status present) flag, bit 11** — Indicates (when set) that bits 56:53 of the IA32_MCi_STATUS MSR are part of the architectural space. Bits 56:55 are reserved, and bits 54:53 are used to report threshold-based error status. Note that when MCG_TES_P is not set, bits 56:53 of the IA32_MCi_STATUS MSR are model-specific.
- **MCG_EXT_CNT, bits 23:16** — Indicates the number of extended machine-check state registers present. This field is meaningful only when the MCG_EXT_P flag is set.
- **MCG_SER_P (software error recovery support present) flag, bit 24** — Indicates (when set) that the processor supports software error recovery (see Section 16.6), and IA32_MCi_STATUS MSR bits 56:55 are used to report the signaling of uncorrected recoverable errors and whether software must take recovery actions for uncorrected errors. Note that when MCG_TES_P is not set, bits 56:53 of the IA32_MCi_STATUS MSR are model-specific. If MCG_TES_P is set but MCG_SER_P is not set, bits 56:55 are reserved.
- **MCG_EMC_P (Enhanced Machine Check Capability) flag, bit 25** — Indicates (when set) that the processor supports enhanced machine check capabilities for firmware first signaling.
- **MCG_ELOG_P (extended error logging) flag, bit 26** — Indicates (when set) 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.

  For additional information about extended error logging interface, see https://cdrdv2.intel.com/v1/dl/getContent/671064.
- **MCG_LMCE_P (local machine check exception) flag, bit 27** — Indicates (when set) that the following interfaces are present:

— an extended state LMCE_S (located in bit 3 of IA32_MCG_STATUS), and

— the IA32_MCG_EXT_CTL MSR, necessary to support Local Machine Check Exception (LMCE).

A non-zero MCG_LMCE_P indicates that, when LMCE is enabled as described in Section 16.3.1.5, some machine check errors may be delivered to only a single logical processor.

The effect of writing to the IA32_MCG_CAP MSR is undefined.

16.3.1.2 IA32_MCG_STATUS MSR

The IA32_MCG_STATUS MSR describes the current state of the processor after a machine-check exception has occurred (see Figure 16-3).

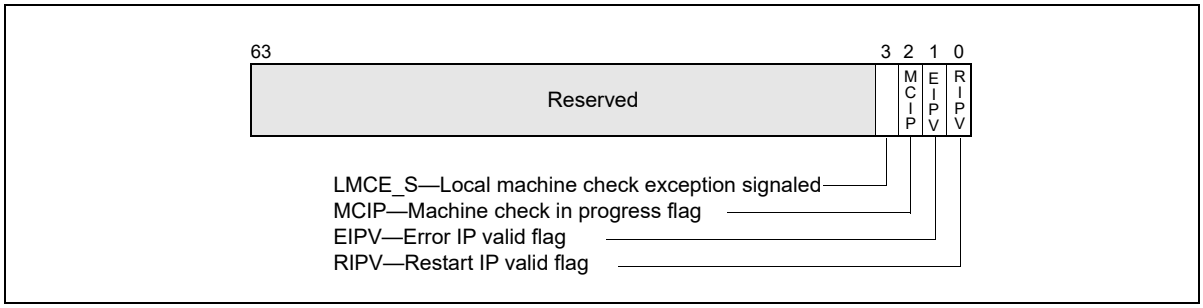


Figure 16-3. IA32_MCG_STATUS Register

Where:

- **RIPV (restart IP valid) flag, bit 0** — Indicates (when set) that program execution can be restarted reliably at the instruction pointed to by the instruction pointer pushed on the stack when the machine-check exception is generated. When clear, the program cannot be reliably restarted at the pushed instruction pointer.
- **EIPV (error IP valid) flag, bit 1** — Indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.
- **MCIP (machine check in progress) flag, bit 2** — Indicates (when set) that a machine-check exception was generated. Software can set or clear this flag. The occurrence of a second Machine-Check Event while MCIP is set will cause the processor to enter a shutdown state. For information on processor behavior in the shutdown state, please refer to the description in Chapter 6, "Interrupt and Exception Handling": "Interrupt 8—Double Fault Exception (#DF)".
- **LMCE_S (local machine check exception signaled), bit 3** — Indicates (when set) that a local machine-check exception was generated. This indicates that the current machine-check event was delivered to only this logical processor.

Bits 63:04 in the IA32_MCG_STATUS MSR are reserved. An attempt to write to the IA32_MCG_STATUS MSR's reserved bits with any value other than 0 results in #GP.

16.3.1.3 IA32_MCG_CTL MSR

The IA32_MCG_CTL MSR is present if the capability flag MCG_CTL_P is set in the IA32_MCG_CAP MSR.

IA32_MCG_CTL controls the reporting of machine-check exceptions. If present, writing 1s to this register enables machine-check features and writing all 0s disables machine-check features. All other values are undefined and/or implementation specific.

16.3.1.4 IA32_MCG_EXT_CTL MSR

The IA32_MCG_EXT_CTL MSR is present if the capability flag MCG_LMCE_P is set in the IA32_MCG_CAP MSR.

IA32_MCG_EXT_CTL.LMCE_EN (bit 0) allows the processor to signal some MCEs to only a single logical processor in the system.

If MCG_LMCE_P is not set in IA32_MCG_CAP, or platform software has not enabled LMCE by setting IA32_FEATURE_CONTROL.LMCE_ENABLED (bit 20), any attempt to write or read IA32_MCG_EXT_CTL will result in #GP.

The IA32_MCG_EXT_CTL MSR is cleared on RESET.

Figure 16-4 shows the layout of the IA32_MCG_EXT_CTL register

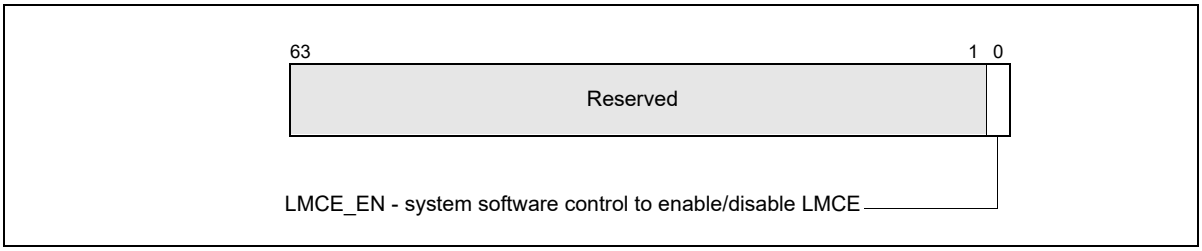


Figure 16-4. IA32_MCG_EXT_CTL Register

where

- **LMCE_EN (local machine check exception enable) flag, bit 0** - System software sets this to allow hardware to signal some MCEs to only a single logical processor. System software can set LMCE_EN only if the platform software has configured IA32_FEATURE_CONTROL as described in Section 16.3.1.5.

16.3.1.5 Enabling Local Machine Check

The intended usage of LMCE requires proper configuration by both platform software and system software. Platform software can turn LMCE on by setting bit 20 (LMCE_ENABLED) in IA32_FEATURE_CONTROL MSR (MSR address 3AH).

System software must ensure that both IA32_FEATURE_CONTROL.Lock (bit 0)and IA32_FEATURE_CONTROL.LMCE_ENABLED (bit 20) are set before attempting to set IA32_MCG_EXT_CTL.LMCE_EN (bit 0). When system software has enabled LMCE, then hardware will determine if a particular error can be delivered only to a single logical processor. Software should make no assumptions about the type of error that hardware can choose to deliver as LMCE. The severity and override rules stay the same as described in Table 16-8 to determine the recovery actions.

16.3.2 Error-Reporting Register Banks

Each error-reporting register bank can contain the IA32_MCi_CTL, IA32_MCi_STATUS, IA32_MCi_ADDR, and IA32_MCi_MISC MSRs. The number of reporting banks is indicated by bits [7:0] of IA32_MCG_CAP MSR (address 0179H). The first error-reporting register (IA32_MC0_CTL) always starts at address 400H.

See Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, for addresses of the error-reporting registers in the Pentium 4, Intel Atom, and Intel Xeon processors; and for addresses of the error-reporting registers P6 family processors.

16.3.2.1 IA32_MCi_CTL MSRs

The IA32_MC*i*_CTL MSR controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units). Each of the 64 flags (EE*j*) represents a potential error. Setting an EE*j* flag enables signaling #MC of the associated error and clearing it disables signaling of the error. Error logging happens regardless of the setting of these bits. The processor drops writes to bits that are not implemented. Figure 16-5 shows the bit fields of IA32_MC*i*_CTL.

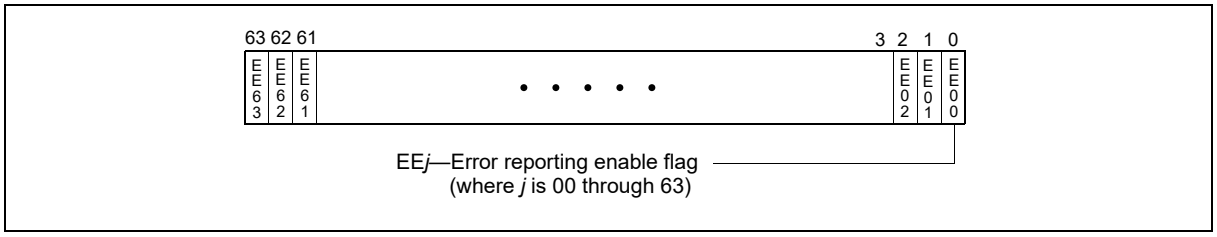


Figure 16-5. IA32_MC*i*_CTL Register

NOTE

For P6 family processors, processors based on Intel Core microarchitecture (excluding those on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH and onward): the operating system or executive software must not modify the contents of the IA32_MC0_CTL MSR. This MSR is internally aliased to the EBL_CR_POWERON MSR and controls platform-specific error handling features. System specific firmware (the BIOS) is responsible for the appropriate initialization of the IA32_MC0_CTL MSR. P6 family processors only allow the writing of all 1s or all 0s to the IA32_MC*i*_CTL MSR.

16.3.2.2 IA32_MCi_STATUS MSRS

Each IA32_MCi_STATUS MSR contains information related to a machine-check error if its VAL (valid) flag is set (see Figure 16-6). Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.

NOTE

Figure 16-6 depicts the IA32_MCi_STATUS MSR when IA32_MCG_CAP[24] = 1, IA32_MCG_CAP[11] = 1 and IA32_MCG_CAP[10] = 1. When IA32_MCG_CAP[24] = 0 and IA32_MCG_CAP[11] = 1, bits 56:55 is reserved and bits 54:53 for threshold-based error reporting. When IA32_MCG_CAP[11] = 0, bits 56:53 are part of the "Other Information" field. The use of bits 54:53 for threshold-based error reporting began with Intel Core Duo processors, and is currently used for cache memory. See Section 16.4, "Enhanced Cache Error reporting," for more information. When IA32_MCG_CAP[10] = 0, bits 52:38 are part of the "Other Information" field. The use of bits 52:38 for corrected MC error count is introduced with Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH.

Where:

- **MCA (machine-check architecture) error code field, bits 15:0** — 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 IA-32 processors that implement the machine-check architecture. See Section 16.9, "Interpreting the MCA Error Codes," and Chapter 17, "Interpreting Machine Check Error Codes," for information on machine-check error codes.
- **Model-specific error code field, bits 31:16** — Specifies the model-specific error code that uniquely identifies the machine-check error condition detected. The model-specific error codes may differ among IA-32 processors for the same machine-check error condition. See Chapter 17, "Interpreting Machine Check Error Codes," for information on model-specific error codes.
- **Reserved, Error Status, and Other Information fields, bits 56:32** —
  - If IA32_MCG_CAP.MCG_EMC_P[bit 25] is 0, bits 37:32 contain "Other Information" that is implementation-specific and is not part of the machine-check architecture.
  - If IA32_MCG_CAP.MCG_EMC_P is 1, "Other Information" is in bits 36:32. If bit 37 is 0, system firmware has not changed the contents of IA32_MCi_STATUS. If bit 37 is 1, system firmware may have edited the contents of IA32_MCi_STATUS.
  - If IA32_MCG_CAP.MCG_CMCI_P[bit 10] is 0, bits 52:38 also contain "Other Information" (in the same sense as bits 37:32).

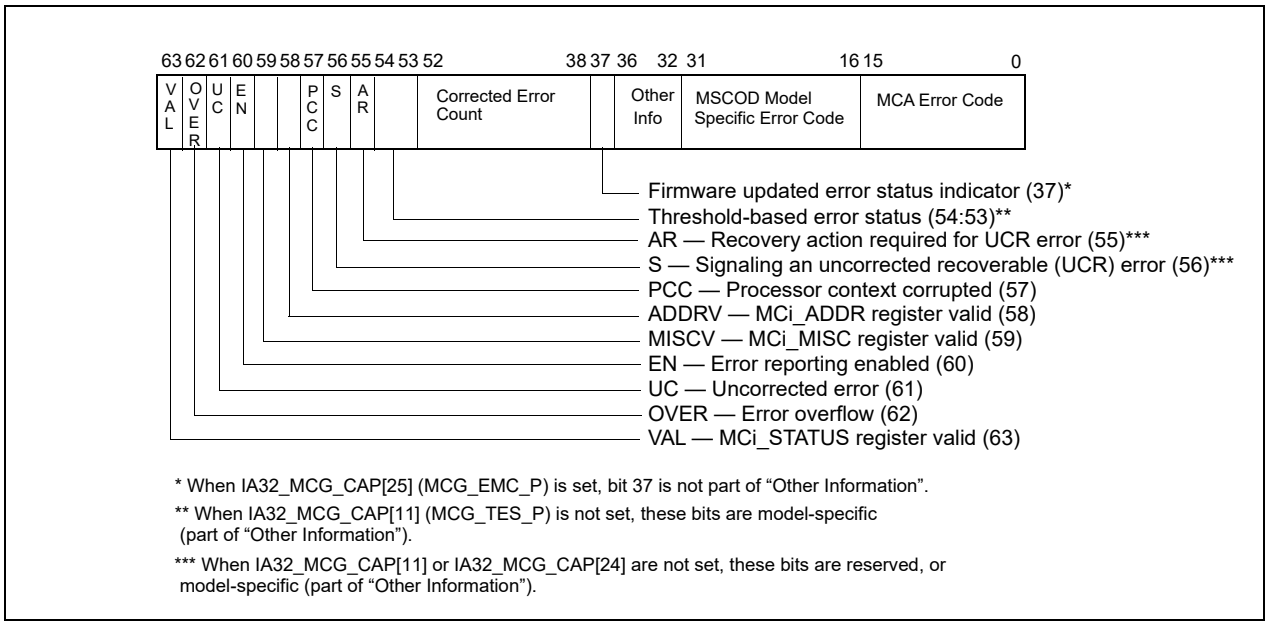


Figure 16-6. IA32_MC*i*_STATUS Register

- If IA32_MCG_CAP[10] is 1, bits 52:38 are architectural (not model-specific). In this case, bits 52:38 reports the value of a 15 bit counter that increments each time a corrected error is observed by the MCA recording bank. This count value will continue to increment until cleared by software. The most significant bit, 52, is a sticky count overflow bit.
- If IA32_MCG_CAP[11] is 0, bits 56:53 also contain "Other Information" (in the same sense).
- If IA32_MCG_CAP[11] is 1, bits 56:53 are architectural (not model-specific). In this case, bits 56:53 have the following functionality:
  - If IA32_MCG_CAP[24] is 0, bits 56:55 are reserved.
  - If IA32_MCG_CAP[24] is 1, bits 56:55 are defined as follows:
  - S (Signaling) flag, bit 56 - Signals the reporting of UCR errors in this MC bank. See Section 16.6.2 for additional details.
  - AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. See Section 16.6.2 for additional details.
  - If the UC bit (Figure 16-6) is 1, bits 54:53 are undefined.
  - If the UC bit (Figure 16-6) is 0, bits 54:53 indicate the status of the hardware structure that reported the threshold-based error. See Table 16-1.

Table 16-1. Bits 54:53 in IA32_MCi_STATUS MSRs when IA32_MCG_CAP[11] = 1 and UC = 0

Bits 54:53	Meaning
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). For more information, see Section 16.4, "Enhanced Cache Error reporting."
10	**Yellow** - Status tracking is provided for the structure posting the event; the current status is yellow (above threshold). For more information, see Section 16.4, "Enhanced Cache Error reporting."
11	Reserved

- **PCC (processor context corrupt) flag, bit 57** — Indicates (when set) 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, and software may be able to restart. When system software supports recovery, consult Section 16.10.4, "Machine-Check Software Handler Guidelines for Error Recovery," for additional rules that apply.
- **ADDRV (IA32_MC*i*_ADDR register valid) flag, bit 58** — Indicates (when set) that the IA32_MCi_ADDR register contains the address where the error occurred (see Section 16.3.2.3, "IA32_MCi_ADDR MSRs"). When clear, this flag indicates that the IA32_MCi_ADDR register is either not implemented or does not contain the address where the error occurred. Do not read these registers if they are not implemented in the processor.
- **MISCV (IA32_MC*i*_MISC register valid) flag, bit 59** — Indicates (when set) that the IA32_MC*i*_MISC register contains additional information regarding the error. When clear, this flag indicates that the IA32_MC*i*_MISC register is either not implemented or does not contain additional information regarding the error. Do not read these registers if they are not implemented in the processor.
- **EN (error enabled) flag, bit 60** — Indicates (when set) that the error was enabled by the associated EEj bit of the IA32_MC*i*_CTL register.
- **UC (error uncorrected) flag, bit 61** — Indicates (when set) that the processor did not or was not able to correct the error condition. When clear, this flag indicates that the processor was able to correct the error condition.
- **OVER (machine check overflow) flag, bit 62** — Indicates (when set) that a machine-check error occurred while the results of a previous error were still in the error-reporting register bank (that is, the VAL bit was already set in the IA32_MC*i*_STATUS register). The processor sets the OVER flag and software is responsible for clearing it. In general, enabled errors are written over disabled errors, and uncorrected errors are written over corrected errors. Uncorrected errors are not written over previous valid uncorrected errors. When MCG_CMCI_P is set, corrected errors may not set the OVER flag. Software can rely on corrected error count in IA32_MCi_Status[52:38] to determine if any additional corrected errors may have occurred. For more information, see Section 16.3.2.2.1, "Overwrite Rules for Machine Check Overflow."
- **VAL (IA32_MC*i*_STATUS register valid) flag, bit 63** — Indicates (when set) that the information within the IA32_MCi_STATUS register is valid. When this flag is set, the processor follows the rules given for the OVER flag in the IA32_MCi_STATUS register when overwriting previously valid entries. The processor sets the VAL flag and software is responsible for clearing it.

16.3.2.2.1 Overwrite Rules for Machine Check Overflow

Table 16-2 shows the overwrite rules for how to treat a second event if the MC bank already contains a valid log from an earlier event – that is, what to do if the valid bit for an MC bank already is set to 1. When more than one structure posts events in a given bank, these rules specify whether a new event will overwrite a previous posting or not. These rules define a priority for uncorrected (highest priority), yellow, and green/unmonitored (lowest priority) status.

In Table 16-2, the values in the two left-most columns are IA32_MCi_STATUS[54:53].

Table 16-2. Overwrite Rules for Enabled Errors

First Event	Second Event	UC bit	Color	MCA Info
00/green	00/green	0	00/green	either
00/green	yellow	0	yellow	second error
yellow	00/green	0	yellow	first error
yellow	yellow	0	yellow	either
00/green/yellow	UC	1	undefined	second
UC	00/green/yellow	1	undefined	first

If a second event overwrites a previously posted event, the information (as guarded by individual valid bits) in the MCi bank is entirely from the second event. Similarly, if a first event is retained, all of the information previously posted for that event is retained. In general, when the logged error or the recent error is a corrected error, the OVER bit (MCi_Status[62]) may be set to indicate an overflow. When MCG_CMCI_P is set in IA32_MCG_CAP, system software should consult IA32_MCi_STATUS[52:38] to determine if additional corrected errors may have

occurred. Software may re-read IA32_MCi_STATUS, IA32_MCi_ADDR, and IA32_MCi_MISC appropriately to ensure data collected represent the last error logged.

After software polls a posting and clears the register, the valid bit is no longer set and therefore the meaning of the rest of the bits, including the yellow/green/00 status field in bits 54:53, is undefined. The yellow/green indication will only be posted for events associated with monitored structures – otherwise the unmonitored (00) code will be posted in IA32_MC*i*_STATUS[54:53].

16.3.2.3 IA32_MCi_ADDR MSRs

The IA32_MC*i*_ADDR MSR contains the address of the code or data memory location that produced the machine-check error if the ADDRV flag in the IA32_MC*i*_STATUS register is set (see Section 16-7, "IA32_MCi_ADDR MSR"). The IA32_MCi_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MCi_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general protection exception.

The address returned is an offset into a segment, linear address, or physical address. This depends on the error encountered. When these registers are implemented, these registers can be cleared by explicitly writing 0s to these registers. Writing 1s to these registers will cause a general-protection exception. See Figure 16-7.

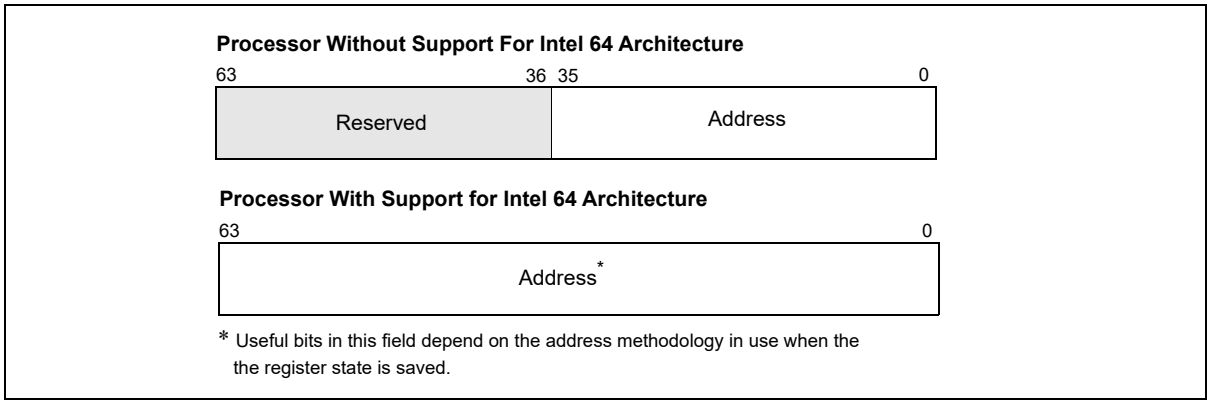


Figure 16-7. IA32_MC*i*_ADDR MSR

16.3.2.4 IA32_MCi_MISC MSRs

The IA32_MC*i*_MISC MSR contains additional information describing the machine-check error if the MISCV flag in the IA32_MC*i*_STATUS register is set. The IA32_MCi_MISC_MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MCi_STATUS register is clear.

When not implemented in the processor, all reads and writes to this MSR will cause a general protection exception. When implemented in a processor, these registers can be cleared by explicitly writing all 0s to them; writing 1s to them causes a general-protection exception to be generated. This register is not implemented in any of the error-reporting register banks for the P6 or Intel Atom family processors.

If both MISCV and IA32_MCG_CAP[24] are set, the IA32_MCi_MISC_MSR is defined according to Figure 16-8 to support software recovery of uncorrected errors (see Section 16.6).

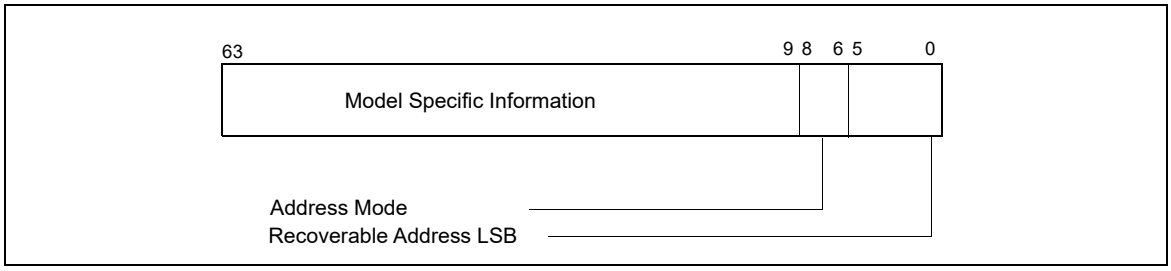


Figure 16-8. UCR Support in IA32_MCi_MISC Register

- Recoverable Address LSB (bits 5:0): The lowest valid recoverable address bit. Indicates the position of the least significant bit (LSB) of the recoverable error address. For example, if the processor logs bits [43:9] of the address, the LSB sub-field in IA32_MCi_MISC is 01001b (9 decimal). For this example, bits [8:0] of the recoverable error address in IA32_MCi_ADDR should be ignored.
- Address Mode (bits 8:6): Address mode for the address logged in IA32_MCi_ADDR. The supported address modes are given in Table 16-3.

Table 16-3. Address Mode in IA32_MCi_MISC[8:6]

IA32_MCi_MISC[8:6] Encoding	Definition
000	Segment Offset
001	Linear Address
010	Physical Address
011	Memory Address
100 to 110	Reserved
111	Generic

- Model Specific Information (bits 63:9): Not architecturally defined.

16.3.2.4.2 IOMCA

Logging and Signaling of errors from PCI Express domain is governed by PCI Express Advanced Error Reporting (AER) architecture. PCI Express architecture divides errors in two categories: Uncorrectable errors and Correctable errors. Uncorrectable errors can further be classified as Fatal or Non-Fatal. Uncorrected IO errors are signaled to the system software either as AER Message Signaled Interrupt (MSI) or via platform specific mechanisms such as NMI. Generally, the signaling mechanism is controlled by BIOS and/or platform firmware. Certain processors support an error handling mode, called IOMCA mode, where Uncorrected PCI Express errors are signaled in the form of machine check exception and logged in machine check banks.

When a processor is in this mode, Uncorrected PCI Express errors are logged in the MCACOD field of the IA32_MCi_STATUS register as Generic I/O error. The corresponding MCA error code is defined in Table 15-8. IA32_MCi_Status [15:0] Simple Error Code Encoding. Machine check logging complements and does not replace AER logging that occurs inside the PCI Express hierarchy. The PCI Express Root Complex and Endpoints continue to log the error in accordance with PCI Express AER mechanism. In IOMCA mode, MCi_MISC register in the bank that logged IOMCA can optionally contain information that link the Machine Check logs with the AER logs or proprietary logs. In such a scenario, the machine check handler can utilize the contents of MCi_MISC to locate the next level of error logs corresponding to the same error. Specifically, if MCi_Status.MISCV is 1 and MCACOD is 0x0E0B, MCi_MISC contains the PCI Express address of the Root Complex device containing the AER Logs. Software can consult the header type and class code registers in the Root Complex device's PCIe Configuration space to determine what type of device it is. This Root Complex device can either be a PCI Express Root Port, PCI Express Root Complex Event Collector or a proprietary device.

Errors that originate from PCI Express or Legacy Endpoints are logged in the corresponding Root Port in addition to the generating device. If MISCV=1 and MCi_MISC contains the address of the Root Port or a Root Complex Event collector, software can parse the AER logs to learn more about the error.

If MISCV=1 and MCi_MISC points to a device that is neither a Root Complex Event Collector not a Root Port, software must consult the Vendor ID/Device ID and use device specific knowledge to locate and interpret the error log registers. In some cases, the Root Complex device configuration space may not be accessible to the software and both the Vendor and Device ID read as 0xFFFF.

- The format of MCi_MISC for IOMCA errors is shown in Table 16-4.

Table 16-4. Address Mode in IA32_MCi_MISC[8:6]

63:40	39:32	31:16	15:9	8:6	5:0
RSVD	PCI Express Segment number	PCI Express Requestor ID	RSVD	ADDR MODE[1]	RECOV ADDR LSB[1]

NOTES:

1. Not Applicable if ADDRV=0.

Refer to PCI Express Specification 3.0 for definition of PCI Express Requestor ID and AER architecture. Refer to PCI Firmware Specification 3.0 for an explanation of PCI Ex-press Segment number and how software can access configuration space of a PCI Ex-press device given the segment number and Requestor ID.

16.3.2.5 IA32_MCi_CTL2 MSRs

The IA32_MC*i*_CTL2 MSR provides the programming interface to use corrected MC error signaling capability that is indicated by IA32_MCG_CAP[10] = 1. Software must check for the presence of IA32_MC*i*_CTL2 on a per-bank basis.

When IA32_MCG_CAP[10] = 1, the IA32_MCi_CTL2 MSR for each bank exists, i.e., reads and writes to these MSR are supported. However, signaling interface for corrected MC errors may not be supported in all banks.

The layout of IA32_MC*i*_CTL2 is shown in Figure 16-9.

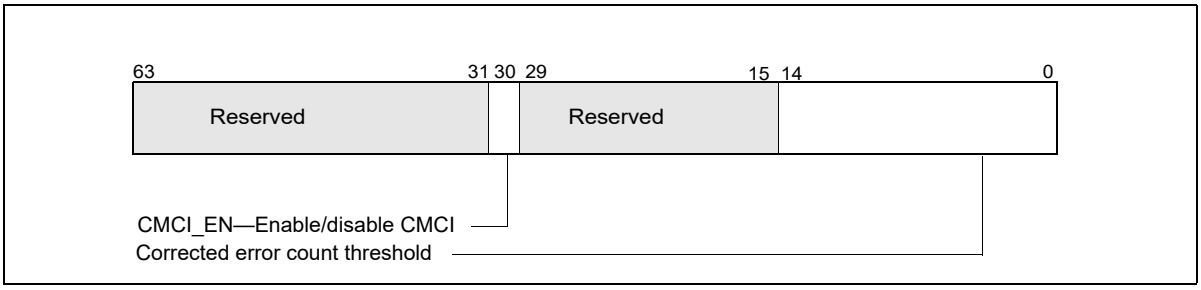


Figure 16-9. IA32_MCi_CTL2 Register

- **Corrected error count threshold, bits 14:0** — Software must initialize this field. The value is compared with the corrected error count field in IA32_MCi_STATUS, bits 38 through 52. An overflow event is signaled to the CMCI LVT entry (see Table 11-1) in the APIC when the count value equals the threshold value. The new LVT entry in the APIC is at 02F0H offset from the APIC_BASE. If CMCI interface is not supported for a particular bank (but IA32_MCG_CAP[10] = 1), this field will always read 0.
- **CMCI_EN (Corrected error interrupt enable/disable/indicator), bits 30** — Software sets this bit to enable the generation of corrected machine-check error interrupt (CMCI). If CMCI interface is not supported for a particular bank (but IA32_MCG_CAP[10] = 1), this bit is writeable but will always return 0 for that bank. This bit also indicates CMCI is supported or not supported in the corresponding bank. See Section 16.5 for details of software detection of CMCI facility.

Some microarchitectural sub-systems that are the source of corrected MC errors may be shared by more than one logical processors. Consequently, the facilities for reporting MC errors and controlling mechanisms may be shared by more than one logical processors. For example, the IA32_MC*i*_CTL2 MSR is shared between logical processors sharing a processor core. Software is responsible to program IA32_MC*i*_CTL2 MSR in a consistent manner with CMCI delivery and usage.

After processor reset, IA32_MC*i*_CTL2 MSRs are zeroed.

16.3.2.6 IA32_MCG Extended Machine Check State MSRs

The Pentium 4 and Intel Xeon processors implement a variable number of extended machine-check state MSRs. The MCG_EXT_P flag in the IA32_MCG_CAP MSR indicates the presence of these extended registers, and the MCG_EXT_CNT field indicates the number of these registers actually implemented. See Section 16.3.1.1, "IA32_MCG_CAP MSR." Also see Table 16-5.

Table 16-5. Extended Machine Check State MSRs in Processors Without Support for Intel® 64 Architecture

MSR	Address	Description
IA32_MCG_EAX	180H	Contains state of the EAX register at the time of the machine-check error.
IA32_MCG_EBX	181H	Contains state of the EBX register at the time of the machine-check error.
IA32_MCG_ECX	182H	Contains state of the ECX register at the time of the machine-check error.
IA32_MCG_EDX	183H	Contains state of the EDX register at the time of the machine-check error.
IA32_MCG_ESI	184H	Contains state of the ESI register at the time of the machine-check error.
IA32_MCG_EDI	185H	Contains state of the EDI register at the time of the machine-check error.
IA32_MCG_EBP	186H	Contains state of the EBP register at the time of the machine-check error.
IA32_MCG_ESP	187H	Contains state of the ESP register at the time of the machine-check error.
IA32_MCG_EFLAGS	188H	Contains state of the EFLAGS register at the time of the machine-check error.
IA32_MCG_EIP	189H	Contains state of the EIP register at the time of the machine-check error.
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.

In processors with support for Intel 64 architecture, 64-bit machine check state MSRs are aliased to the legacy MSRs. In addition, there may be registers beyond IA32_MCG_MISC. These may include up to five reserved MSRs (IA32_MCG_RESERVED[1:5]) and save-state MSRs for registers introduced in 64-bit mode. See Table 16-6.

Table 16-6. Extended Machine Check State MSRs In Processors With Support for Intel® 64 Architecture

MSR	Address	Description
IA32_MCG_RAX	180H	Contains state of the RAX register at the time of the machine-check error.
IA32_MCG_RBX	181H	Contains state of the RBX register at the time of the machine-check error.
IA32_MCG_RCX	182H	Contains state of the RCX register at the time of the machine-check error.
IA32_MCG_RDX	183H	Contains state of the RDX register at the time of the machine-check error.
IA32_MCG_RSI	184H	Contains state of the RSI register at the time of the machine-check error.
IA32_MCG_RDI	185H	Contains state of the RDI register at the time of the machine-check error.
IA32_MCG_RBP	186H	Contains state of the RBP register at the time of the machine-check error.
IA32_MCG_RSP	187H	Contains state of the RSP register at the time of the machine-check error.
IA32_MCG_RFLAGS	188H	Contains state of the RFLAGS register at the time of the machine-check error.
IA32_MCG_RIP	189H	Contains state of the RIP register at the time of the machine-check error.
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.

Table 16-6. Extended Machine Check State MSRs In Processors With Support for Intel® 64 Architecture (Contd.)

MSR	Address	Description
IA32_MCG_ RSERVED[1:5]	18BH-18FH	These registers, if present, are reserved.
IA32_MCG_R8	190H	Contains state of the R8 register at the time of the machine-check error.
IA32_MCG_R9	191H	Contains state of the R9 register at the time of the machine-check error.
IA32_MCG_R10	192H	Contains state of the R10 register at the time of the machine-check error.
IA32_MCG_R11	193H	Contains state of the R11 register at the time of the machine-check error.
IA32_MCG_R12	194H	Contains state of the R12 register at the time of the machine-check error.
IA32_MCG_R13	195H	Contains state of the R13 register at the time of the machine-check error.
IA32_MCG_R14	196H	Contains state of the R14 register at the time of the machine-check error.
IA32_MCG_R15	197H	Contains state of the R15 register at the time of the machine-check error.

When a machine-check error is detected on a Pentium 4 or Intel Xeon processor, the processor saves the state of the general-purpose registers, the R/EFLAGS register, and the R/EIP in these extended machine-check state MSRs. This information can be used by a debugger to analyze the error.

These registers are read/write to zero registers. This means software can read them; but if software writes to them, only all zeros is allowed. If software attempts to write a non-zero value into one of these registers, a general-protection (#GP) exception is generated. These registers are cleared on a hardware reset (power-up or RESET), but maintain their contents following a soft reset (INIT reset).

16.3.3 Mapping of the Pentium Processor Machine-Check Errors to the Machine-Check Architecture

The Pentium processor reports machine-check errors using two registers: P5_MC_TYPE and P5_MC_ADDR. The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors map these registers to the IA32_MC*i*_STATUS and IA32_MC*i*_ADDR in the error-reporting register bank. This bank reports on the same type of external bus errors reported in P5_MC_TYPE and P5_MC_ADDR.

The information in these registers can then be accessed in two ways:

- By reading the IA32_MC*i*_STATUS and IA32_MC*i*_ADDR registers as part of a general machine-check exception handler written for Pentium 4, Intel Atom and P6 family processors.
- By reading the P5_MC_TYPE and P5_MC_ADDR registers using the RDMSR instruction.

The second capability permits a machine-check exception handler written to run on a Pentium processor to be run on a Pentium 4, Intel Xeon, Intel Atom, or P6 family processor. There is a limitation in that information returned by the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors is encoded differently than information returned by the Pentium processor. To run a Pentium processor machine-check exception handler on a Pentium 4, Intel Xeon, Intel Atom, or P6 family processor; the handler must be written to interpret P5_MC_TYPE encodings correctly.

16.4 ENHANCED CACHE ERROR REPORTING

Starting with Intel Core Duo processors, cache error reporting was enhanced. In earlier Intel processors, cache status was based on the number of correction events that occurred in a cache. In the new paradigm, called "threshold-based error status", cache status is based on the number of lines (ECC blocks) in a cache that incur repeated corrections. The threshold is chosen by Intel, based on various factors. If a processor supports threshold-based error status, it sets IA32_MCG_CAP[11] (MCG_TES_P) to 1; if not, to 0.

A processor that supports enhanced cache error reporting contains hardware that tracks the operating status of certain caches and provides an indicator of their "health". The hardware reports a "green" status when the number of lines that incur repeated corrections is at or below a pre-defined threshold, and a "yellow" status when the

number of affected lines exceeds the threshold. Yellow status means that the cache reporting the event is operating correctly, but you should schedule the system for servicing within a few weeks.

Intel recommends that you rely on this mechanism for structures supported by threshold-base error reporting.

The CPU/system/platform response to a yellow event should be less severe than its response to an uncorrected error. An uncorrected error means that a serious error has actually occurred, whereas the yellow condition is a warning that the number of affected lines has exceeded the threshold but is not, in itself, a serious event: the error was corrected and system state was not compromised.

The green/yellow status indicator is not a foolproof early warning for an uncorrected error resulting from the failure of two bits in the same ECC block. Such a failure can occur and cause an uncorrected error before the yellow threshold is reached. However, the chance of an uncorrected error increases as the number of affected lines increases.

16.5 CORRECTED MACHINE CHECK ERROR INTERRUPT

Corrected machine-check error interrupt (CMCI) is an architectural enhancement to the machine-check architecture. It provides capabilities beyond those of threshold-based error reporting (Section 16.4). With threshold-based error reporting, software is limited to use periodic polling to query the status of hardware corrected MC errors. CMCI provides a signaling mechanism to deliver a local interrupt based on threshold values that software can program using the IA32_MCi_CTL2 MSRs.

CMCI is disabled by default. System software is required to enable CMCI for each IA32_MCi bank that support the reporting of hardware corrected errors if IA32_MCG_CAP[10] = 1.

System software use IA32_MCi_CTL2 MSR to enable/disable the CMCI capability for each bank and program threshold values into IA32_MCi_CTL2 MSR. CMCI is not affected by the CR4.MCE bit, and it is not affected by the IA32_MCi_CTL MSRs.

To detect the existence of thresholding for a given bank, software writes only bits 14:0 with the threshold value. If the bits persist, then thresholding is available (and CMCI is available). If the bits are all 0's, then no thresholding exists. To detect that CMCI signaling exists, software writes a 1 to bit 30 of the MCi_CTL2 register. Upon subsequent read, if bit 30 = 0, no CMCI is available for this bank and no corrected or UCNA errors will be reported on this bank. If bit 30 = 1, then CMCI is available and enabled.

16.5.1 CMCI Local APIC Interface

The operation of CMCI is depicted in Figure 16-10.

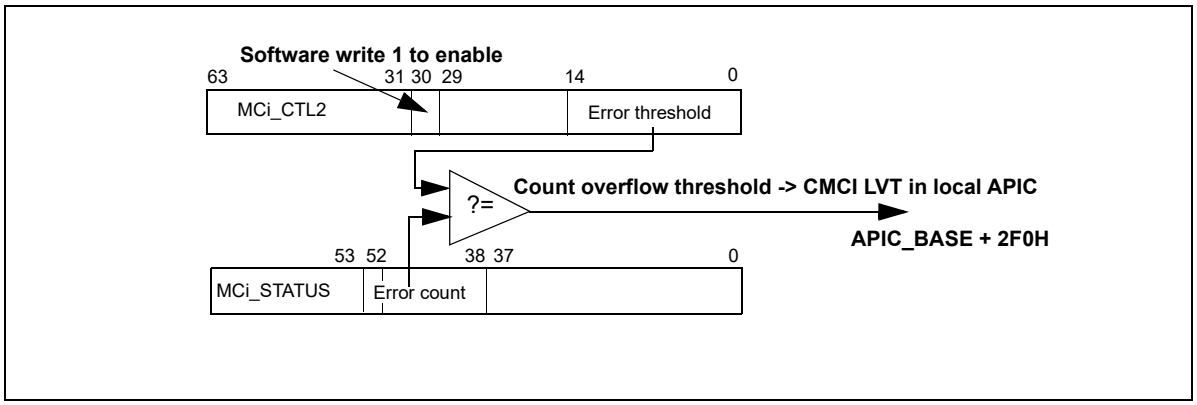


Figure 16-10. CMCI Behavior

CMCI interrupt delivery is configured by writing to the LVT CMCI register entry in the local APIC register space at default address of APIC_BASE + 2F0H. A CMCI interrupt can be delivered to more than one logical processors if multiple logical processors are affected by the associated MC errors. For example, if a corrected bit error in a cache shared by two logical processors caused a CMCI, the interrupt will be delivered to both logical processors sharing

that microarchitectural sub-system. Similarly, package level errors may cause CMCI to be delivered to all logical processors within the package. However, system level errors will not be handled by CMCI.

See Section 11.5.1, "Local Vector Table," for details regarding the LVT CMCI register.

16.5.2 System Software Recommendation for Managing CMCI and Machine Check Resources

System software must enable and manage CMCI, set up interrupt handlers to service CMCI interrupts delivered to affected logical processors, program CMCI LVT entry, and query machine check banks that are shared by more than one logical processors.

This section describes techniques system software can implement to manage CMCI initialization, service CMCI interrupts in a efficient manner to minimize contentions to access shared MSR resources.

16.5.2.1 CMCI Initialization

Although a CMCI interrupt may be delivered to more than one logical processors depending on the nature of the corrected MC error, only one instance of the interrupt service routine needs to perform the necessary service and make queries to the machine-check banks. The following steps describes a technique that limits the amount of work the system has to do in response to a CMCI.

- To provide maximum flexibility, system software should define per-thread data structure for each logical processor to allow equal-opportunity and efficient response to interrupt delivery. Specifically, the per-thread data structure should include a set of per-bank fields to track which machine check bank it needs to access in response to a delivered CMCI interrupt. The number of banks that needs to be tracked is determined by IA32_MCG_CAP[7:0].
- Initialization of per-thread data structure. The initialization of per-thread data structure must be done serially on each logical processor in the system. The sequencing order to start the per-thread initialization between different logical processor is arbitrary. But it must observe the following specific detail to satisfy the shared nature of specific MSR resources:
  a. Each thread initializes its data structure to indicate that it does not own any MC bank registers.
  b. Each thread examines IA32_MCi_CTL2[30] indicator for each bank to determine if another thread has already claimed ownership of that bank.
     - If IA32_MCi_CTL2[30] had been set by another thread. This thread can not own bank *i* and should proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
     - If IA32_MCi_CTL2[30] = 0, proceed to step c.
  c. Check whether writing a 1 into IA32_MCi_CTL2[30] can return with 1 on a subsequent read to determine this bank can support CMCI.
     - If IA32_MCi_CTL2[30] = 0, this bank does not support CMCI. This thread can not own bank *i* and should proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
     - If IA32_MCi_CTL2[30] = 1, modify the per-thread data structure to indicate this thread claims ownership to the MC bank; proceed to initialize the error threshold count (bits 15:0) of that bank as described in Chapter 16, "CMCI Threshold Management". Then proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
- After the thread has examined all of the machine check banks, it sees if it owns any MC banks to service CMCI. If any bank has been claimed by this thread:
  — Ensure that the CMCI interrupt handler has been set up as described in Chapter 16, "CMCI Interrupt Handler".
  — Initialize the CMCI LVT entry, as described in Section 16.5.1, "CMCI Local APIC Interface."
  — Log and clear all of IA32_MCi_Status registers for the banks that this thread owns. This will allow new errors to be logged.

16.5.2.2 CMCI Threshold Management

The Corrected MC error threshold field, IA32_MCi_CTL2[14:0], is architecturally defined. Specifically, all these bits are writable by software, but different processor implementations may choose to implement less than 15 bits as threshold for the overflow comparison with IA32_MCi_STATUS[52:38]. The following describes techniques that software can manage CMCI threshold to be compatible with changes in implementation characteristics:

- Software can set the initial threshold value to 1 by writing 1 to IA32_MCi_CTL2[14:0]. This will cause overflow condition on every corrected MC error and generates a CMCI interrupt.
- To increase the threshold and reduce the frequency of CMCI servicing:
  a. Find the maximum threshold value a given processor implementation supports. The steps are:
     - Write 7FFFH to IA32_MCi_CTL2[14:0],
     - Read back IA32_MCi_CTL2[14:0]; these 15 bits (14:0) contain the maximum threshold supported by the processor.
  b. Increase the threshold to a value below the maximum value discovered using step a.

16.5.2.3 CMCI Interrupt Handler

The following describes techniques system software may consider to implement a CMCI service routine:

- The service routine examines its private per-thread data structure to check which set of MC banks it has ownership. If the thread does not have ownership of a given MC bank, proceed to the next MC bank. Ownership is determined at initialization time which is described in Section 16.5.2.1.

If the thread had claimed ownership to an MC bank, this technique will allow each logical processors to handle corrected MC errors independently and requires no synchronization to access shared MSR resources. Consult Example 16-5 for guidelines on logging when processing CMCI.

16.6 RECOVERY OF UNCORRECTED RECOVERABLE (UCR) ERRORS

Recovery of uncorrected recoverable machine check errors is an enhancement in machine-check architecture. The first processor that supports this feature is 45 nm Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_2EH; 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. This allows system software to perform recovery action on a certain class of uncorrected errors and continue execution.

16.6.1 Detection of Software Error Recovery Support

Software must use bit 24 of IA32_MCG_CAP (MCG_SER_P) to detect the presence of software error recovery support (see Figure 16-2). When IA32_MCG_CAP[24] is set, this indicates that the processor supports software error recovery. When this bit is clear, this indicates that there is no support for error recovery from the processor and the primary responsibility of the machine check handler is logging the machine check error information and shutting down the system.

The new class of architectural MCA errors from which system software can attempt recovery is called Uncorrected Recoverable (UCR) Errors. UCR errors are uncorrected errors that have been detected and signaled but have not corrupted the processor context. For certain UCR errors, this means that once system software has performed a certain recovery action, it is possible to continue execution on this processor. UCR error reporting provides an error containment mechanism for data poisoning. The machine check handler will use the error log information from the error reporting registers to analyze and implement specific error recovery actions for UCR errors.

16.6.2 UCR Error Reporting and Logging

IA32_MCi_STATUS MSR is used for reporting UCR errors and existing corrected or uncorrected errors. The definitions of IA32_MCi_STATUS, including bit fields to identify UCR errors, is shown in Figure 16-6. UCR errors can be

signaled through either the corrected machine check interrupt (CMCI) or machine check exception (MCE) path depending on the type of the UCR error.

When IA32_MCG_CAP[24] is set, a UCR error is indicated by the following bit settings in the IA32_MCi_STATUS register:

- Valid (bit 63) = 1
- UC (bit 61) = 1
- PCC (bit 57) = 0

Additional information from the IA32_MCi_MISC and the IA32_MCi_ADDR registers for the UCR error are available when the ADDRV and the MISCV flags in the IA32_MCi_STATUS register are set (see Section 16.3.2.4). The MCA error code field of the IA32_MCi_STATUS register indicates the type of UCR error. System software can interpret the MCA error code field to analyze and identify the necessary recovery action for the given UCR error.

In addition, the IA32_MCi_STATUS register bit fields, bits 56:55, are defined (see Figure 16-6) to provide additional information to help system software to properly identify the necessary recovery action for the UCR error:

- S (Signaling) flag, bit 56 - Indicates (when set) that a machine check exception was generated for the UCR error reported in this MC bank and system software needs to check the AR flag and the MCA error code fields in the IA32_MCi_STATUS register to identify the necessary recovery action for this error. When the S flag in the IA32_MCi_STATUS register is clear, this UCR error was not signaled via a machine check exception and instead was reported as a corrected machine check (CMC). System software is not required to take any recovery action when the S flag in the IA32_MCi_STATUS register is clear.
- AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. This recovery action must be completed successfully before any additional work is scheduled for this processor. When the RIPV flag in the IA32_MCG_STATUS is clear, an alternative execution stream needs to be provided; when the MCA error code specific recovery specific recovery action cannot be successfully completed, system software must shut down the system. When the AR flag in the IA32_MCi_STATUS register is clear, system software may still take MCA error code specific recovery action but this is optional; system software can safely resume program execution at the instruction pointer saved on the stack from the machine check exception when the RIPV flag in the IA32_MCG_STATUS register is set.

Both the S and the AR flags in the IA32_MCi_STATUS register are defined to be sticky bits, which mean that once set, the processor does not clear them. Only software and good power-on reset can clear the S and the AR-flags. Both the S and the AR flags are only set when the processor reports the UCR errors (MCG_CAP[24] is set).

16.6.3 UCR Error Classification

With the S and AR flag encoding in the IA32_MCi_STATUS register, UCR errors can be classified as:

- Uncorrected no action required (UCNA) - is a UCR error that is not signaled via a machine check exception and, instead, is reported to system software as a corrected machine check error. UCNA errors indicate that some data in the system is corrupted, but the data has not been consumed and the processor state is valid and you may continue execution on this processor. UCNA errors require no action from system software to continue execution. A UCNA error is indicated with UC=1, PCC=0, S=0 and AR=0 in the IA32_MCi_STATUS register.
- Software recoverable action optional (SRAO) - a UCR error is signaled either via a machine check exception or CMCI. System software recovery action is optional and not required to continue execution from this machine check exception. SRAO errors indicate that some data in the system is corrupt, but the data has not been consumed and the processor state is valid. SRAO errors provide the additional error information for system software to perform a recovery action. An SRAO error when signaled as a machine check is indicated with UC=1, PCC=0, S=1, EN=1 and AR=0 in the IA32_MC*i*_STATUS register. In cases when SRAO is signaled via CMCI the error signature is indicated via UC=1, PCC=0, S=0. Recovery actions for SRAO errors are MCA error code specific. The MISCV and the ADDRV flags in the IA32_MC*i*_STATUS register are set when the additional error information is available from the IA32_MC*i*_MISC and the IA32_MC*i*_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MC*i*_STATUS register to identify the specific recovery action for a given SRAO error. If MISCV and ADDRV are not set, it is recommended that no system software error recovery be performed however, system software can resume execution.
- Software recoverable action required (SRAR) - a UCR error that requires system software to take a recovery action on this processor before scheduling another stream of execution on this processor. SRAR errors indicate

that the error was detected and raised at the point of the consumption in the execution flow. An SRAR error is indicated with UC=1, PCC=0, S=1, EN=1 and AR=1 in the IA32_MCi_STATUS register. Recovery actions are MCA error code specific. The MISCV and the ADDRV flags in the IA32_MCi_STATUS register are set when the additional error information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MCi_STATUS register to identify the specific recovery action for a given SRAR error. If MISCV and ADDRV are not set, it is recommended that system software shutdown the system.

Table 16-7 summarizes UCR, corrected, and uncorrected errors.

Table 16-7. MC Error Classifications

Type of Error[1]	UC	EN	PCC	S	AR	Signaling	Software Action	Example
Uncorrected Error (UC)	1	1	1	x	x	MCE	If EN=1, reset the system, else log and OK to keep the system running.	
SRAR	1	1	0	1	1	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, must bugcheck. If OVER=1, reset system, else take specific recovery action.	Cache to processor load error.
SRAO	1	x[2]	0	x[2]	0	MCE/CMC	For known MCACOD, take specific recovery action; For unknown MCACOD, OK to keep the system running.	Patrol scrub and explicit writeback poison errors.
UCNA	1	x	0	0	0	CMC	Log the error and Ok to keep the system running.	Poison detection error.
Corrected Error (CE)	0	x	x	x	x	CMC	Log the error and no corrective action required.	ECC in caches and memory.

NOTES:

1. SRAR, SRAO and UCNA errors are supported by the processor only when IA32_MCG_CAP[24] (MCG_SER_P) is set.
2. EN=1, S=1 when signaled via MCE. EN=x, S=0 when signaled via CMC.

16.6.4 UCR Error Overwrite Rules

In general, the overwrite rules are as follows:

- UCR errors will overwrite corrected errors.
- Uncorrected (PCC=1) errors overwrite UCR (PCC=0) errors.
- UCR errors are not written over previous UCR errors.
- Corrected errors do not write over previous UCR errors.

Regardless of whether the 1st error is retained or the 2nd error is overwritten over the 1st error, the OVER flag in the IA32_MCi_STATUS register will be set to indicate an overflow condition. As the S flag and AR flag in the IA32_MCi_STATUS register are defined to be sticky flags, a second event cannot clear these 2 flags once set, however the MC bank information may be filled in for the 2nd error. The table below shows the overwrite rules and how to treat a second error if the first event is already logged in a MC bank along with the resulting bit setting of the UC, PCC, and AR flags in the IA32_MCi_STATUS register. As UCNA and SRA0 errors do not require recovery action from system software to continue program execution, a system reset by system software is not required unless the AR flag or PCC flag is set for the UCR overflow case (OVER=1, VAL=1, UC=1, PCC=0).

Table 16-8 lists overwrite rules for uncorrected errors, corrected errors, and uncorrected recoverable errors.

Table 16-8. Overwrite Rules for UC, CE, and UCR Errors

First Event	Second Event	UC	PCC	S	AR	MCA Bank	Reset System
CE	UCR	1	0	0 if UCNA, else 1	1 if SRAR, else 0	second	yes, if AR=1
UCR	CE	1	0	0 if UCNA, else 1	1 if SRAR, else 0	first	yes, if AR=1

Table 16-8. Overwrite Rules for UC, CE, and UCR Errors

First Event	Second Event	UC	PCC	S	AR	MCA Bank	Reset System
UCNA	UCNA	1	0	0	0	first	no
UCNA	SRAO	1	0	1	0	first	no
UCNA	SRAR	1	0	1	1	first	yes
SRAO	UCNA	1	0	1	0	first	no
SRAO	SRAO	1	0	1	0	first	no
SRAO	SRAR	1	0	1	1	first	yes
SRAR	UCNA	1	0	1	1	first	yes
SRAR	SRAO	1	0	1	1	first	yes
SRAR	SRAR	1	0	1	1	first	yes
UCR	UC	1	1	undefined	undefined	second	yes
UC	UCR	1	1	undefined	undefined	first	yes

16.7 MACHINE-CHECK AVAILABILITY

The machine-check architecture and machine-check exception (#MC) are model-specific features. Software can execute the CPUID instruction to determine whether a processor implements these features. Following the execution of the CPUID instruction, the settings of the MCA flag (bit 14) and MCE flag (bit 7) in EDX indicate whether the processor implements the machine-check architecture and machine-check exception.

16.8 MACHINE-CHECK INITIALIZATION

To use the processors machine-check architecture, software must initialize the processor to activate the machine-check exception and the error-reporting mechanism.

Example 16-1 gives pseudocode for performing this initialization. This pseudocode checks for the existence of the machine-check architecture and exception; it then enables machine-check exception and the error-reporting register banks. The pseudocode shown is compatible with the Pentium 4, Intel Xeon, Intel Atom, P6 family, and Pentium processors.

Following power up or power cycling, IA32_MC*i*_STATUS registers are not guaranteed to have valid data until after they are initially cleared to zero by software (as shown in the initialization pseudocode in Example 16-1).

Example 16-1. Machine-Check Initialization Pseudocode

```
Check CPUID Feature Flags for MCE and MCA support
IF CPU supports MCE
THEN
    IF CPU supports MCA
    THEN
        IF (IA32_MCG_CAP.MCG_CTL_P = 1)
        (* IA32_MCG_CTL register is present *)
        THEN
            IA32_MCG_CTL ← FFFFFFFFFFFFFFFFH;
            (* enables all MCA features *)
        FI

        IF (IA32_MCG_CAP.MCG_LMCE_P = 1 and IA32_FEATURE_CONTROL.LOCK = 1 and IA32_FEATURE_CONTROL.LMCE_ENABLED = 1)
        (* IA32_MCG_EXT_CTL register is present and platform has enabled LMCE to permit system software to use LMCE *)
        THEN
            IA32_MCG_EXT_CTL ← IA32_MCG_EXT_CTL | 01H;
            (* System software enables LMCE capability for hardware to signal MCE to a single logical processor*)
        FI
```

```
        (* Determine number of error-reporting banks supported *)
        COUNT← IA32_MCG_CAP.Count;
        MAX_BANK_NUMBER ← COUNT - 1;

        IF (Processor Family is 6H and Processor EXTMODEL:MODEL is less than 1AH)
        THEN
            (* Enable logging of all errors except for MC0_CTL register *)
            FOR error-reporting banks (1 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFFH;
            OD

        ELSE
            (* Enable logging of all errors including MC0_CTL register *)
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFFH;
            OD
        FI

        (* BIOS clears all errors only on power-on reset *)
        IF (BIOS detects Power-on reset)
        THEN
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_STATUS ← 0;
            OD
        ELSE
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                (Optional for BIOS and OS) Log valid errors
                (OS only) IA32_MCi_STATUS ← 0;
            OD

        FI
    FI

    Setup the Machine Check Exception (#MC) handler for vector 18 in IDT

    Set the MCE bit (bit 6) in CR4 register to enable Machine-Check Exceptions
FI
```

16.9 INTERPRETING THE MCA ERROR CODES

When the processor detects a machine-check error condition, it writes a 16-bit error code to the MCA error code field of one of the IA32_MC*i*_STATUS registers and sets the VAL (valid) flag in that register. The processor may also write a 16-bit model-specific error code in the IA32_MC*i*_STATUS register depending on the implementation of the machine-check architecture of the processor.

The MCA error codes are architecturally defined for Intel 64 and IA-32 processors. To determine the cause of a machine-check exception, the machine-check exception handler must read the VAL flag for each IA32_MC*i*_STATUS register. If the flag is set, the machine check-exception handler must then read the MCA error code field of the register. It is the encoding of the MCA error code field [15:0] that determines the type of error being reported and not the register bank reporting it.

There are two types of MCA error codes: simple error codes and compound error codes.

16.9.1 Simple Error Codes

Table 16-9 shows the simple error codes. These unique codes indicate global error information.

Table 16-9. IA32_MCi_Status [15:0] Simple Error Code Encoding

Error Code	Binary Encoding	Meaning
No Error	0000 0000 0000 0000	No error has been reported to this bank of error-reporting registers.
Unclassified	0000 0000 0000 0001	This error has not been classified into the MCA error classes.
Microcode ROM Parity Error	0000 0000 0000 0010	Parity error in internal microcode ROM
External Error	0000 0000 0000 0011	The BINIT# from another processor caused this processor to enter machine check.[1]
FRC Error	0000 0000 0000 0100	FRC (functional redundancy check) main/secondary error.
Internal Parity Error	0000 0000 0000 0101	Internal parity error.
SMM Handler Code Access Violation	0000 0000 0000 0110	An attempt was made by the SMM Handler to execute outside the ranges specified by SMRR.
Internal Timer Error	0000 0100 0000 0000	Internal timer error.
I/O Error	0000 1110 0000 1011	generic I/O error.
Internal Unclassified	0000 01xx xxxx xxxx	Internal unclassified errors. [2]

NOTES:

1. BINIT# assertion will cause a machine check exception if the processor (or any processor on the same external bus) has BINIT# observation enabled during power-on configuration (hardware strapping) and if machine check exceptions are enabled (by setting CR4.MCE = 1).
2. At least one X must equal one. Internal unclassified errors have not been classified.

16.9.2 Compound Error Codes

Compound error codes describe errors related to the TLBs, memory, caches, bus and interconnect logic, and internal timer. A set of sub-fields is common to all of compound errors. These sub-fields describe the type of access, level in the cache hierarchy, and type of request. Table 16-10 shows the general form of the compound error codes.

Table 16-10. IA32_MCi_Status [15:0] Compound Error Code Encoding

Type	Form	Interpretation
Generic Cache Hierarchy	000F 0000 0000 11LL	Generic cache hierarchy error
TLB Errors	000F 0000 0001 TTLL	{TT}TLB{LL}_ERR
Memory Controller Errors	000F 0000 1MMM CCCC	{MMM}_CHANNEL{CCCC}_ERR
Cache Hierarchy Errors	000F 0001 RRRR TTLL	{TT}CACHE{LL}_{RRRR}_ERR
Extended Memory Errors	000F 0010 1MMM CCCC	{MMM}_CHANNEL{CCCC}_ERR
Bus and Interconnect Errors	000F 1PPT RRRR IILL	BUS{LL}_{PP}_{RRRR}_{II}_{T}_ERR

The "Interpretation" column in the table indicates the name of a compound error. The name is constructed by substituting mnemonics for the sub-field names given within curly braces. For example, the error code ICACHEL1_RD_ERR is constructed from the form:

{TT}CACHE{LL}_{RRRR}_ERR,
where {TT} is replaced by I, {LL} is replaced by L1, and {RRRR} is replaced by RD.

For more information on the "Form" and "Interpretation" columns, see Section 16.9.2.1, "Correction Report Filtering (F) Bit," through Section 16.9.2.5, "Bus and Interconnect Errors."

16.9.2.1 Correction Report Filtering (F) Bit

Starting with Intel Core Duo processors, bit 12 in the "Form" column in Table 16-10 is used to indicate that a particular posting to a log may be the last posting for corrections in that line/entry, at least for some time:

- 0 in bit 12 indicates "normal" filtering (original P6/Pentium4/Atom/Xeon processor meaning).
- 1 in bit 12 indicates "corrected" filtering (filtering is activated for the line/entry in the posting). Filtering means that some or all of the subsequent corrections to this entry (in this structure) will not be posted. The enhanced error reporting introduced with the Intel Core Duo processors is based on tracking the lines affected by repeated corrections (see Section 16.4, "Enhanced Cache Error reporting"). This capability is indicated by IA32_MCG_CAP[11]. Only the first few correction events for a line are posted; subsequent redundant correction events to the same line are not posted. Uncorrected events are always posted.

The behavior of error filtering after crossing the yellow threshold is model-specific. Filtering has meaning only for corrected errors (UC=0 in IA32_MCi_STATUS MSR). System software must ignore filtering bit (12) for uncorrected errors.

16.9.2.2 Transaction Type (TT) Sub-Field

The 2-bit TT sub-field (Table 16-11) indicates the type of transaction (data, instruction, or generic). The sub-field applies to the TLB, cache, and interconnect error conditions. Note that interconnect error conditions are primarily associated with P6 family and Pentium processors, which utilize an external APIC bus separate from the system bus. The generic type is reported when the processor cannot determine the transaction type.

Table 16-11. Encoding for TT (Transaction Type) Sub-Field

Transaction Type	Mnemonic	Binary Encoding
Instruction	I	00
Data	D	01
Generic	G	10

16.9.2.3 Level (LL) Sub-Field

The 2-bit LL sub-field (see Table 16-12) indicates the level in the memory hierarchy where the error occurred (level 0, level 1, level 2, or generic). The LL sub-field also applies to the TLB, cache, and interconnect error conditions. The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors support two levels in the cache hierarchy and one level in the TLBs. Again, the generic type is reported when the processor cannot determine the hierarchy level.

Table 16-12. Level Encoding for LL (Memory Hierarchy Level) Sub-Field

Hierarchy Level	Mnemonic	Binary Encoding
Level 0	L0	00
Level 1	L1	01
Level 2	L2	10
Generic	LG	11

16.9.2.4 Request (RRRR) Sub-Field

The 4-bit RRRR sub-field (see Table 16-13) indicates the type of action associated with the error. Actions include read and write operations, prefetches, cache evictions, and snoops. Generic error is returned when the type of error cannot be determined. Generic read and generic write are returned when the processor cannot determine the type of instruction or data request that caused the error. Eviction and snoop requests apply only to the caches. All of the other requests apply to TLBs, caches, and interconnects.

Table 16-13. Encoding of Request (RRRR) Sub-Field

Request Type	Mnemonic	Binary Encoding
Generic Error	ERR	0000
Generic Read	RD	0001
Generic Write	WR	0010
Data Read	DRD	0011
Data Write	DWR	0100
Instruction Fetch	IRD	0101
Prefetch	PREFETCH	0110
Eviction	EVICT	0111
Snoop	SNOOP	1000
Page Walk	PW	1001
EPT Page Walk	EPW	1010

16.9.2.5 Bus and Interconnect Errors

The bus and interconnect errors are defined with the 2-bit PP (participation), 1-bit T (time-out), and 2-bit II (memory or I/O) sub-fields, in addition to the LL and RRRR sub-fields (see Table 16-14). The bus error conditions are implementation dependent and related to the type of bus implemented by the processor. Likewise, the interconnect error conditions are predicated on a specific implementation-dependent interconnect model that describes the connections between the different levels of the storage hierarchy. The type of bus is implementation dependent, and as such is not specified in this document. A bus or interconnect transaction consists of a request involving an address and a response.

Table 16-14. Encodings of PP, T, and II Sub-Fields

Sub-Field	Transaction	Mnemonic	Binary Encoding
PP (Participation)	Local processor* originated request	SRC	00
	Local processor* responded to request	RES	01
	Local processor* observed error as third party	OBS	10
	Generic		11
T (Time-out)	Request timed out	TIMEOUT	1
	Request did not time out	NOTIMEOUT	0
II (Memory or I/O)	Memory Access	M	00
	Reserved		01
	I/O	IO	10
	Other transaction		11

NOTE:

* Local processor differentiates the processor reporting the error from other system components (including the APIC, other processors, etc.).

16.9.2.6 Memory Controller and Extended Memory Errors

The memory controller errors are defined with the 3-bit MMM (memory transaction type), and 4-bit CCCC (channel) sub-fields. The encodings for MMM and CCCC are defined in Table 16-15. Extended Memory errors use the same encodings and are used to report errors in memory used as a cache.

Table 16-15. Encodings of MMM and CCCC Sub-Fields

Sub-Field	Transaction	Mnemonic	Binary Encoding
MMM	Generic undefined request	GEN	000
	Memory read error	RD	001
	Memory write error	WR	010
	Address/Command Error	AC	011
	Memory Scrubbing Error	MS	100
	Reserved		101-111
CCCC	Channel number	CHN	0000-1110
	Channel not specified		1111

Note that the CCCC channel number may be enumerated from zero separately by each memory controller on a system. On a multi-socket system, or a system with multiple memory controllers per socket, it is necessary to also consider which machine check bank logged the error. See Chapter 17 for details on specific implementations.

16.9.3 Architecturally Defined UCR Errors

Software recoverable compound error code are defined in this section.

16.9.3.1 Architecturally Defined SRAO Errors

The following two SRAO errors are architecturally defined.

- UCR Errors detected by memory controller scrubbing; and
- UCR Errors detected during L3 cache (L3) explicit writebacks.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 16-10). Their values and compound encoding format are given in Table 16-16.

Table 16-16. MCA Compound Error Code Encoding for SRAO Errors

Type	MCACOD Value	MCA Error Code Encoding[1]
Memory Scrubbing	C0H - CFH	0000_0000_1100_CCCC 000F 0000 1MMM CCCC (Memory Controller Error), where Memory subfield MMM = 100B (memory scrubbing) Channel subfield CCCC = channel # or generic
L3 Explicit Writeback	17AH	0000_0001_0111_1010 000F 0001 RRRR TTLL (Cache Hierarchy Error) where Request subfields RRRR = 0111B (Eviction) Transaction Type subfields TT = 10B (Generic) Level subfields LL = 10B

NOTES:

1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 16-17 lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAO errors.

Table 16-17. IA32_MCi_STATUS Values for SRAO Errors

SRAO Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Memory Scrubbing	1	0	1	x[1]	1	1	0	x[1]	0	COH-CFH
L3 Explicit Writeback	1	0	1	x[1]	1	1	0	x[1]	0	17AH

NOTES:

1. When signaled as MCE, EN=1 and S=1. If error was signaled via CMC, then EN=x, and S=0.

For both the memory scrubbing and L3 explicit writeback errors, the ADDRV and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

MCE signal is broadcast to all logical processors as outlined in Section 16.10.4.1. If LMCE is supported and enabled, some errors (not limited to UCR errors) may be delivered to only a single logical processor. System software should consult IA32_MCG_STATUS.LMCE_S to determine if the MCE signaled is only to this logical processor.

IA32_MCi_STATUS banks can be shared by logical processors within a core or within the same package. So several logical processors may find an SRAO error in the shared IA32_MCi_STATUS bank but other processors do not find it in any of the IA32_MCi_STATUS banks. Table 16-18 shows the RIPV and EIPV flag indication in the IA32_MCG_STATUS register for the memory scrubbing and L3 explicit writeback errors on both the reporting and non-reporting logical processors.

Table 16-18. IA32_MCG_STATUS Flag Indication for SRAO Errors

SRAO Type	Reporting Logical Processors		Non-reporting Logical Processors	
	RIPV	EIPV	RIPV	EIPV
Memory Scrubbing	1	0	1	0
L3 Explicit Writeback	1	0	1	0

16.9.3.2 Architecturally Defined SRAR Errors

The following six SRAR errors are architecturally defined:

- UCR Errors detected on data load;
- UCR Errors detected on data page walk;
- UCR Errors detected on data page walk on EPT;
- UCR Errors detected on instruction fetch;
- UCR Errors detected on instruction fetch page walk; and
- UCR Errors detected on instruction fetch page walk on EPT.

The MCA error code encodings for these six architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 16-10). Their values and compound encoding format are given in Table 16-19.

Table 16-19. MCA Compound Error Code Encoding for SRAR Errors

Type	MCACOD Value	MCA Error Code Encoding[1]
Data Load	134H	0000_0001_0011_0100: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0011B (Data Load), Transaction Type subfield TT= 01B (Data), and Level subfield LL = 00B (Level 0).
Data Page Walk	194H	0000_0001_1001_0100: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1001B (Page Walk), Transaction Type subfield TT= 01B (Data), and Level subfield LL = 00B (Level 0).
Data Page Walk on EPT	1A4H	0000_0001_1010_0100: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1010B (EPT Page Walk), Transaction Type subfield TT= 01B (Data), and Level subfield LL = 00B (Level 0).
Instruction Fetch	150H	0000_0001_0101_0000: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0101B (Instruction Fetch), Transaction Type subfield TT= 00B (Instruction), and Level subfield LL = 00B (Level 0).
Instruction Fetch Page Walk	190H	0000_0001_1001_0000: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1001B (Page Walk), Transaction Type subfield TT= 00B (Instruction), and Level subfield LL = 00B (Level 0).
Instruction Fetch Page Walk on EPT	1A0H	0000_0001_1010_0000: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1010B (EPT Page Walk), Transaction Type subfield TT= 00B (Instruction), and Level subfield LL = 00B (Level 0).

NOTES:

1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 16-20 lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAR errors.

Table 16-20. IA32_MCi_STATUS Values for All Defined SRAR Errors

SRAR Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR
All defined SRAR errors defined in Table 16-19	1	0	1	1	1	1	0	1	1

For all defined SRAR errors, the ADDRV and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the data load and instruction fetch errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

MCE signal is broadcast to all logical processors on the system on which the UCR errors are supported, except when the processor supports LMCE and LMCE is enabled by system software (see Section 16.3.1.5). The IA32_MCG_STATUS MSR allows system software to distinguish the affected logical processor of an SRAR error amongst logical processors that observed SRAR via MCi_STATUS bank.

Table 16-21 shows the RIPV and EIPV flag indication in the IA32_MCG_STATUS register for the data load and instruction fetch errors on both the reporting and non-reporting logical processors. The recoverable SRAR error reported by a processor may be continuable, where the system software can interpret the context of continuable as follows: the error was isolated, contained. If software can rectify the error condition in the current instruction stream, the execution context on that logical processor can be continued without loss of information.

Table 16-21. IA32_MCG_STATUS Flag Indication for SRAR Errors

SRAR Type	Affected Logical Processor			Non-Affected Logical Processors		
	RIPV	EIPV	Continuable	RIPV	EIPV	Continuable
Recoverable-continuable	1	1	Yes[1]	1	0	Yes
Recoverable-not-continuable	0	x	No			

NOTES:

1. See the definition of the context of "continuable" above and additional details below.

SRAR Error And Affected Logical Processors

The affected logical processor is the one that has detected and raised an SRAR error at the point of the consumption in the execution flow. The affected logical processor should find the Data Load or the Instruction Fetch error information in the IA32_MCi_STATUS register that is reporting the SRAR error.

Table 16-21 list the actionable scenarios that system software can respond to an SRAR error on an affected logical processor according to RIPV and EIPV values:

- Recoverable-continuable SRAR Error (RIPV=1, EIPV=1):

  For recoverable-continuable SRAR errors, the affected logical processor should find that both the IA32_MCG_STATUS.RIPV and the IA32_MCG_STATUS.EIPV flags are set, indicating that system software may be able to restart execution from the interrupted context if it is able to rectify the error condition. If system software cannot rectify the error condition then it must treat the error as a recoverable error where restarting execution with the interrupted context is not possible. Restarting without rectifying the error condition will result in most cases with another SRAR error on the same instruction.

- Recoverable-not-continuable SRAR Error (RIPV=0, EIPV=x):

  For recoverable-not-continuable errors, the affected logical processor should find that either

  — IA32_MCG_STATUS.RIPV= 0, IA32_MCG_STATUS.EIPV=1, or

  — IA32_MCG_STATUS.RIPV= 0, IA32_MCG_STATUS.EIPV=0.

  In either case, this indicates that the error is detected at the instruction pointer saved on the stack for this machine check exception and restarting execution with the interrupted context is not possible. System software may take the following recovery actions for the affected logical processor:

  - The current executing thread cannot be continued. System software must terminate the interrupted stream of execution and provide a new stream of execution on return from the machine check handler for the affected logical processor.

SRAR Error And Non-Affected Logical Processors

The logical processors that observed but not affected by an SRAR error should find that the RIPV flag in the IA32_MCG_STATUS register is set and the EIPV flag in the IA32_MCG_STATUS register is cleared, indicating that it is safe to restart the execution at the instruction saved on the stack for the machine check exception on these processors after the recovery action is successfully taken by system software.

16.9.4 Multiple MCA Errors

When multiple MCA errors are detected within a certain detection window, the processor may aggregate the reporting of these errors together as a single event, i.e., a single machine exception condition. If this occurs, system software may find multiple MCA errors logged in different MC banks on one logical processor or find multiple MCA errors logged across different processors for a single machine check broadcast event. In order to handle multiple UCR errors reported from a single machine check event and possibly recover from multiple errors, system software may consider the following:

- Whether it can recover from multiple errors is determined by the most severe error reported on the system. If the most severe error is found to be an unrecoverable error (VAL=1, UC=1, PCC=1 and EN=1) after system software examines the MC banks of all processors to which the MCA signal is broadcast, recovery from the multiple errors is not possible and system software needs to reset the system.

- When multiple recoverable errors are reported and no other fatal condition (e.g., overflowed condition for SRAR error) is found for the reported recoverable errors, it is possible for system software to recover from the multiple recoverable errors by taking necessary recovery action for each individual recoverable error. However, system software can no longer expect one to one relationship with the error information recorded in the IA32_MCi_STATUS register and the states of the RIPV and EIPV flags in the IA32_MCG_STATUS register as the states of the RIPV and the EIPV flags in the IA32_MCG_STATUS register may indicate the information for the most severe error recorded on the processor. System software is required to use the RIPV flag indication in the IA32_MCG_STATUS register to make a final decision of recoverability of the errors and find the restart-ability requirement after examining each IA32_MCi_STATUS register error information in the MC banks.

  In certain cases where system software observes more than one SRAR error logged for a single logical processor, it can no longer rely on affected threads as specified in Table 15-20 above. System software is recommended to reset the system if this condition is observed.

16.9.5 Machine-Check Error Codes Interpretation

Chapter 17, "Interpreting Machine Check Error Codes," provides information on interpreting the MCA error code, model-specific error code, and other information error code fields. For P6 family processors, information has been included on decoding external bus errors. For Pentium 4 and Intel Xeon processors; information is included on external bus, internal timer and cache hierarchy errors.

16.10 GUIDELINES FOR WRITING MACHINE-CHECK SOFTWARE

The machine-check architecture and error logging can be used in three different ways:

- To detect machine errors during normal instruction execution, using the machine-check exception (#MC).
- To periodically check and log machine errors.
- To examine recoverable UCR errors, determine software recoverability and perform recovery actions via a machine-check exception handler or a corrected machine-check interrupt handler.

To use the machine-check exception, the operating system or executive software must provide a machine-check exception handler. This handler may need to be designed specifically for each family of processors.

A special program or utility is required to log machine errors.

Guidelines for writing a machine-check exception handler or a machine-error logging utility are given in the following sections.

16.10.1 Machine-Check Exception Handler

The machine-check exception (#MC) corresponds to vector 18. To service machine-check exceptions, a trap gate must be added to the IDT. The pointer in the trap gate must point to a machine-check exception handler. Two approaches can be taken to designing the exception handler:

1. The handler can merely log all the machine status and error information, then call a debugger or shut down the system.
2. The handler can analyze the reported error information and, in some cases, attempt to correct the error and restart the processor.

For Pentium 4, Intel Xeon, Intel Atom, P6 family, and Pentium processors; virtually all machine-check conditions cannot be corrected (they result in abort-type exceptions). The logging of status and error information is therefore a baseline implementation requirement.

When IA32_MCG_CAP[24] is clear, consider the following when writing a machine-check exception handler:

- To determine the nature of the error, the handler must read each of the error-reporting register banks. The count field in the IA32_MCG_CAP register gives number of register banks. The first register of register bank 0 is at address 400H.

- The VAL (valid) flag in each IA32_MC*i*_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and do not need to be checked.
- To write a portable exception handler, only the MCA error code field in the IA32_MC*i*_STATUS register should be checked. See Section 16.9, "Interpreting the MCA Error Codes," for information that can be used to write an algorithm to interpret this field.
- Correctable errors are corrected automatically by the processor. The UC flag in each IA32_MCi_STATUS register indicates whether the processor automatically corrected an error.
- The RIPV, PCC, and OVER flags in each IA32_MCi_STATUS register indicate whether recovery from the error is possible. If PCC or OVER are set, recovery is not possible. If RIPV is not set, program execution can not be restarted reliably. When recovery is not possible, the handler typically records the error information and signals an abort to the operating system.
- The RIPV flag in the IA32_MCG_STATUS register indicates whether the program can be restarted at the instruction indicated by the instruction pointer (the address of the instruction pushed on the stack when the exception was generated). If this flag is clear, the processor may still be able to be restarted (for debugging purposes) but not without loss of program continuity.
- For unrecoverable errors, the EIPV flag in the IA32_MCG_STATUS register indicates whether the instruction indicated by the instruction pointer pushed on the stack (when the exception was generated) is related to the error. If the flag is clear, the pushed instruction may not be related to the error.
- The MCIP flag in the IA32_MCG_STATUS register indicates whether a machine-check exception was generated. Before returning from the machine-check exception handler, software should clear this flag so that it can be used reliably by an error logging utility. The MCIP flag also detects recursion. The machine-check architecture does not support recursion. When the processor detects machine-check recursion, it enters the shutdown state.

Example 16-2 gives typical steps carried out by a machine-check exception handler.

Example 16-2. Machine-Check Exception Handler Pseudocode

```
IF CPU supports MCE
    THEN
        IF CPU supports MCA
            THEN
                call errorlogging routine; (* returns restartability *)
        FI;
    ELSE (* Pentium(R) processor compatible *)
        READ P5_MC_ADDR
        READ P5_MC_TYPE;
        report RESTARTABILITY to console;
FI;
IF error is not restartable
    THEN
        report RESTARTABILITY to console;
        abort system;
FI;
CLEAR MCIP flag in IA32_MCG_STATUS;
```

16.10.2 Pentium Processor Machine-Check Exception Handling

Machine-check exception handler on P6 family, Intel Atom and later processor families, should follow the guidelines described in Section 16.10.1 and Example 16-2 that check the processor's support of MCA.

NOTE

On processors that support MCA (CPUID.1.EDX.MCA = 1) reading the P5_MC_TYPE and P5_MC_ADDR registers may produce invalid data.

When machine-check exceptions are enabled for the Pentium processor (MCE flag is set in control register CR4), the machine-check exception handler uses the RDMSR instruction to read the error type from the P5_MC_TYPE

register and the machine check address from the P5_MC_ADDR register. The handler then normally reports these register values to the system console before aborting execution (see Example 16-2).

16.10.3 Logging Correctable Machine-Check Errors

The error handling routine for servicing the machine-check exceptions is responsible for logging uncorrected errors.

If a machine-check error is correctable, the processor does not generate a machine-check exception for it. To detect correctable machine-check errors, a utility program must be written that reads each of the machine-check error-reporting register banks and logs the results in an accounting file or data structure. This utility can be implemented in either of the following ways.

- A system daemon that polls the register banks on an infrequent basis, such as hourly or daily.
- A user-initiated application that polls the register banks and records the exceptions. Here, the actual polling service is provided by an operating-system driver or through the system call interface.
- An interrupt service routine servicing CMCI can read the MC banks and log the error. Please refer to Section 16.10.4.2 for guidelines on logging correctable machine checks.

Example 16-3 gives pseudocode for an error logging utility.

Example 16-3. Machine-Check Error Logging Pseudocode

```
Assume that execution is restartable;
IF the processor supports MCA
    THEN
    FOR each bank of machine-check registers
        DO
            READ IA32_MCi_STATUS;
            IF VAL flag in IA32_MCi_STATUS = 1
                THEN
                    IF ADDRV flag in IA32_MCi_STATUS = 1
                        THEN READ IA32_MCi_ADDR;
                    FI;
                    IF MISCV flag in IA32_MCi_STATUS = 1
                        THEN READ IA32_MCi_MISC;
                    FI;
                    IF MCIP flag in IA32_MCG_STATUS = 1
                        (* Machine-check exception is in progress *)
                        AND PCC flag in IA32_MCi_STATUS = 1
                        OR RIPV flag in IA32_MCG_STATUS = 0
                        (* execution is not restartable *)
                            THEN
                                RESTARTABILITY = FALSE;
                                return RESTARTABILITY to calling procedure;
                    FI;
                    Save time-stamp counter and processor ID;
                    Set IA32_MCi_STATUS to all 0s;
                    Execute serializing instruction (i.e., CPUID);
            FI;
        OD;
FI;
```

If the processor supports the machine-check architecture, the utility reads through the banks of error-reporting registers looking for valid register entries. It then saves the values of the IA32_MC*i*_STATUS, IA32_MC*i*_ADDR, IA32_MC*i*_MISC, and IA32_MCG_STATUS registers for each bank that is valid. The routine minimizes processing time by recording the raw data into a system data structure or file, reducing the overhead associated with polling. User utilities analyze the collected data in an off-line environment.

When the MCIP flag is set in the IA32_MCG_STATUS register, a machine-check exception is in progress and the machine-check exception handler has called the exception logging routine.

Once the logging process has been completed the exception-handling routine must determine whether execution can be restarted, which is usually possible when damage has not occurred (The PCC flag is clear, in the IA32_MC*i*_STATUS register) and when the processor can guarantee that execution is restartable (the RIPV flag is set in the IA32_MCG_STATUS register). If execution cannot be restarted, the system is not recoverable and the exception-handling routine should signal the console appropriately before returning the error status to the Operating System kernel for subsequent shutdown.

The machine-check architecture allows buffering of exceptions from a given error-reporting bank although the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors do not implement this feature. The error logging routine should provide compatibility with future processors by reading each hardware error-reporting bank's IA32_MC*i*_STATUS register and then writing 0s to clear the OVER and VAL flags in this register. The error logging utility should re-read the IA32_MC*i*_STATUS register for the bank ensuring that the valid bit is clear. The processor will write the next error into the register bank and set the VAL flags.

Additional information that should be stored by the exception-logging routine includes the processor's time-stamp counter value, which provides a mechanism to indicate the frequency of exceptions. A multiprocessing operating system stores the identity of the processor node incurring the exception using a unique identifier, such as the processor's APIC ID (see Section 11.8, "Handling Interrupts").

The basic algorithm given in Example 16-3 can be modified to provide more robust recovery techniques. For example, software has the flexibility to attempt recovery using information unavailable to the hardware. Specifically, the machine-check exception handler can, after logging carefully analyze the error-reporting registers when the error-logging routine reports an error that does not allow execution to be restarted. These recovery techniques can use external bus related model-specific information provided with the error report to localize the source of the error within the system and determine the appropriate recovery strategy.

16.10.4 Machine-Check Software Handler Guidelines for Error Recovery

16.10.4.1 Machine-Check Exception Handler for Error Recovery

When writing a machine-check exception (MCE) handler to support software recovery from Uncorrected Recoverable (UCR) errors, consider the following:

- When IA32_MCG_CAP [24] is zero, there are no recoverable errors supported and all machine-check are fatal exceptions. The logging of status and error information is therefore a baseline implementation requirement.
- When IA32_MCG_CAP [24] is 1, certain uncorrected errors called uncorrected recoverable (UCR) errors may be software recoverable. The handler can analyze the reported error information, and in some cases attempt to recover from the uncorrected error and continue execution.
- For processors on which CPUID reports DisplayFamily_DisplayModel as 06H_0EH and onward, an MCA signal is broadcast to all logical processors in the system; 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. Due to the potentially shared machine check MSR resources among the logical processors on the same package/core, the MCE handler may be required to synchronize with the other processors that received a machine check error and serialize access to the machine check registers when analyzing, logging, and clearing the information in the machine check registers.
  - On processors that indicate ability for local machine-check exception (MCG_LMCE_P), hardware can choose to report the error to only a single logical processor if system software has enabled LMCE by setting IA32_MCG_EXT_CTL[LMCE_EN] = 1 as outlined in Section 16.3.1.5.
- The VAL (valid) flag in each IA32_MCi_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and should not be checked.
- The MCE handler is primarily responsible for processing uncorrected errors. The UC flag in each IA32_MCi_Status register indicates whether the reported error was corrected (UC=0) or uncorrected (UC=1). The MCE handler can optionally log and clear the corrected errors in the MC banks if it can implement software algorithm to avoid the undesired race conditions with the CMCI or CMC polling handler.
- For uncorrectable errors, the EIPV flag in the IA32_MCG_STATUS register indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is

generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.

- The MCIP flag in the IA32_MCG_STATUS register indicates whether a machine-check exception was generated. When a machine check exception is generated, it is expected that the MCIP flag in the IA32_MCG_STATUS register is set to 1. If it is not set, this machine check was generated by either an INT 18 instruction or some piece of hardware signaling an interrupt with vector 18.

When IA32_MCG_CAP [24] is 1, the following rules can apply when writing a machine check exception (MCE) handler to support software recovery:

- The PCC flag in each IA32_MCi_STATUS register indicates whether recovery from the error is possible for uncorrected errors (UC=1). If the PCC flag is set for enabled uncorrected errors (UC=1 and EN=1), recovery is not possible. When recovery is not possible, the MCE handler typically records the error information and signals the operating system to reset the system.
- The RIPV flag in the IA32_MCG_STATUS register indicates whether restarting the program execution from the instruction pointer saved on the stack for the machine check exception is possible. When the RIPV is set, program execution can be restarted reliably when recovery is possible. If the RIPV flag is not set, program execution cannot be restarted reliably. In this case the recovery algorithm may involve terminating the current program execution and resuming an alternate thread of execution upon return from the machine check handler when recovery is possible. When recovery is not possible, the MCE handler signals the operating system to reset the system.
- When the EN flag is zero but the VAL and UC flags are one in the IA32_MCi_STATUS register, the reported uncorrected error in this bank is not enabled. As uncorrected errors with the EN flag = 0 are not the source of machine check exceptions, the MCE handler should log and clear non-enabled errors when the S bit is set and should continue searching for enabled errors from the other IA32_MCi_STATUS registers. Note that when IA32_MCG_CAP [24] is 0, any uncorrected error condition (VAL =1 and UC=1) including the one with the EN flag cleared are fatal and the handler must signal the operating system to reset the system. For the errors that do not generate machine check exceptions, the EN flag has no meaning.
- When the VAL flag is one, the UC flag is one, the EN flag is one and the PCC flag is zero in the IA32_MCi_STATUS register, the error in this bank is an uncorrected recoverable (UCR) error. The MCE handler needs to examine the S flag and the AR flag to find the type of the UCR error for software recovery and determine if software error recovery is possible.
- When both the S and the AR flags are clear in the IA32_MCi_STATUS register for the UCR error (VAL=1, UC=1, EN=x and PCC=0), the error in this bank is an uncorrected no-action required error (UCNA). UCNA errors are uncorrected but do not require any OS recovery action to continue execution. These errors indicate that some data in the system is corrupt, but that data has not been consumed and may not be consumed. If that data is consumed a non-UCNA machine check exception will be generated. UCNA errors are signaled in the same way as corrected machine check errors and the CMCI and CMC polling handler is primarily responsible for handling UCNA errors. Like corrected errors, the MCA handler can optionally log and clear UCNA errors as long as it can avoid the undesired race condition with the CMCI or CMC polling handler. As UCNA errors are not the source of machine check exceptions, the MCA handler should continue searching for uncorrected or software recoverable errors in all other MC banks.
- When the S flag in the IA32_MCi_STATUS register is set for the UCR error ((VAL=1, UC=1, EN=1 and PCC=0), the error in this bank is software recoverable and it was signaled through a machine-check exception. The AR flag in the IA32_MCi_STATUS register further clarifies the type of the software recoverable errors.
- When the AR flag in the IA32_MCi_STATUS register is clear for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action optional (SRAO) error. The MCE handler and the operating system can analyze the IA32_MCi_STATUS [15:0] to implement MCA error code specific optional recovery action, but this recovery action is optional. System software can resume the program execution from the instruction pointer saved on the stack for the machine check exception when the RIPV flag in the IA32_MCG_STATUS register is set.
- Even if the OVER flag in the IA32_MCi_STATUS register is set for the SRAO error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=0), the MCE handler can take recovery action for the SRAO error logged in the IA32_MCi_STATUS register. Since the recovery action for SRAO errors is optional, restarting the program execution from the instruction pointer saved on the stack for the machine check exception is still possible for the overflowed SRAO error if the RIPV flag in the IA32_MCG_STATUS is set.

- When the AR flag in the IA32_MCi_STATUS register is set for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action required (SRAR) error. The MCE handler and the operating system must take recovery action in order to continue execution after the machine-check exception. The MCA handler and the operating system need to analyze the IA32_MCi_STATUS [15:0] to determine the MCA error code specific recovery action. If no recovery action can be performed, the operating system must reset the system.
- When the OVER flag in the IA32_MCi_STATUS register is set for the SRAR error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=1), the MCE handler cannot take recovery action as the information of the SRAR error in the IA32_MCi_STATUS register was potentially lost due to the overflow condition. Since the recovery action for SRAR errors must be taken, the MCE handler must signal the operating system to reset the system.
- When the MCE handler cannot find any uncorrected (VAL=1, UC=1 and EN=1) or any software recoverable errors (VAL=1, UC=1, EN=1, PCC=0 and S=1) in any of the IA32_MCi banks of the processors, this is an unexpected condition for the MCE handler and the handler should signal the operating system to reset the system.
- Before returning from the machine-check exception handler, software must clear the MCIP flag in the IA32_MCG_STATUS register. The MCIP flag is used to detect recursion. The machine-check architecture does not support recursion. When the processor receives a machine check when MCIP is set, it automatically enters the shutdown state.

Example 16-4 gives pseudocode for an MC exception handler that supports recovery of UCR.

Example 16-4. Machine-Check Error Handler Pseudocode Supporting UCR

```
MACHINE CHECK HANDLER:  (* Called from INT 18 handler *)
NOERROR = TRUE;
ProcessorCount = 0;
IF CPU supports MCA
    THEN
        RESTARTABILITY = TRUE;
        IF (Processor Family = 6 AND DisplayModel ≥ 0EH) OR (Processor Family > 6)
            THEN
                IF ( MCG_LMCE = 1)
                    MCA_BROADCAST = FALSE;
                ELSE
                    MCA_BROADCAST = TRUE;
                FI;
                Acquire SpinLock;
                ProcessorCount++;  (* Allowing one logical processor at a time to examine machine check registers *)
                CALL MCA ERROR PROCESSING; (* returns RESTARTABILITY and NOERROR *)
            ELSE
                MCA_BROADCAST = FALSE;
                (* Implement a rendezvous mechanism with the other processors if necessary *)
                CALL MCA ERROR PROCESSING;
        FI;
    ELSE (* Pentium(R) processor compatible *)
        READ P5_MC_ADDR
        READ P5_MC_TYPE;
        RESTARTABILITY = FALSE;
FI;

IF NOERROR = TRUE
    THEN
        IF NOT (MCG_RIPV = 1 AND MCG_EIPV = 0)
            THEN
                RESTARTABILITY = FALSE;
        FI
FI;

IF RESTARTABILITY = FALSE
    THEN
        Report RESTARTABILITY to console;
        Reset system;
```

```
FI;

IF MCA_BROADCAST = TRUE
    THEN
        IF ProcessorCount = MAX_PROCESSORS
          AND NOERROR = TRUE
            THEN
                Report RESTARTABILITY to console;
                Reset system;
        FI;
        Release SpinLock;
        Wait till ProcessorCount = MAX_PROCESSRS on system;
        (* implement a timeout and abort function if necessary *)
FI;
CLEAR IA32_MCG_STATUS;
RESUME Execution;
(* End of MACHINE CHECK HANDLER*)

MCA ERROR PROCESSING:   (* MCA Error Processing Routine called from MCA Handler *)
IF MCIP flag in IA32_MCG_STATUS = 0
    THEN (* MCIP=0 upon MCA is unexpected *)
        RESTARTABILITY = FALSE;
FI;
FOR each bank of machine-check registers
    DO
        CLEAR_MC_BANK = FALSE;
        READ IA32_MCi_STATUS;
        IF VAL Flag in IA32_MCi_STATUS = 1
            THEN
                IF UC Flag in IA32_MCi_STATUS = 1
                    THEN
                        IF Bit 24 in IA32_MCG_CAP = 0
                            THEN (* the processor does not support software error recovery *)
                                RESTARTABILITY = FALSE;
                                NOERROR = FALSE;
                                GOTO LOG MCA REGISTER;
                        FI;
                        (* the processor supports software error recovery *)
                        IF EN Flag in IA32_MCi_STATUS = 0 AND OVER Flag in IA32_MCi_STATUS=0
                            THEN (* It is a spurious MCA Log. Log and clear the register *)
                                CLEAR_MC_BANK = TRUE;
                                GOTO LOG MCA REGISTER;
                        FI;
                        IF PCC = 1 and EN = 1 in IA32_MCi_STATUS
                            THEN (* processor context might have been corrupted *)
                                RESTARTABILITY = FALSE;
                            ELSE (* It is a uncorrected recoverable (UCR) error *)
                                IF S Flag in IA32_MCi_STATUS = 0
                                    THEN
                                        IF AR Flag in IA32_MCi_STATUS = 0
                                            THEN (* It is a uncorrected no action required (UCNA) error *)
                                                GOTO CONTINUE; (* let CMCI and CMC polling handler to process *)
                                            ELSE
                                                RESTARTABILITY = FALSE; (* S=0, AR=1 is illegal *)
                                        FI
                                FI;
                                IF RESTARTABILITY = FALSE
                                    THEN (* no need to take recovery action if RESTARTABILITY is already false *)
                                        NOERROR = FALSE;
                                        GOTO LOG MCA REGISTER;
                                FI;
                                (* S in IA32_MCi_STATUS = 1 *)
                                IF AR Flag in IA32_MCi_STATUS = 1
                                    THEN (* It is a software recoverable and action required (SRAR) error *)
                                        IF OVER Flag in IA32_MCi_STATUS = 1
```

```
                                    THEN
                                        RESTARTABILITY = FALSE;
                                        NOERROR = FALSE;
                                        GOTO LOG MCA REGISTER;
                                FI
                                IF MCACOD Value in IA32_MCi_STATUS is recognized
                                   AND Current Processor is an Affected Processor
                                    THEN
                                        Implement MCACOD specific recovery action;
                                        CLEAR_MC_BANK = TRUE;
                                    ELSE
                                        RESTARTABILITY = FALSE;
                                FI;
                            ELSE (* It is a software recoverable and action optional (SRAO) error *)
                                IF OVER Flag in IA32_MCi_STATUS = 0 AND
                                 MCACOD in IA32_MCi_STATUS is recognized
                                    THEN
                                        Implement MCACOD specific recovery action;
                                FI;
                                CLEAR_MC_BANK = TRUE;
                        FI; AR
                    FI; PCC
                    NOERROR = FALSE;
                    GOTO LOG MCA REGISTER;
                ELSE  (* It is a corrected error; continue to the next IA32_MCi_STATUS *)
                    GOTO CONTINUE;
            FI; UC
        FI; VAL
LOG MCA REGISTER:
        SAVE IA32_MCi_STATUS;
        If MISCV in IA32_MCi_STATUS
            THEN
                SAVE IA32_MCi_MISC;
        FI;
        IF ADDRV in IA32_MCi_STATUS
            THEN
                SAVE IA32_MCi_ADDR;
        FI;
        IF CLEAR_MC_BANK = TRUE
            THEN
                SET all 0 to IA32_MCi_STATUS;
                If MISCV in IA32_MCi_STATUS
                    THEN
                        SET all 0 to IA32_MCi_MISC;
                FI;
                IF ADDRV in IA32_MCi_STATUS
                    THEN
                        SET all 0 to IA32_MCi_ADDR;
                FI;
        FI;
        CONTINUE:
    OD;
( *END FOR *)
RETURN;
(* End of MCA ERROR PROCESSING*)
```

16.10.4.2 Corrected Machine-Check Handler for Error Recovery

When writing a corrected machine check handler, which is invoked as a result of CMCI or called from an OS CMC Polling dispatcher, consider the following:

- The VAL (valid) flag in each IA32_MCi_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank does not contain valid error information and does not need to be checked.

- The CMCI or CMC polling handler is responsible for logging and clearing corrected errors. The UC flag in each IA32_MCi_Status register indicates whether the reported error was corrected (UC=0) or not (UC=1).
- When IA32_MCG_CAP [24] is one, the CMC handler is also responsible for logging and clearing uncorrected no-action required (UCNA) errors. When the UC flag is one but the PCC, S, and AR flags are zero in the IA32_MCi_STATUS register, the reported error in this bank is an uncorrected no-action required (UCNA) error. In cases when SRAO error are signaled as UCNA error via CMCI, software can perform recovery for those errors identified in Table 16-16.
- In addition to corrected errors and UCNA errors, the CMC handler optionally logs uncorrected (UC=1 and PCC=1), software recoverable machine check errors (UC=1, PCC=0 and S=1), but should avoid clearing those errors from the MC banks. Clearing these errors may result in accidentally removing these errors before these errors are actually handled and processed by the MCE handler for attempted software error recovery.

Example 16-5 gives pseudocode for a CMCI handler with UCR support.

Example 16-5. Corrected Error Handler Pseudocode with UCR Support

```
Corrected Error HANDLER:  (* Called from CMCI handler or OS CMC Polling Dispatcher*)
IF CPU supports MCA
    THEN
        FOR each bank of machine-check registers
            DO
                READ IA32_MCi_STATUS;
                IF VAL flag in IA32_MCi_STATUS = 1
                    THEN
                        IF UC Flag in IA32_MCi_STATUS = 0 (* It is a corrected error *)
                            THEN
                                GOTO LOG CMC ERROR;
                            ELSE
                                IF Bit 24 in IA32_MCG_CAP = 0
                                    THEN
                                        GOTO CONTINUE;
                                FI;
                                IF S Flag in IA32_MCi_STATUS = 0 AND AR Flag in IA32_MCi_STATUS = 0
                                    THEN (* It is a uncorrected no action required error *)
                                        GOTO LOG CMC ERROR
                                FI
                                IF EN Flag in IA32_MCi_STATUS = 0
                                    THEN (* It is a spurious MCA error *)
                                        GOTO LOG CMC ERROR
                                FI;
                        FI;
                FI;
                GOTO CONTINUE;
            LOG CMC ERROR:
                SAVE IA32_MCi_STATUS;
                If MISCV Flag in IA32_MCi_STATUS
                    THEN
                        SAVE IA32_MCi_MISC;
                        SET all 0 to IA32_MCi_MISC;
                FI;
                IF ADDRV Flag in IA32_MCi_STATUS
                    THEN
                        SAVE IA32_MCi_ADDR;
                        SET all 0 to IA32_MCi_ADDR
                FI;
                SET all 0 to IA32_MCi_STATUS;
                CONTINUE:
            OD;
    ( *END FOR *)
FI;
```

11. Updates to Chapter 18, Volume 3B

Change bars and violet 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:

- Updated Section 18.18.7, "Monitoring Resource Selection and Reporting Infrastructure," and Figures 18-24 and 18-25 to add the IA32_QM_CTR overflow bit (bit 61).
- Typo correction to the caption of Figure 18-40, "Layout of the IA32_L3_IO_RDT_CFG MSR for Enabling Non-CPU Agent Intel® RDT." Previously, this caption was erroneously titled "Layout of the IA32_L3_IO_QOS_CFG MSR for Enabling Non-CPU Agent Intel® RDT."
- Updated all instances of "COS" to "CLOS" in this chapter, including figures. These changes are not marked with change bars or violet text.

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 that 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 that enable control over shared platform resources are described in Section 18.19, "Intel® Resource Director Technology (Intel® RDT) Allocation Features."
- Features that enable control over shared platform resources for non-CPU agents are described in Section 18.20, "Intel® Resource Director Technology (Intel® RDT) for Non-CPU Agents."[1]

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.

1. Additional information about Intel® RDT can be found in the document titled "Intel® Resource Director Technology Architecture Specification," available here: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

- **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 MSRs 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.

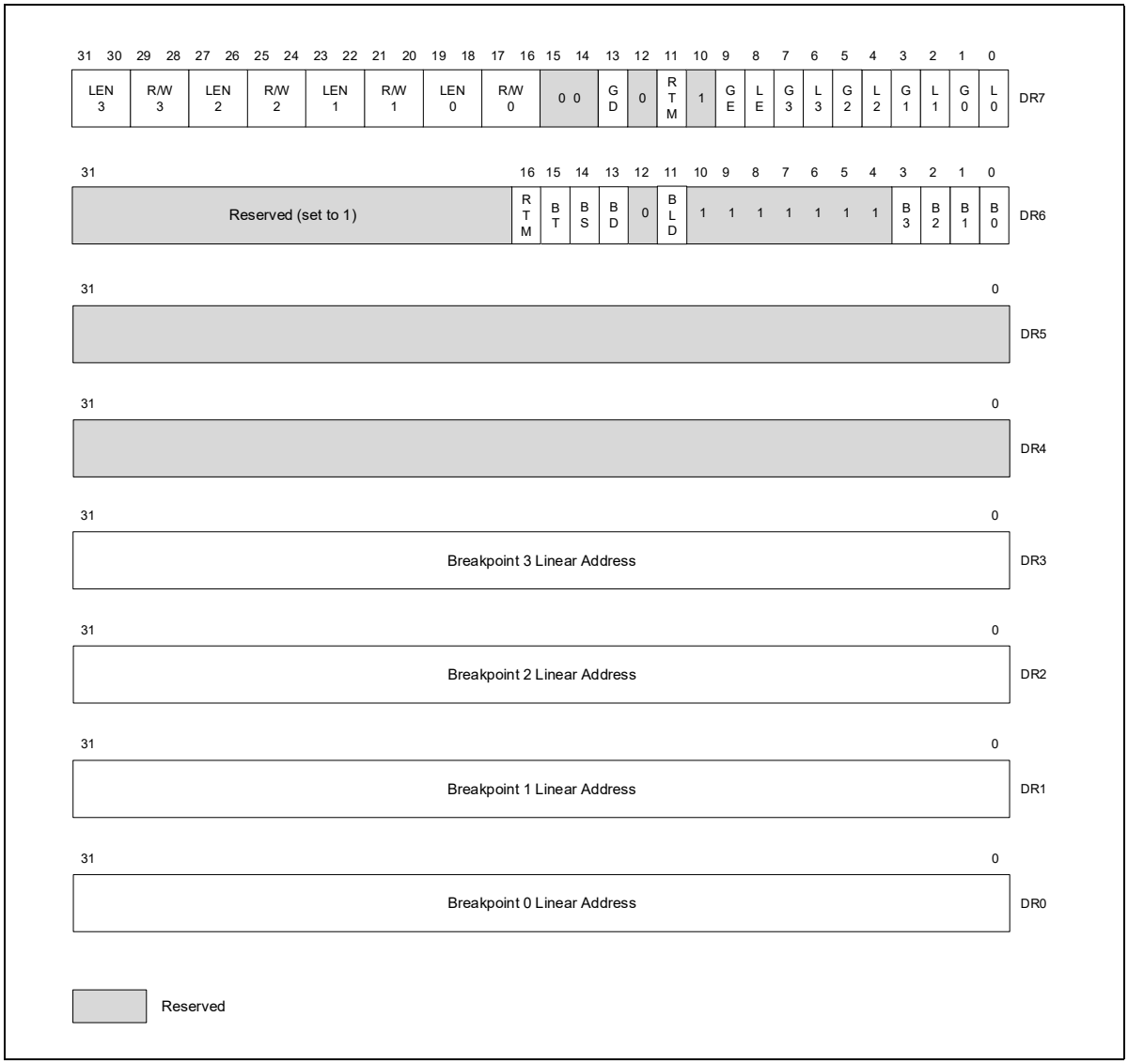


Figure 18-1. Debug Registers

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 though 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 DR1 DR2 DR3	R/W0 = 11 (Read/Write) R/W1 = 01 (Write) R/W2 = 11 (Read/Write) R/W3 = 01 (Write)	A0001H A0002H B0002H C0000H	LEN0 = 00 (1 byte) LEN1 = 00 (1 byte) LEN2 = 01) (2 bytes) LEN3 = 11 (4 bytes)
Data Accesses			
Operation		**Address**	**Access Length (In Bytes)**
Data operations that trap - Read or write - Read or write - Write - Write - Read or write - Read or write - Read or write - Write - Write - Write		 A0001H A0001H A0002H A0002H B0001H B0002H B0002H C0000H C0001H C0003H	 1 2 1 2 4 1 2 4 2 1

Table 18-1. Breakpoint Examples (Contd.)

Debug Register Setup			
Debug Register	**R/W*n***	**Breakpoint Address**	**LEN*n***
Data operations that do not trap - Read or write - Read - Read or write - Read or write - Read - Read or write		 A0000H A0002H A0003H B0000H C0000H C0004H	 1 1 4 2 2 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.

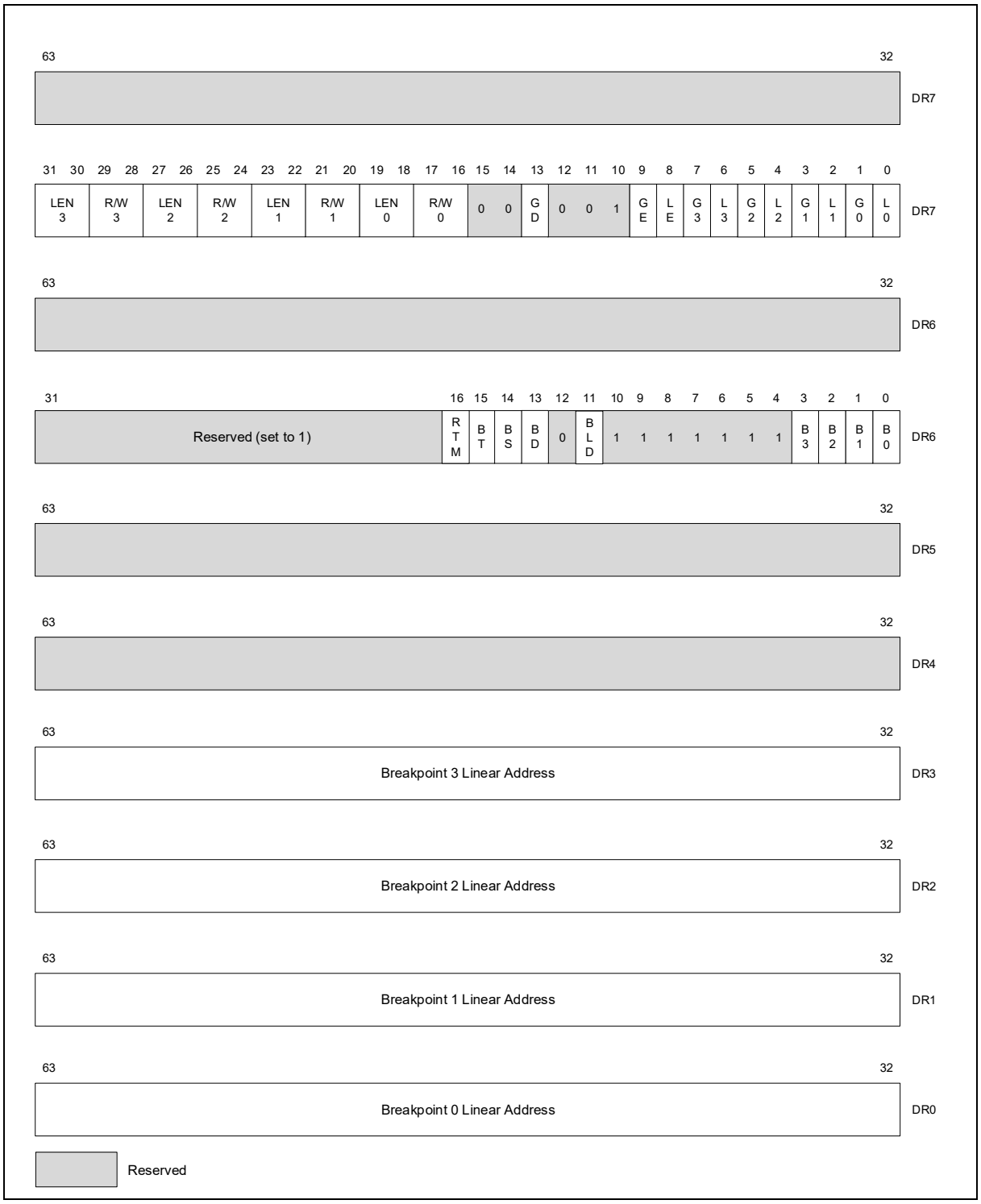


Figure 18-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

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/Wn = 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/Wn = 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/Wn = 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/Wn = 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 the 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.

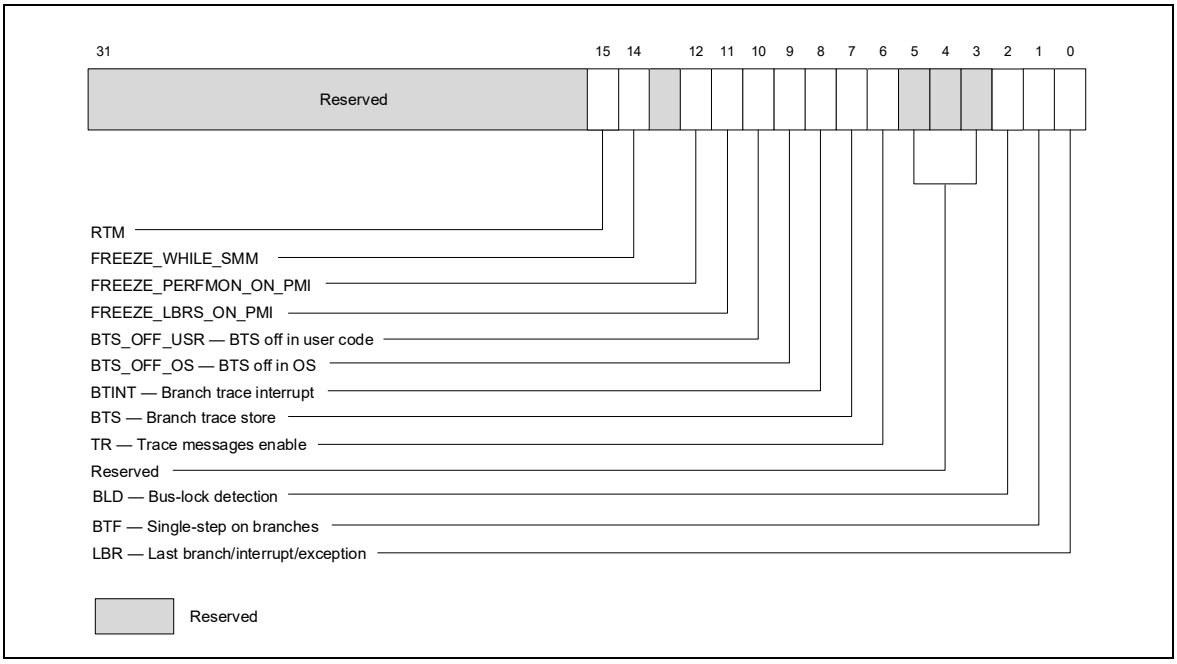


Figure 18-3. IA32_DEBUGCTL MSR for Processors Based on Intel® Core™ Microarchitecture

- **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_CAPABIL-ITIES.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 the CPUID instruction in Chapter 3 of the 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 employs 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.

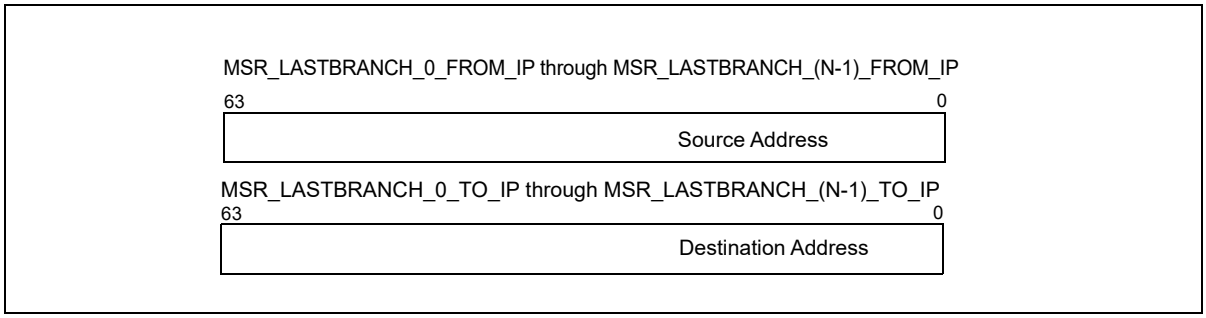


Figure 18-4. 64-bit Address Layout of LBR MSR

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.

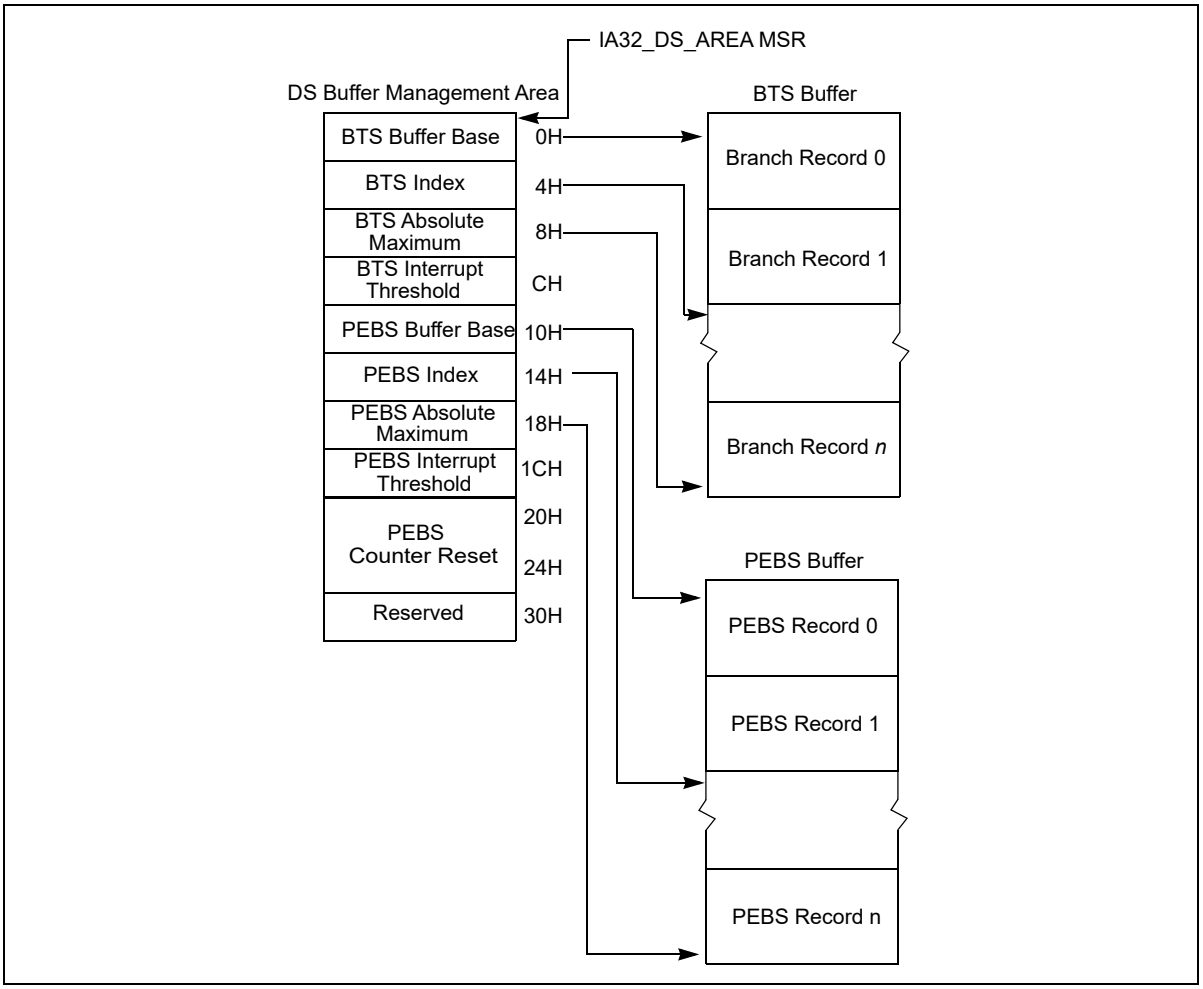


Figure 18-5. DS Save Area Example[1]

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).

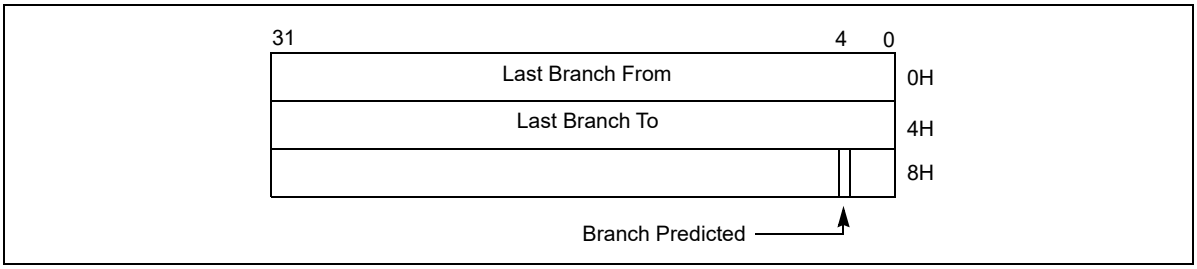


Figure 18-6. 32-bit Branch Trace Record Format

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.

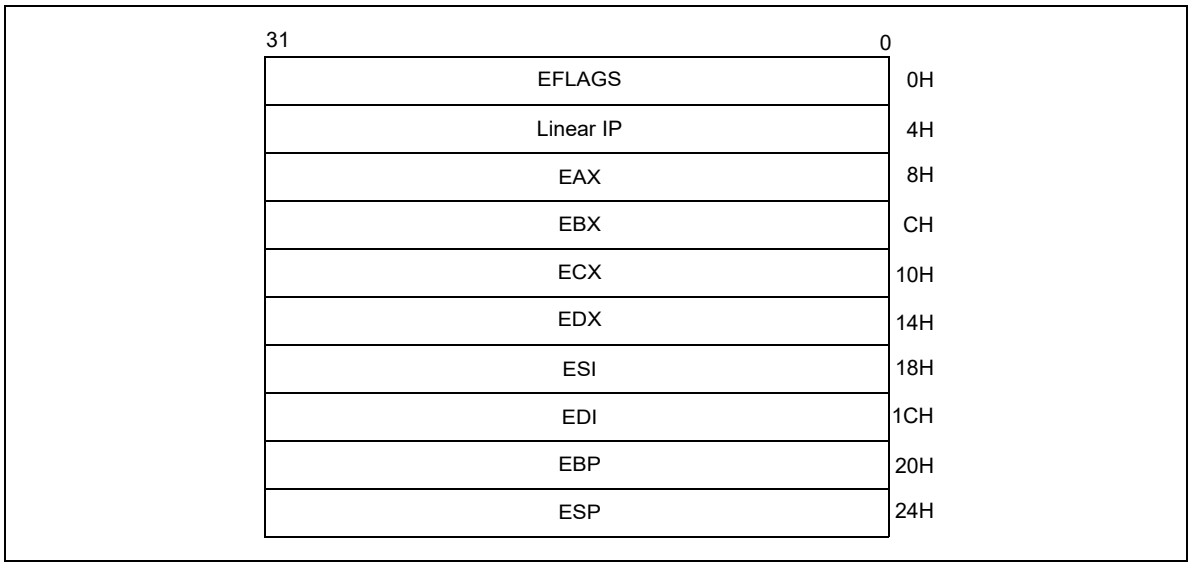


Figure 18-7. PEBS Record Format

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.

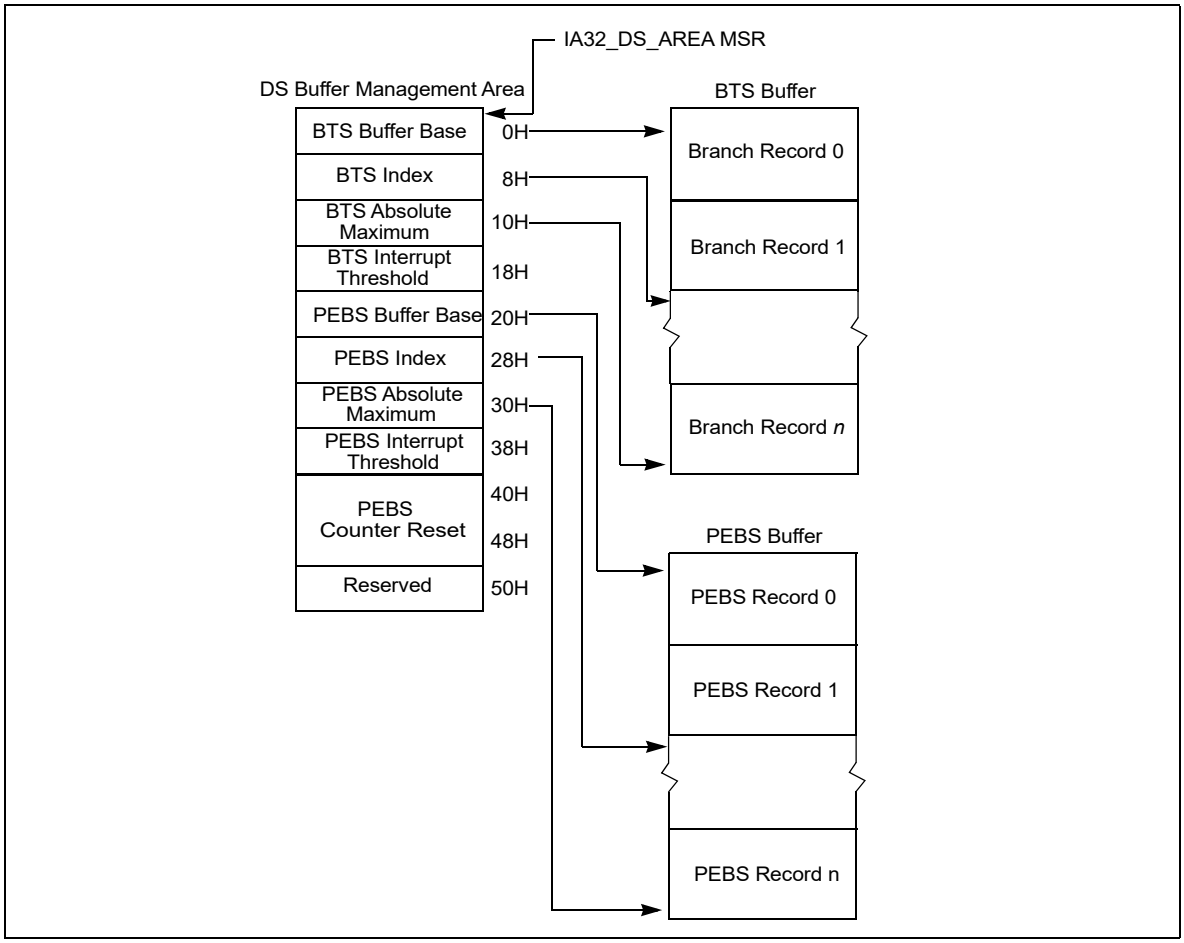


Figure 18-8. IA-32e Mode DS Save Area Example[1]

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).

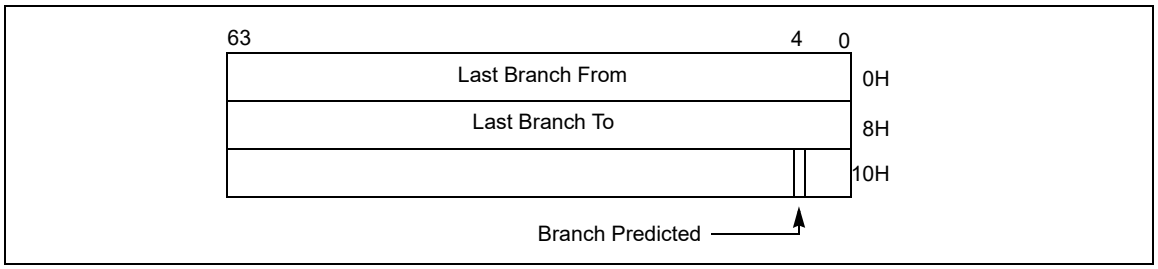


Figure 18-9. 64-bit Branch Trace Record Format

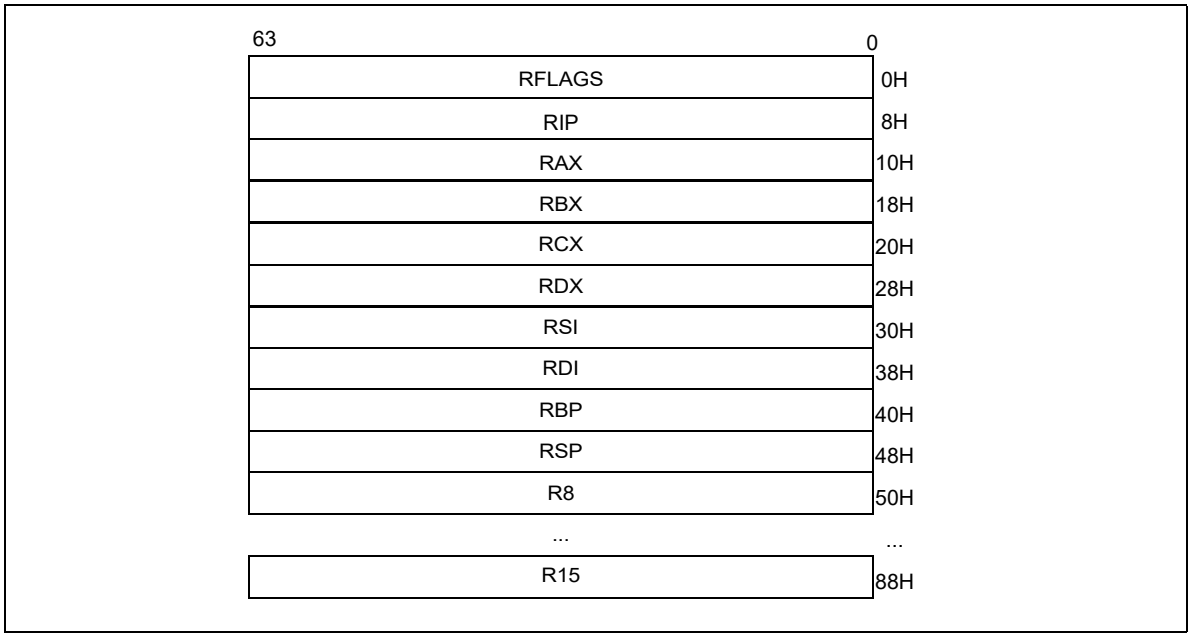


Figure 18-10. 64-bit PEBS Record Format

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 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).

3. 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_L-BR_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. MISPRED 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.

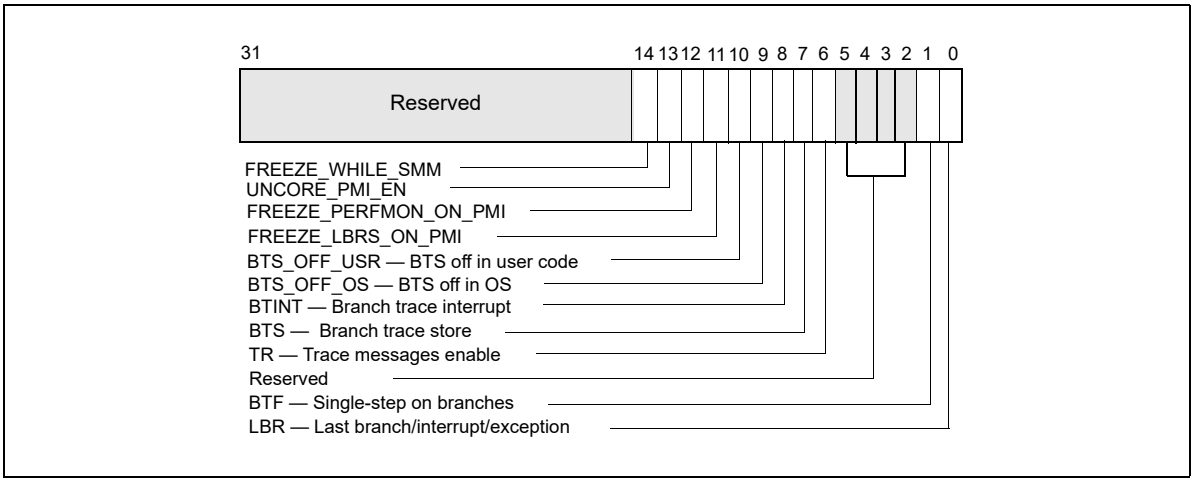


Figure 18-11. IA32_DEBUGCTL MSR for Processors Based on Nehalem Microarchitecture

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 support 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 consist 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_LAST-BRANCH_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® Microar-chitecture."
- **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."

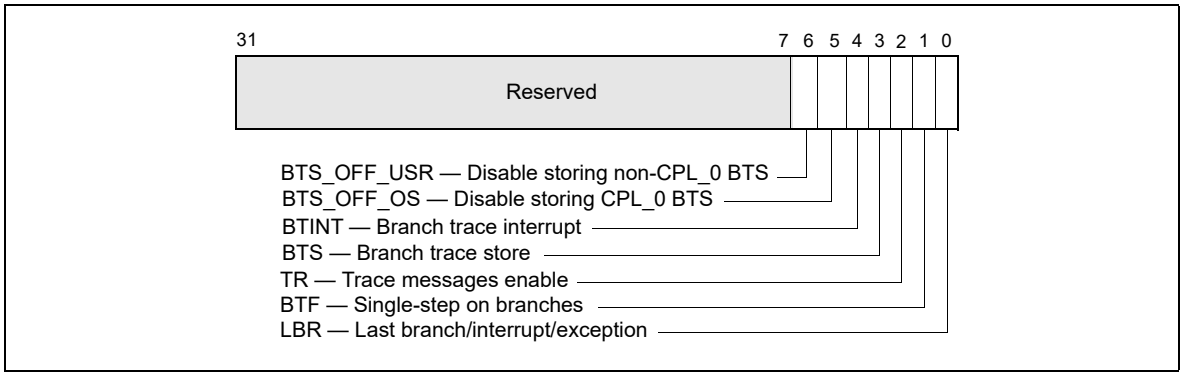


Figure 18-12. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **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 it 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.

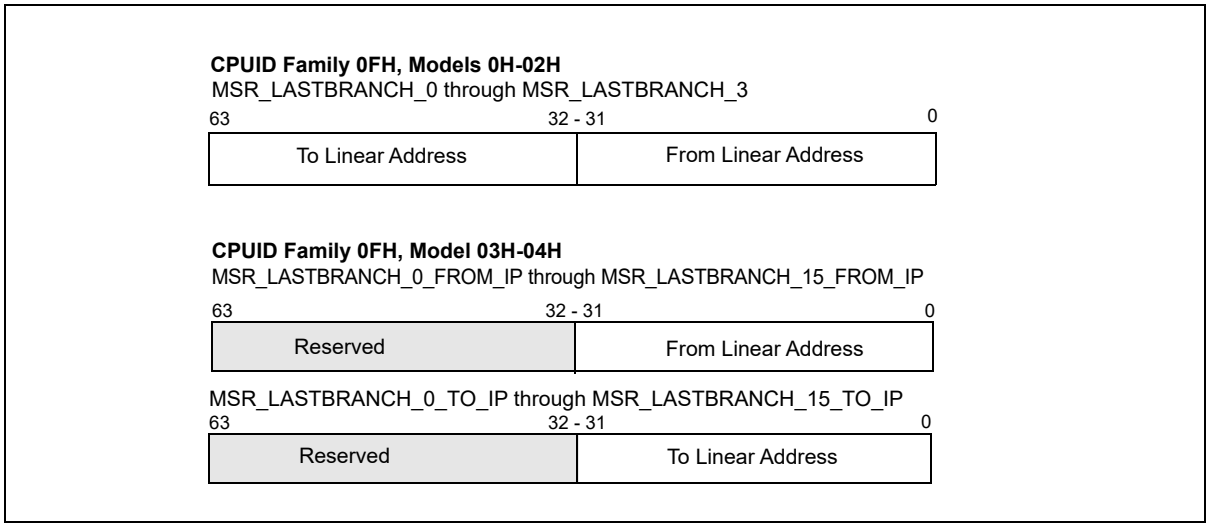


Figure 18-13. LBR MSR Branch Record Layout for the Pentium 4 and Intel® Xeon® Processor Family

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.

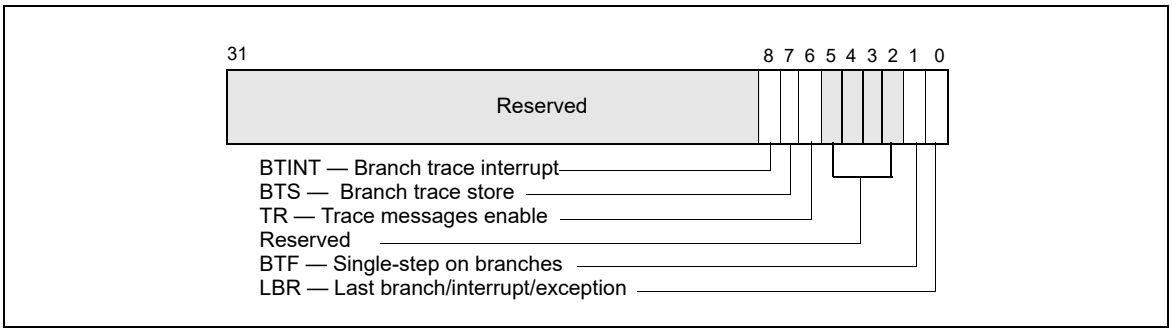


Figure 18-14. IA32_DEBUGCTL MSR for Intel® Core™ Solo and Intel® Core™ Duo Processors

- **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 Last-ExceptionFromIP 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.

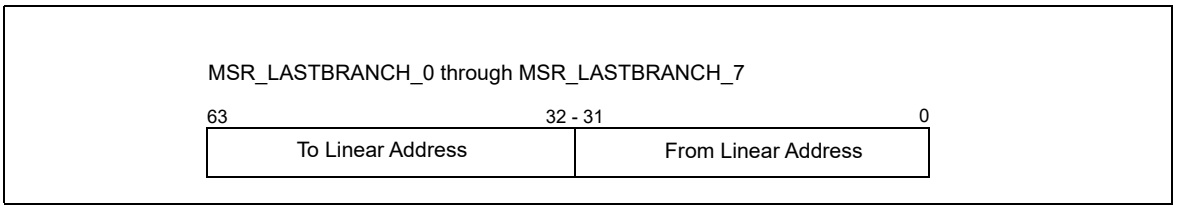


Figure 18-15. LBR Branch Record Layout for the Intel® Core™ Solo and Intel® Core™ Duo Processor

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.
  - **PB*i* (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 BP*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, 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.

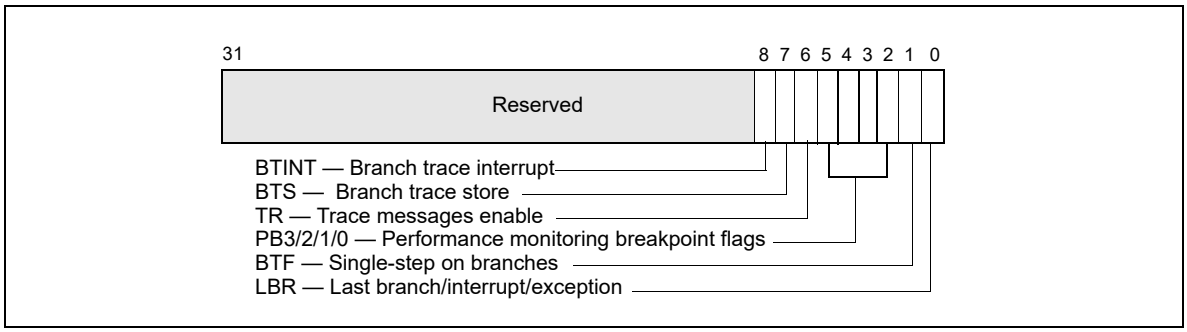


Figure 18-16. MSR_DEBUGCTLB MSR for Pentium M Processors

- **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.

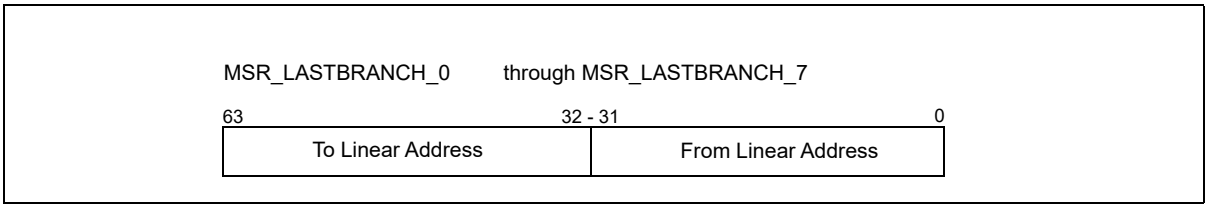


Figure 18-17. LBR Branch Record Layout for the Pentium M Processor

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: DEBUGCTLMSR, 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 DEBUGCTLMSR Register

The version of the DEBUGCTLMSR 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 DEBUGCTLMSR 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.

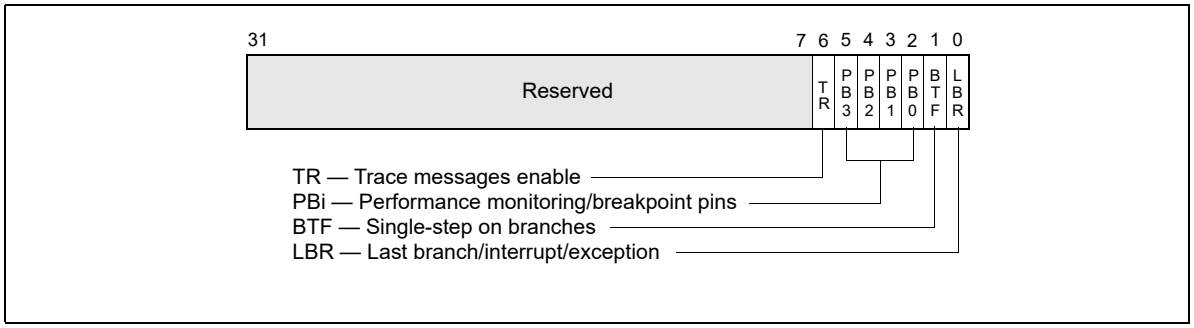


Figure 18-18. DEBUGCTLMSR Register (P6 Family Processors)

- **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 BP*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 DEBUGCTLMSR 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.80000001H: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:

TSC_Value = (ART_Value * CPUID.15H:EBX[31:0])/ CPUID.15H:EAX[31:0] + K

Where 'K' is an offset that can be adjusted by a privileged agent[1].

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 ID**s (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. To read back monitored data, software configures 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.

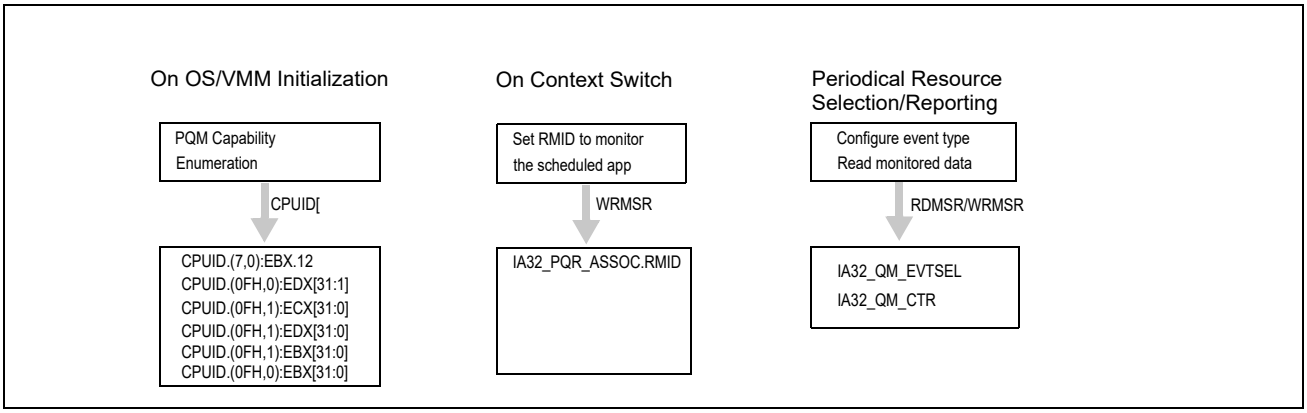


Figure 18-19. Platform Shared Resource Monitoring Usage Flow

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 CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.(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:

- CPUID 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 CPUID with EAX=0 to discover the "cpuid_maxLeaf" supported in the processor;
2. If cpuid_maxLeaf >= 7, then execute CPUID with EAX=7, ECX= 0 to verify CPUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] is set;
3. If CPUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] = 1, then execute CPUID with EAX=0FH, ECX= 0 to query available resource types that support monitoring;
4. If CPUID.(EAX=0FH, ECX=0):EDX.L3[bit 1] = 1, then execute CPUID with EAX=0FH, ECX= 1 to query the specific capabilities of L3 Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.
5. If CPUID.(EAX=0FH, ECX=0):EDX reports additional resource types supporting monitoring, then execute CPUID with EAX=0FH, ECX set to a corresponding resource type ID (ResID) as enumerated by the bit position of CPUID.(EAX=0FH, ECX=0):EDX.

18.18.4 Monitoring Resource Type and Capability Enumeration

CPUID 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 CPUID 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.

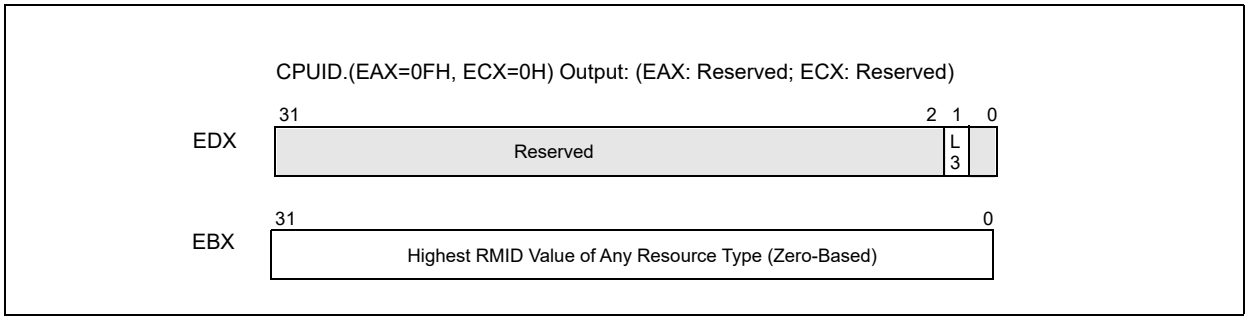


Figure 18-20. CPUID.(EAX=0FH, ECX=0H) Monitoring Resource Type Enumeration

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 MSRs 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.

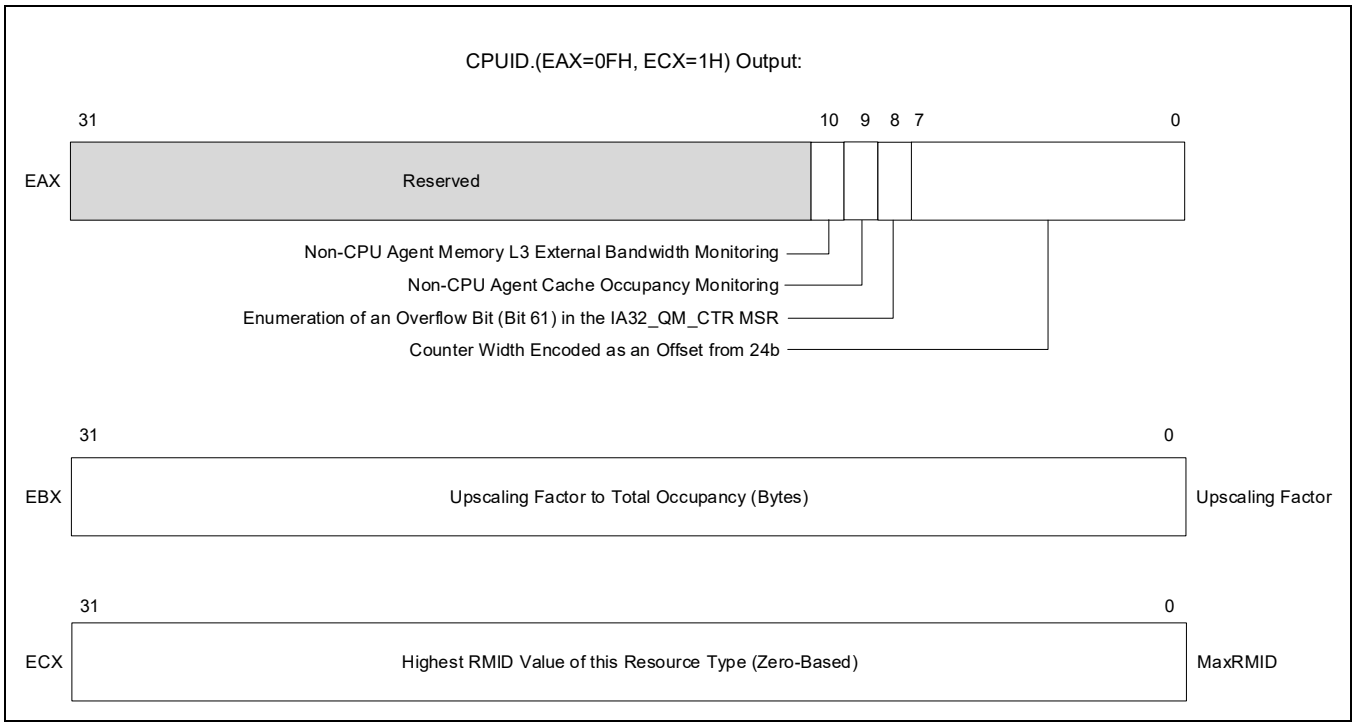


Figure 18-21. L3 Cache Monitoring Capability Enumeration Data (CPUID.(EAX=0FH, ECX=1H))

CPUID.(EAX=0FH, ECX=1H).EAX[7:0]: Encode counter width as offset from 24b. See Section 18.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.(EAX=0FH, ECX=1H).EAX[bit 8]: If 1, indicates the presence of an overflow bit in the IA32_QM_CTR MSR. See Section 18.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.(EAX=0FH, ECX=1H).EAX[bit 9]: If 1, indicates the presence of non-CPU agent Intel RDT CMT support. See Section 18.20 for details. Bits 31:11 of EAX are reserved.

CPUID.(EAX=0FH, ECX=1H).EAX[bit 10]:If 1, indicates the presence of non-CPU agent Intel RDT MBM support. See Section 18.20 for details. Bits 31:11 of EAX are reserved.

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.

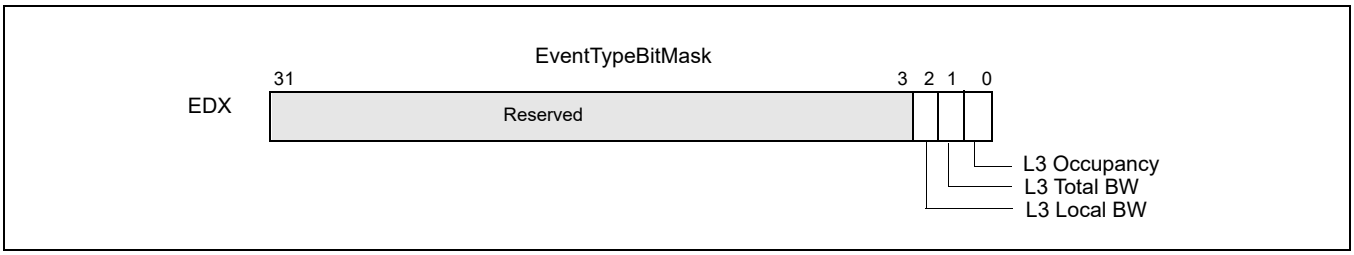


Figure 18-22. L3 Cache Monitoring Capability Enumeration Event Type Bit Vector (CPUID.(EAX=0FH, ECX=1H))

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 returns with bit 0 set. The corresponding event ID is shown in Table 18-18. The L3 occupancy data accumulated in the IA32_QM_CTR MSR 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 are defined 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 is shown in 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

A field is added to CPUID to enumerate the MBM counter width in platforms that support the extensible MBM counter width feature.

- CPUID.(EAX=0FH, ECX=1H).EAX[7:0]: Encode counter width as offset from 24b in bits[7:0]. In EAX bits 7:0, the counter width is encoded as an offset from 24b. A value of zero in this field means 24-bit counters are supported. A value of 8 indicates that 32-bit counters are supported, as in the 3rd generation Intel Xeon Scalable Processor Family. With this enumerable counter width, the requirement that software polls at 1Hz is removed. Software may poll at a varying rate with a reduced risk of rollover. Under typical conditions, rollover will likely require hundreds of seconds (though this value is not explicitly specified and may vary and decrease in future processor generations as memory bandwidths increase). Suppose software seeks to ensure that rollover does not occur more than once between samples. In that case, sampling at 1Hz while consuming the enumerated counter widths' worth of data will provide this guarantee for a specific platform and counter width under all conditions.
- CPUID.(EAX=0FH, ECX=1H).EAX[8]: Enumeration of the presence of an overflow bit in the IA32_QM_CTR MSR via EAX bit[8]. Software that uses the MBM event retrieval MSR interface should be updated to comprehend this new format, which enables up to 62-bit MBM counters to be provided by future platforms. Higher-level software that consumes the resulting bandwidth values is not expected to be affected. An overflow bit is defined in the IA32_QM_CTR MSR, bit 61, if CPUID.(EAX=0FH, ECX=1H).EAX[bit 8] is set. This rollover bit will be set on overflow of the MBM counters and reset upon read. Current processors do not support this capability.

18.18.6 Monitoring Resource RMID Association

After Monitoring and sub-features have 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 + CPUID.(EAX=0FH, ECX=0):EBX[31:0])). The value of IA32_PQR_ASSOC after power-on is 0.

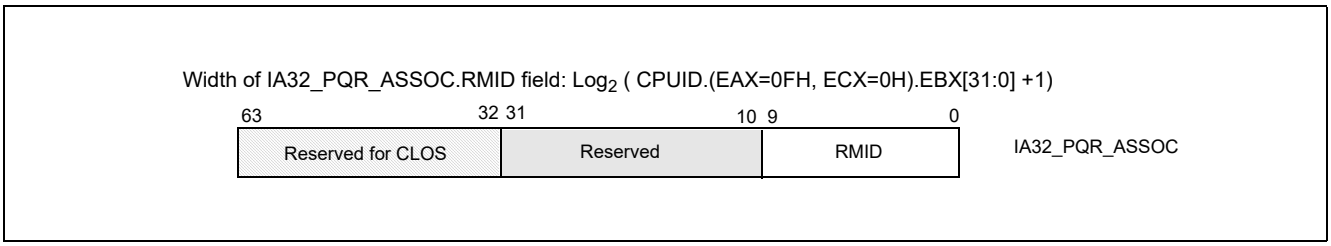


Figure 18-23. IA32_PQR_ASSOC MSR

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. IA32_QM_EVTSEL.EvtID (bits 7:0) specifies 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 bits 63 and 62 are both clear. The IA32_QM_CTR.Overflow (bit 61) is present if CPUID.(EAX = 0FH, ECX=1):EAX[bit 8] is set. This bit is set on overflow of the MBM counters and will reset upon read. 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.

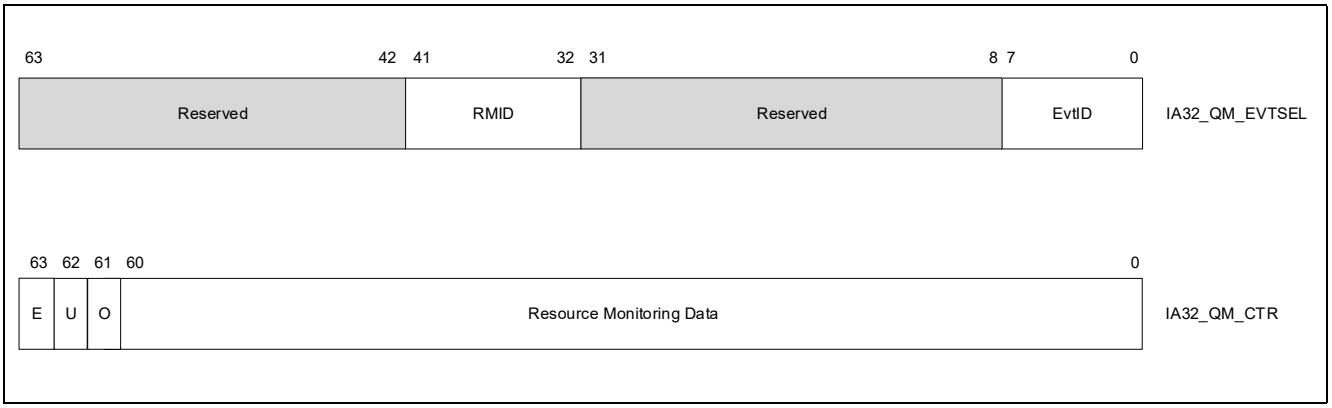


Figure 18-24. IA32_QM_EVTSEL and IA32_QM_CTR MSRs

18.18.8 Monitoring Programming Considerations

Figure 18-25 illustrates how system software can program IA32_QOSEVTSEL and IA32_QM_CTR to perform resource monitoring.

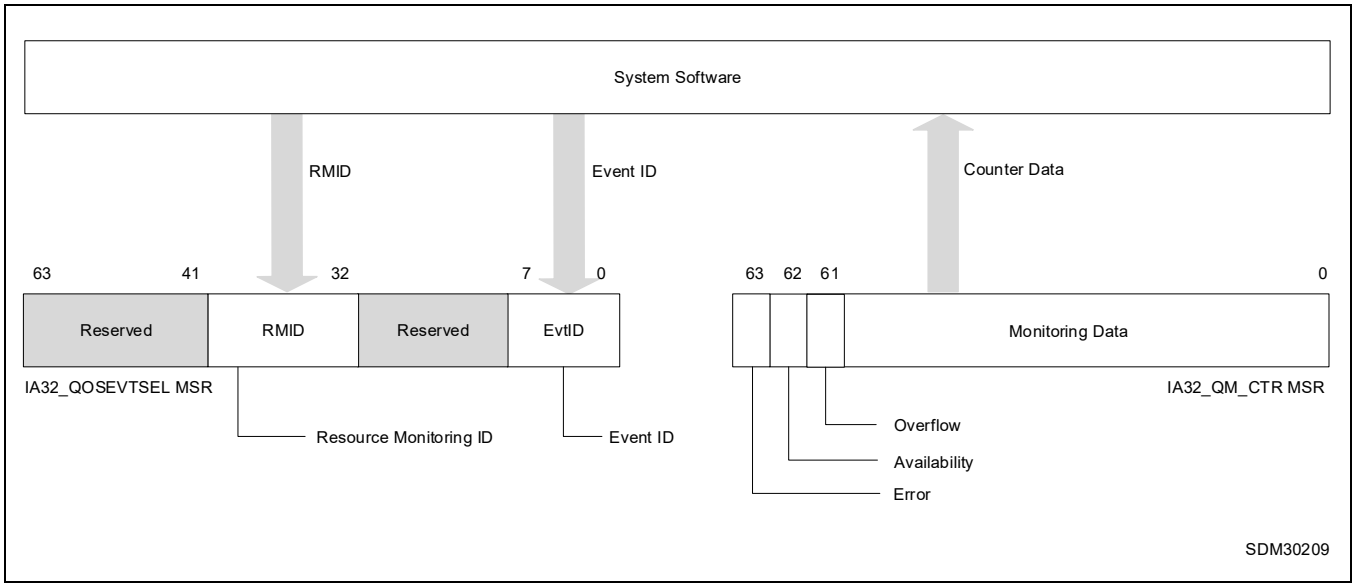


Figure 18-25. Software Usage of Cache Monitoring Resources

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 is 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 changes to counter width are enumerated to software; see Section 18.18.5.2 for details.

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 maximum RMID is used.

18.18.8.2 Monitoring Operation With Power Saving Features

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.
- 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.

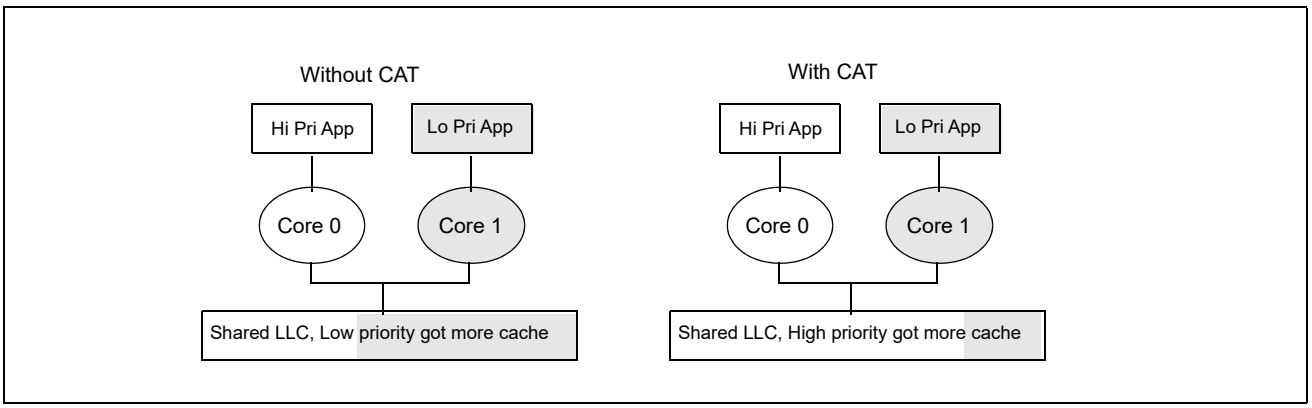


Figure 18-26. Cache Allocation Technology Enables Allocation of More Resources to High Priority Applications

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 (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 CLOS when an application, thread or VM is scheduled.

The usage of Classes of Service (CLOS) are consistent across resources and a CLOS may have multiple resource control attributes attached, which reduces software overhead at context swap time. Rather than adding new types of CLOS tags per resource for instance, the CLOS 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 CLOS 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 CLOS to an executing software thread (i.e., associating the active CR3 of a logical processor with the CLOS in IA32_PQR_ASSOC).
- Implementation-dependent mechanisms to indicate which CLOS is associated with a memory access and to enforce the cache allocation on a per CLOS 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 CLOS may vary for each resource type, and these details can be enumerated via CPUID.

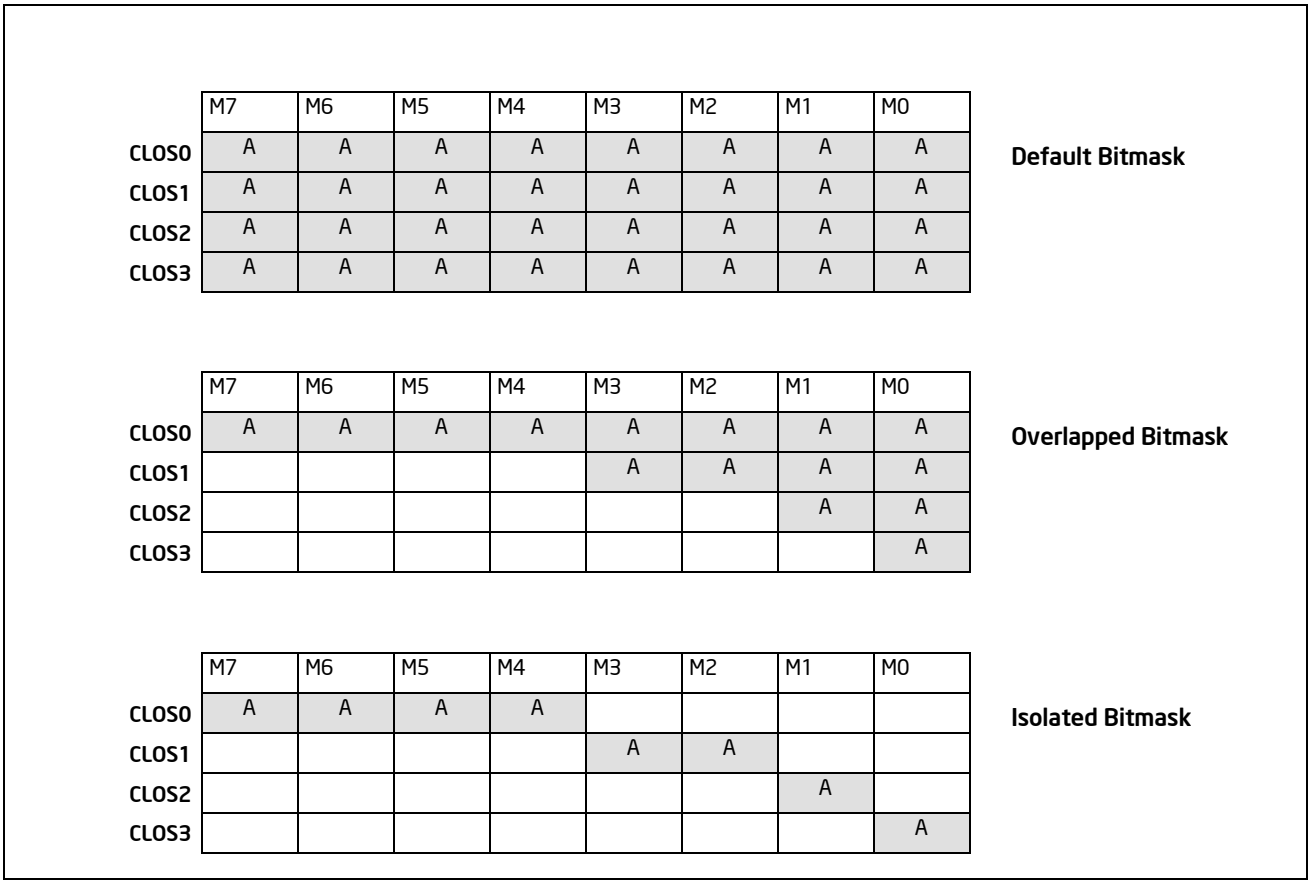


Figure 18-27. Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bit length of 8 are shown in Figure 18-27. Note that all (and only) contiguous '1' combinations are allowed (e.g., FFFFH, 0FF0H, 003CH, etc.), unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. 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 CLOS are implementation-dependent) have full access to the cache. The second case shows an overlapped case,

which would allow some lower-priority threads to 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, CLOS0 should typically be considered and configured as the highest priority CLOS, followed by CLOS1, and so on, though there is no hardware restriction enforcing this mapping. When the system boots all threads are initialized to CLOS0, 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.

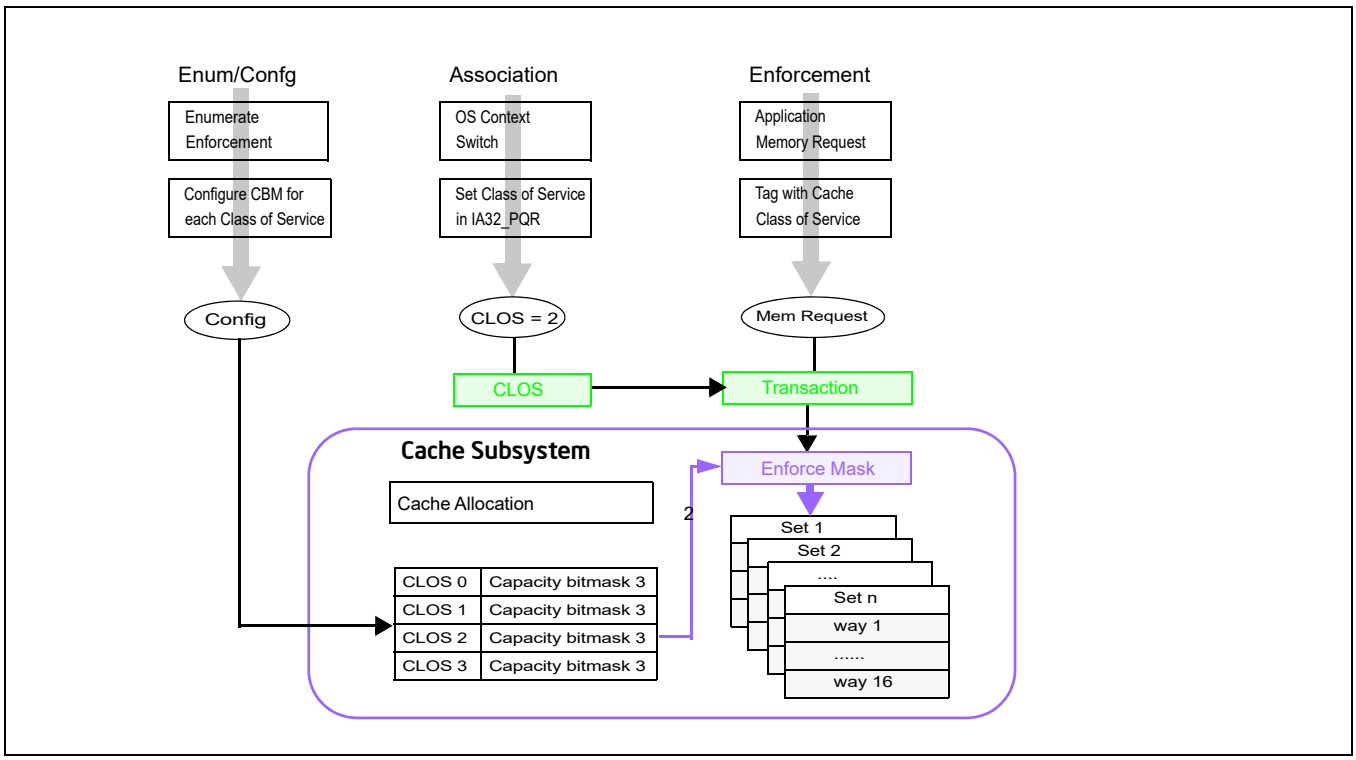


Figure 18-28. Class of Service and Cache Capacity Bitmasks

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, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. 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 CLOS (i.e., the corresponding CLOS in the PQR) and all requests to the CAT-capable resource from that logical processor are tagged with that CLOS (in other words, the application thread is configured to belong to a specific CLOS). 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 (CLOS). 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 CLOS for CAT is re-indexed, with the lower-half of the CLOS 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 CLOS number to enable control over the cache space which a given CLOS (and thus applications, threads or VMs) may occupy. Shown at the bottom are example mask configurations where CDP is enabled, and each CLOS 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-CLOS basis, depending on application and system needs.

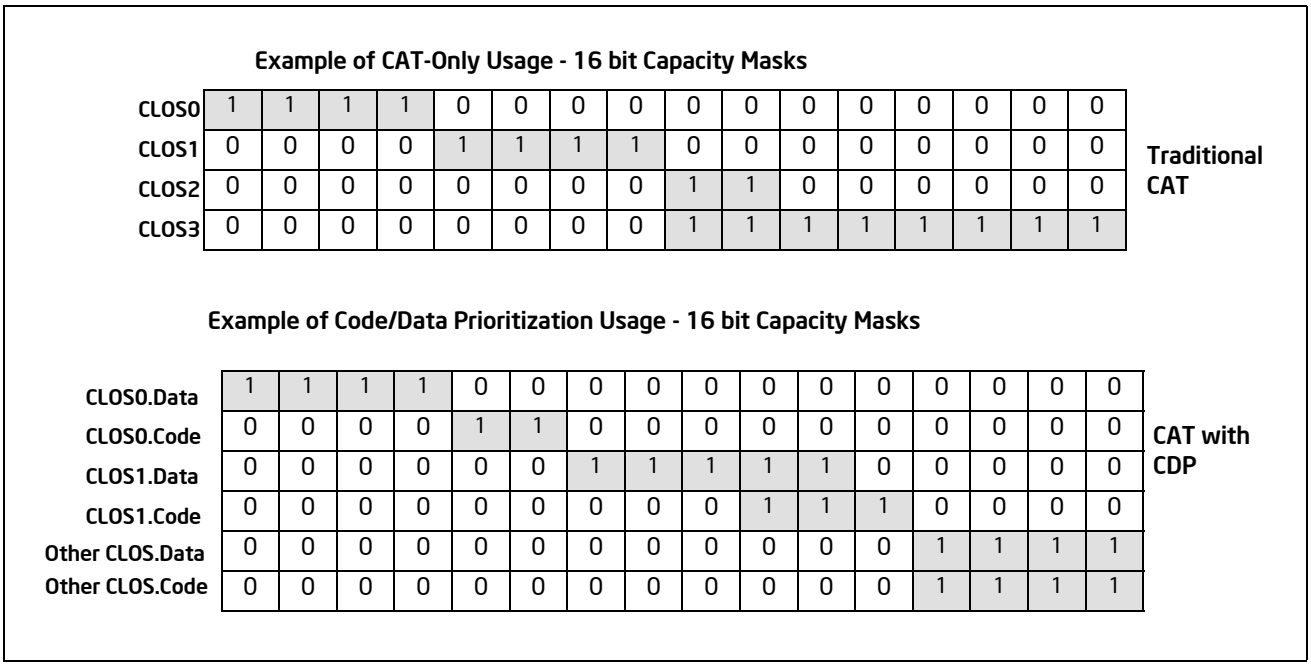


Figure 18-29. Code and Data Capacity Bitmasks of CDP

When CDP is enabled, the existing mask space for CAT-only operation is split. As an example if the system supports 16 CAT-only CLOS, 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 CLOS is reduced by half. Code/Data masks are defined per-CLOS 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.

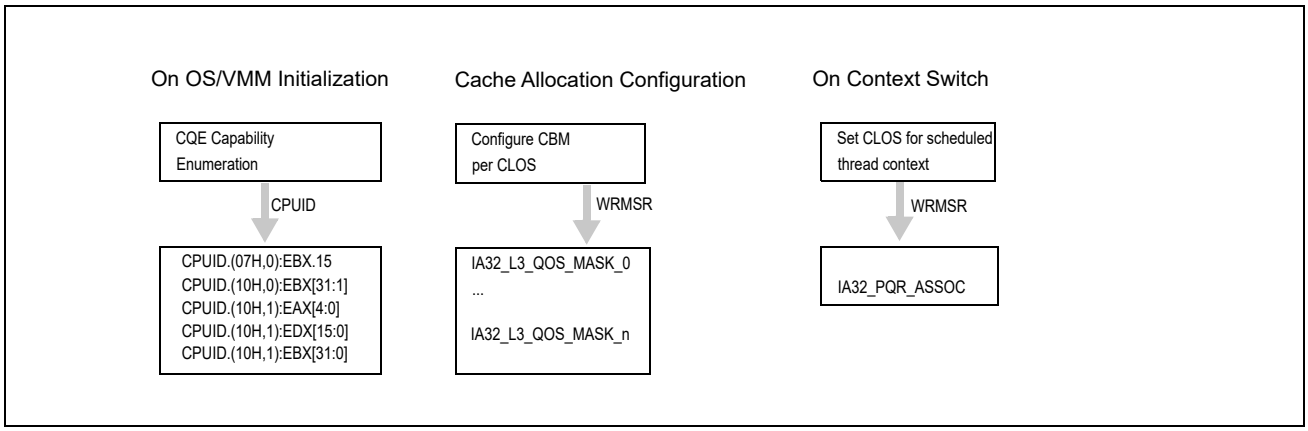


Figure 18-30. Cache Allocation Technology Usage Flow

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 CLOS, 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 CLOS field that OS/VMM can use to assign a logical processor to an available CLOS. The set of CLOS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software CLOS 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).

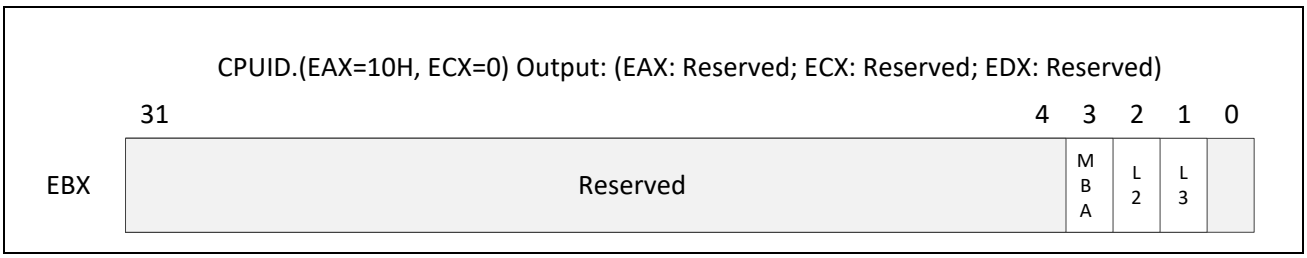


Figure 18-31. CPUID.(EAX=10H, ECX=0H) Available Resource Type Identification

— For ECX>0, EAX[4:0] reports the length of the capacity bitmask (ECX=1 or 2 for L3 CAT or L2 CAT respectively). Add one to the return value to get the result, 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.

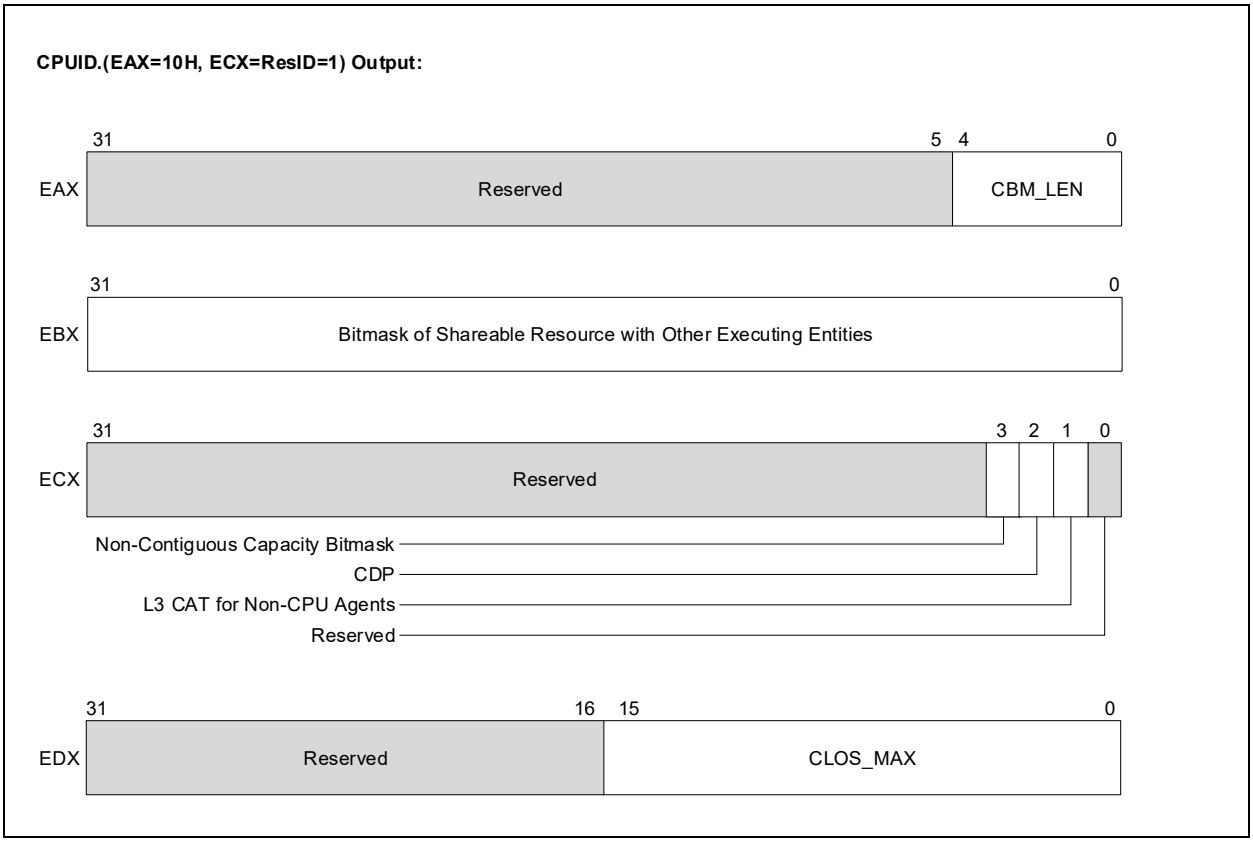


Figure 18-32. L3 Cache Allocation Technology and CDP Enumeration

- 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:
  — CPUID.(EAX=10H, ECX=ResID=1):EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, 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[bit 1]: If 1, indicates L3 CAT for non-CPU agents is supported. Bits 0 and 31:4 of ECX are reserved. See section 18.20 for details.

— 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). Bits 0 and 31:4 of ECX are reserved.

— CPUID.(EAX=10H, ECX=1):ECX[bit 3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous. Bits 0 and 31:4 of ECX are reserved.

— CPUID.(EAX=10H, ECX=1):EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

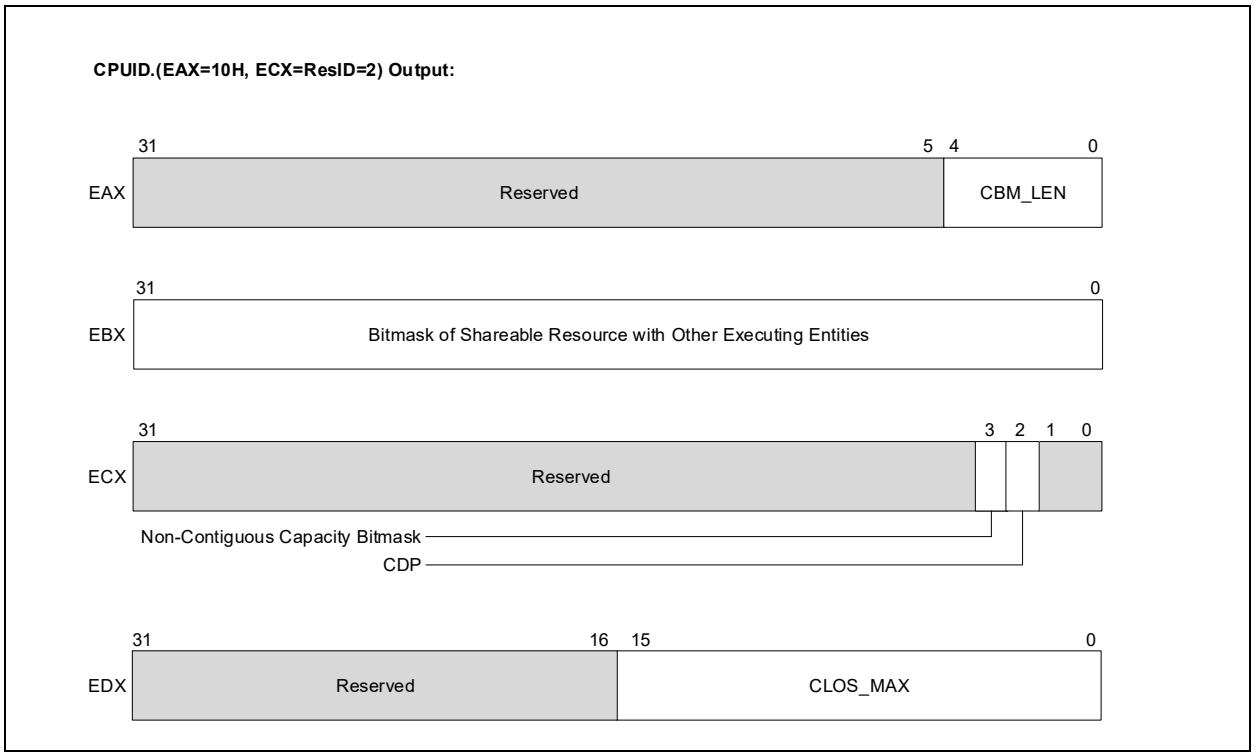


Figure 18-33. L2 Cache Allocation Technology

- 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:

  — CPUID.(EAX=10H, ECX=ResID=2):EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, 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). Bits 1:0 and 31:4 of ECX are reserved.

  — CPUID.(EAX=10H, ECX=2):ECX[bit 3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits which are set in the various IA32_L2_MASK_n registers do not have to be contiguous. Bits 1:0 and 31:4 of ECX are reserved.

— CPUID.(EAX=10H, ECX=2):EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

A note on migration of Classes of Service (CLOS): Software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the performance of the Cache Allocation Technology feature may result if CLOS 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 CLOS 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 CLOS supported using CPUID (see Section 18.19.4.2), each CLOS 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 CLOS, 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 resource type 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.

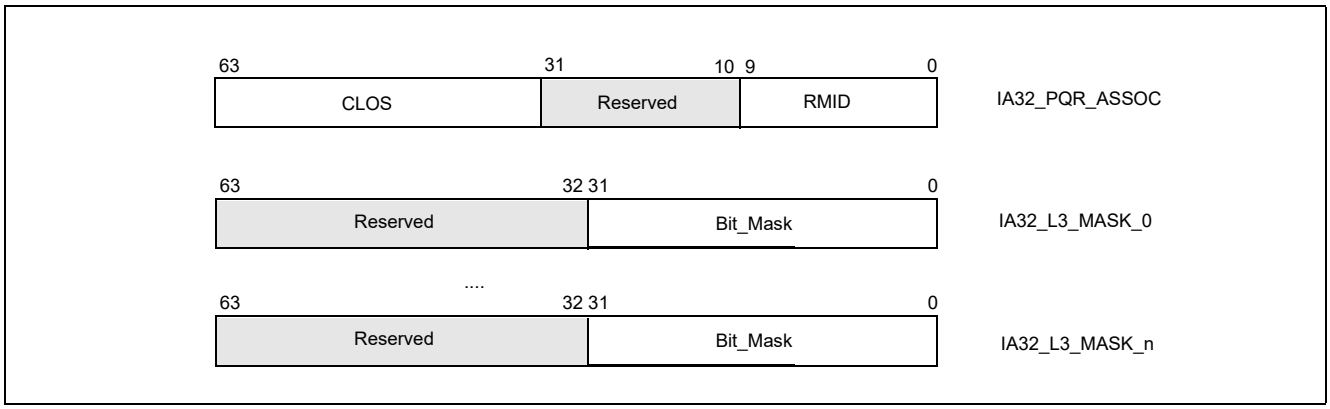


Figure 18-34. IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- 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.

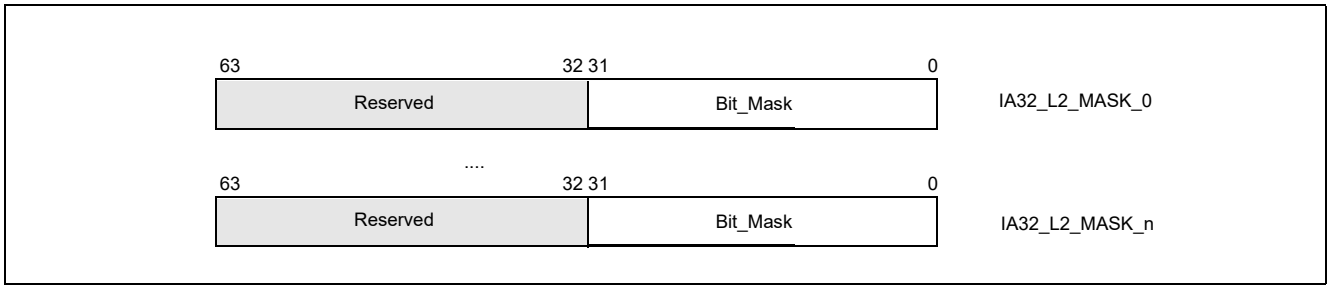


Figure 18-35. IA32_L2_MASK_n MSRs

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 (CLOS) using the IA32_PQR_ASSOC MSR as described in Section 18.19.4.4, "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.CLOS 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 CLOS 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 CLOS value in IA32_PQR_ASSOC.CLOS greater than MAX_CLOS_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.CLOS is never written then the CAT capability defaults to using CLOS 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 CLOS 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.CLOS_MAX_CAT specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=1):EDX.CLOS_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.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

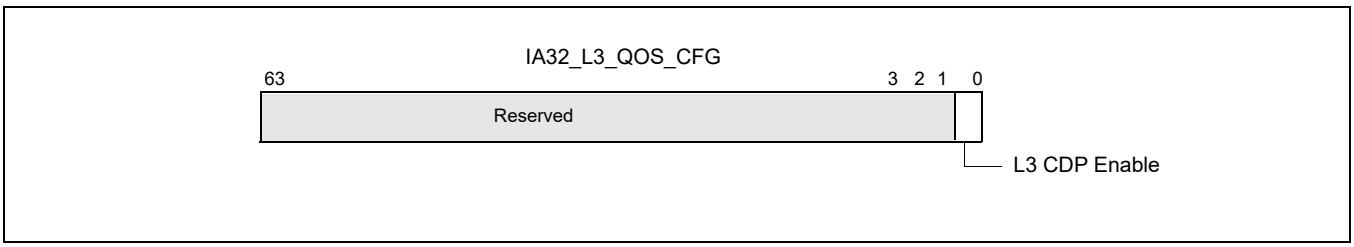


Figure 18-36. Layout of IA32_L3_QOS_CFG

IA32_L3_QOS_CFG default values are all 0s at RESET, the mask MSRs are all 1s. Hence, all logical processors are initialized in CLOS0 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 CLOS. The re-mapping is shown in Table 18-19.

Table 18-19. Re-indexing of CLOS Numbers and Mapping to CAT/CDP Mask MSRs

Mask MSR	CAT-only Operation	CDP Operation
IA32_L3_QOS_Mask_0	CLOS0	CLOS0.Data
IA32_L3_QOS_Mask_1	CLOS1	CLOS0.Code
IA32_L3_QOS_Mask_2	CLOS2	CLOS1.Data
IA32_L3_QOS_Mask_3	CLOS3	CLOS1.Code
IA32_L3_QOS_Mask_4	CLOS4	CLOS2.Data
IA32_L3_QOS_Mask_5	CLOS5	CLOS2.Code
….	….	….
IA32_L3_QOS_Mask_’2n’	CLOS’2n’	CLOS’n’.Data
IA32_L3_QOS_Mask_’2n+1’	CLOS’2n+1’	CLOS’n’.Code

One can derive the MSR address for the data mask or code mask for a given CLOS 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 CLOS 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, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type 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.CLOS_MAX_CAT specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=2):EDX.CLOS_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.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

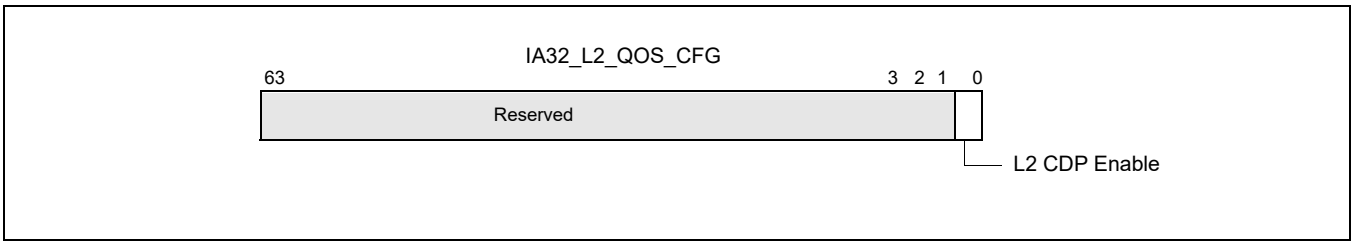


Figure 18-37. Layout of IA32_L2_QOS_CFG

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 CLOS0 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 CLOS. This remapping is the same as the remapping shown in Table 18-19 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 CLOS 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, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type 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 CLOS number which are valid in CDP operation is used, otherwise undefined behavior may result. For instance in a case with 16 CAT CLOS, since CLOS are reduced by half when CDP is enabled, software should ensure that only CLOS 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 CLOS 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 CLOS[0] before enabling or disabling CDP.

18.19.6.3 Cache Allocation Technology Dynamic Configuration

All Intel Resource Director Technology (Intel 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 CLOS (the max CLOS number is specified in CPUID.(EAX=10H, ECX=ResID):EDX[15:0]), or
- Writing a CLOS 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 CLOS value in IA32_PQR_ASSOC.CLOS outside of the lower half of the CLOS 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 CLOS on the core will be returned.

When reading an IA32_*resourceType*_MASK_*n* register the current capacity bit mask for CLOS 'n' will be returned.

As noted previously, software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if CLOS 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 CLOS 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 CLOS with a dedicated partition in the cache. Such an SMM handler can switch to the dedicated CLOS immediately upon entering SMM, and switching back to the previously running CLOS 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 CLOS concept applies to both CAT and CDP (for instance, CLOS[5] means the same thing whether CAT or CDP

is in use, and the CLOS has associated resource usage constraint attributes including cache capacity masks). The mapping of CLOS 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 CLOS 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 CLOS 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. It was introduced in the Intel Xeon Scalable Processor Family. This feature provides a method to control applications that may be over-utilizing bandwidth relative to their priority in environments such as the data-center.

The MBA feature uses existing constructs from the Intel 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. Certain generations include advanced hardware controllers for efficiency. For more information, refer to the "Intel® Resource Director Technology Architecture Specification."

The following sections describe CPU interfaces to Memory Bandwidth Allocation, such as CPUID enumeration and configuration interfaces (MSRs).

18.19.7.1 Memory Bandwidth Allocation Enumeration

Similar to other Intel 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 Intel 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 18-31, 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 18-38 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 Intel 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 Intel 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.

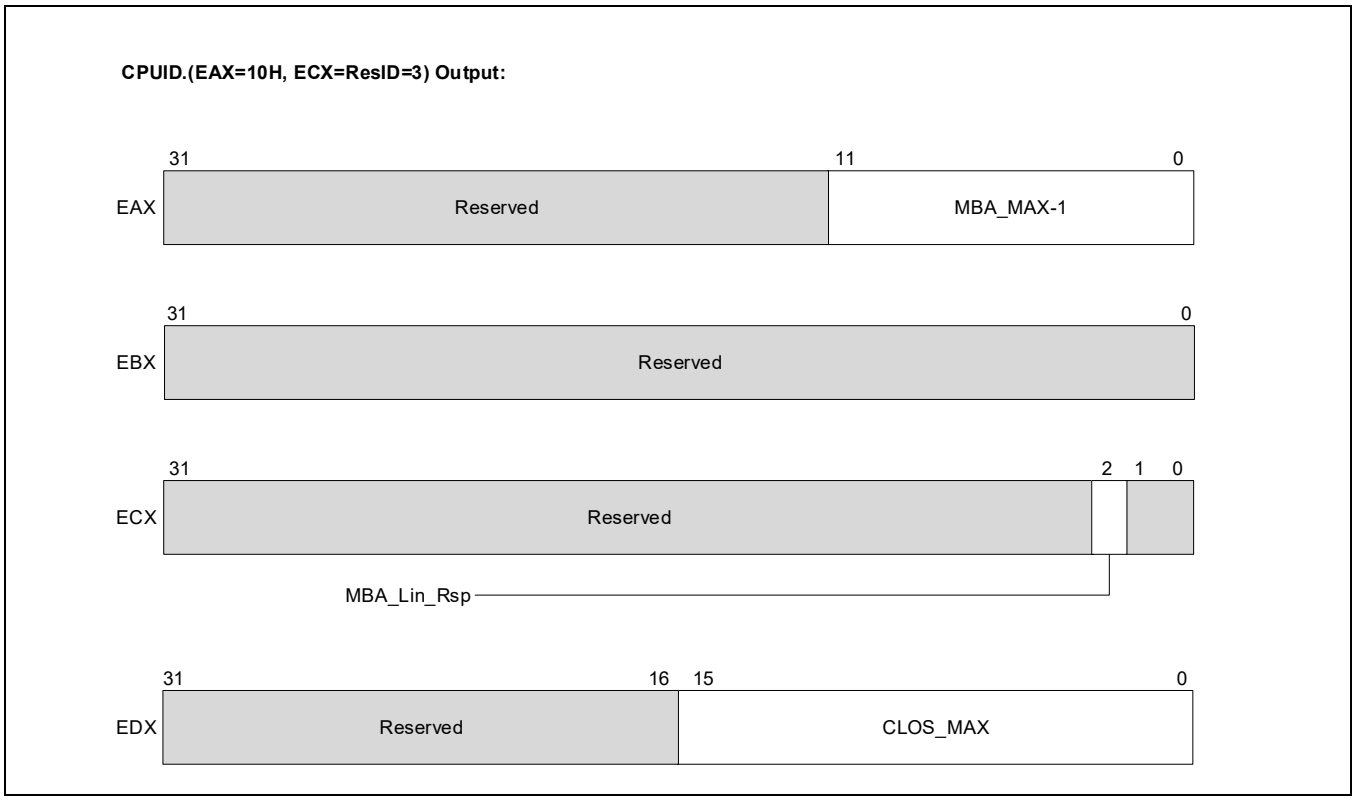


Figure 18-38. CPUID.(EAX=10H, ECX=3H) MBA Feature Details Identification

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 Intel 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_B-W_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_'CLOS_MAX'	D50H + CLOS_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.

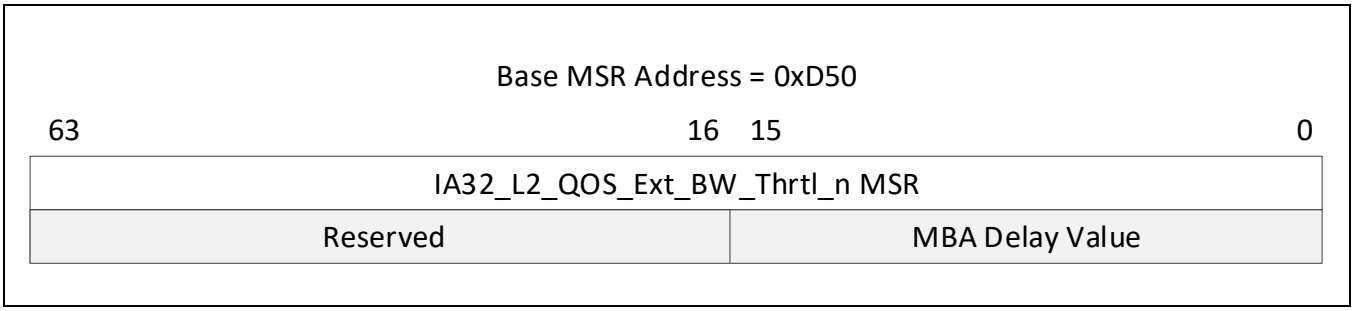


Figure 18-39. IA32_L2_QoS_Ext_BW_Thrtl_n MSR Definition

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

Different versions of Memory Bandwidth Allocation have various usage considerations and improving efficiency over time. See the "Intel® Resource Director Technology Architecture Specification" for additional details.

18.20 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FOR NON-CPU AGENTS

This section describes Intel RDT features for non-CPU agents. CPU agents are threads running on IA cores. Non-CPU agents include PCIe and CXL devices and integrated accelerators, thus broadly encompassing the set of agents that read from and write to either caches or memory, excluding IA cores. The non-CPU agent Intel RDT features enable monitoring of I/O device shared cache and memory bandwidth and cache allocation control. This provides features for I/O devices equivalent to the CPU agent Intel RDT capabilities CMT, MBM, and CAT (discussed in Section 18.18 and Section 18.19). Refer to the "Intel® Resource Director Technology Architecture Specification" regarding design goals, use cases, software architecture, ACPI enumeration, and MMIO register interfaces.

"Non-CPU agent Intel RDT" refers to capabilities that monitor and control non-CPU agents' resource utilization, including PCIe and CXL devices and integrated accelerators. Non-CPU agent Intel RDT may be called I/O RDT in some literature. In this document, the term "non-CPU agent Intel RDT" is used.

18.20.1 Non-CPU Agent Intel® RDT Features Enumeration Details

CPU agent Intel RDT features use the CPUID instruction to enumerate supported features and the level of support. Architectural Model-Specific Registers (MSRs) are interfaces to the monitoring and allocation features, as described in Sections 18.18 and 18.19.

Non-CPU agent Intel RDT builds on CPU agent Intel RDT by extending CPUID to indicate the presence and integration of non-CPU agent Intel RDT and by providing rich enumeration information in vendor-specific extensions to the Advanced Configuration and Power Interface (ACPI), in particular in the I/O RDT (IRDT) table. The ACPI extensions detailed in the "Intel® Resource Director Technology Architecture Specification" provide mechanisms to comprehend the structure of devices attached behind I/O blocks to particular links and what forms of tagging are supported on a per-link basis.

It is recommended that software parse CPUID and ACPI to obtain a detailed understanding of platform support and capabilities before attempting to use non-CPU agent Intel RDT.

18.20.1.1 CPUID-Based Enumeration for Non-CPU Agent Intel® RDT Feature

CPUID-based enumeration provides a method by which all architectural Intel RDT features may be enumerated.

For CPU agent Intel RDT, monitoring details are enumerated in a CPUID sub-leaf denoted as CPUID.(EAX=0FH, ECX=ResID), where ResID corresponds to a resource ID bit index from the CPUID.(EAX=0FH, ECX=0) sub-leaf. Similarly, Intel RDT allocation features are described in CPUID.(EAX=10H, ECX=ResID). (Note that the ResID bit positions are not guaranteed to be symmetric or have the same encodings.)

No CPUID leaves or sub-leaves are created for non-CPU agent Intel RDT. Rather, non-CPU agent Intel RDT extends the existing Intel RDT CPUID sub-leaves with a bit per resource type, indicating whether non-CPU agent Intel RDT monitoring or control is present. CPUID.(EAX=0FH, ECX=ResID=1):EAX[bits 9, 10] represents the presence of CMT and MBM features for non-CPU agents. CPUID.(EAX=10H, ECX=ResID=1):ECX[bit 1] represents the presence of the CAT feature for non-CPU agents.

Specifically, for non-CPU Agent Intel RDT Monitoring (see Figure 18-21):

- Bits are added in the CPU Agent Intel RDT CMT/MBM leaf: CPUID.(EAX=0FH, ECX=ResID=1):EAX[bits 9, 10].
  - EAX[bit 9]: If set, indicates the presence of non-CPU Agent Cache Occupancy Monitoring (the equivalent of CPU Agent Intel RDT's CMT feature).
  - EAX[bit 10]: If set, indicates the presence of non-CPU Agent memory L3 external BW monitoring (the equivalent of CPU Agent Intel RDT's MBM feature).

For non-CPU Agent Intel RDT Allocation (see Figure 18-32):

- New bit in L3 CAT leaf: CPUID.(EAX=10H, ECX=ResID=1):ECX[bit 1].
  - ECX[bit 1]: If set, indicates the presence of non-CPU Agent Cache Allocation Technology (the equivalent of CPU Agent Intel RDT's L3 CAT feature).
- As before, ECX[bit 2] indicates that L3 CDP is supported if set.

Note that no equivalent bits are defined in CPUID.(EAX=10H, ECX=ResID=2) as there is no ability for devices to fill into core L2 caches.

If any of these non-CPU agent Intel RDT enumeration bits are set, indicating that a monitoring feature or allocation feature is present, it also indicates the presence of the IA32_L3_IO_RDT_CFG architectural MSR. This MSR may be used to enable the non-CPU agent Intel RDT features. See Section 18.20.2 for MSR details.

The presence of Intel RDT is a prerequisite for using the equivalent non-CPU agent Intel RDT feature. If a particular CPU agent Intel RDT feature is absent, any attempt to use non-CPU agent Intel RDT equivalents will result in general protection faults in the MSR interface. Attempts to enable unsupported features in the I/O complex will result in writes to the corresponding MMIO enable or configuration interfaces being ignored.

Software may use the existing CPUID leaves to gather the maximum number of RMID and CLOS tags for each resource level (e.g., L3 cache), and non-CPU agent Intel RDT is also subject to these limits.

Some platforms may support a mix of features, for instance, supporting L3 CAT architectural controls and the non-CPU agent Intel RDT equivalent, but no CMT/MBM monitoring or non-CPU agent monitoring equivalent, and these capabilities should be enumerated on a per-platform basis.

18.20.1.2 ACPI Enumeration

When support for non-CPU agent Intel RDT features is detected using CPUID, ACPI may be consulted for further details on the level of feature support, device structures behind various I/O ports, and the specific MMIO interfaces used to control a given device.

Non-CPU agent Intel RDT enumeration is via the “IRDT” ACPI table. For more information, refer to the “Intel® Resource Director Technology Architecture Specification.”

18.20.2 Non-CPU Agent Intel® RDT Feature Enable MSR

Before configuring non-CPU agent Intel RDT through MMIO, the feature should be enabled using the non-CPU agent Intel RDT Feature Enable MSR, IA32_L3_IO_RDT_CFG (MSR address 0C83H). As described in Section 18.20.1.1, the presence of one or more CPUID bits indicating support for one or more non-CPU agent Intel RDT features also indicates the presence of this MSR. This MSR may be used to enable the non-CPU agent Intel RDT features.

Two bits are defined in this MSR. Bit 0, when set, enables non-CPU agent RDT resource allocation features. Bit 1, when set, enables non-CPU agent Intel RDT monitoring features.

The L3 Non-CPU agent Intel RDT Monitoring Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource monitoring features are present.

The L3 Non-CPU agent Intel RDT Allocation Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource allocation features are present.

The default value is 0x0, so both classes of features are disabled by default. All bits not defined are reserved. Writing a non-zero value to any reserved bit will generate a General Protection Fault (#GP(0)).

This MSR is scoped at the L3 cache level and is cleared on system reset. It is expected that the software will configure this MSR consistently across all L3 caches that may be present on that package.

The definition of the IA32_L3_IO_RDT_CFG MSR is shown in Figure 18-40.

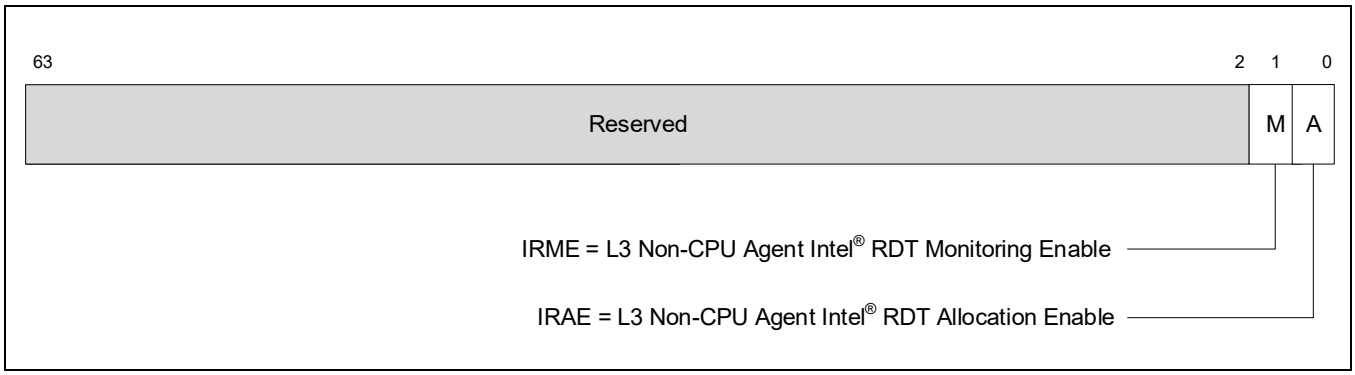


Figure 18-40. Layout of the IA32_L3_IO_RDT_CFG MSR for Enabling Non-CPU Agent Intel® RDT

12. Updates to Chapter 25, Volume 3C

Change bars and violet 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:

- Updated Section 25.6.2, "Processor-Based VM-Execution Controls," to indicate that bit 9 (INVLPG exiting) also applies to INVPCID.

CHAPTER 25
VIRTUAL MACHINE CONTROL STRUCTURES

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.

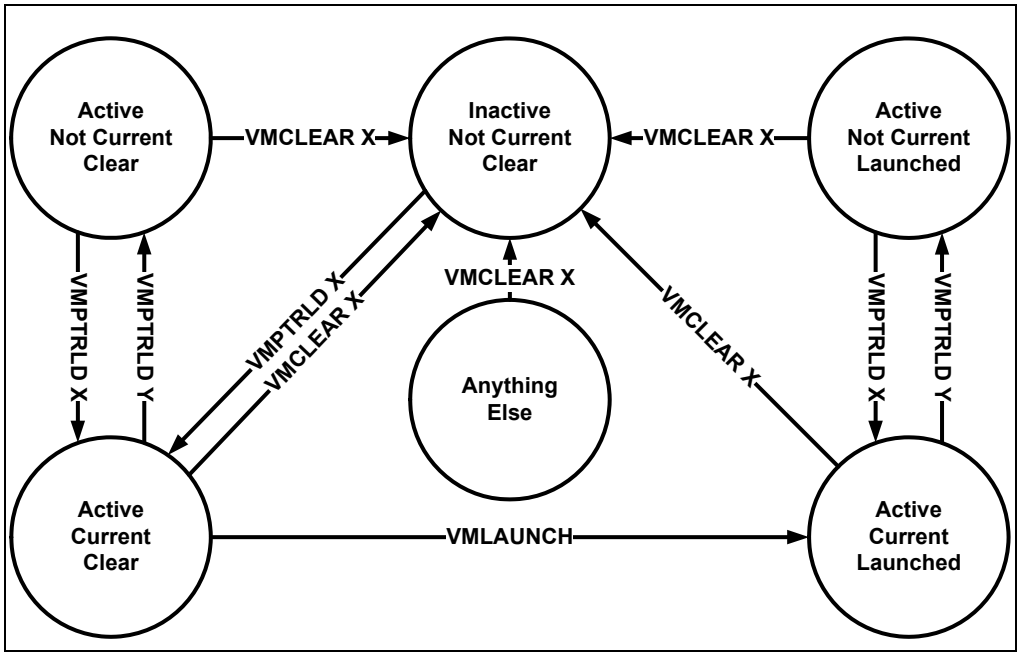


Figure 25-1. States of VMCS X

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_CTLS2 (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 type[3]. 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_PIN-BASED_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 and INVPCID cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC 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 unpaged 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_PROC-BASED_CTLS2 (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.
7	Virtualize IA32_SPEC_CTRL	If this control is 1, the operation of the RDMSR and WRMSR instructions is changed when accessing the IA32_SPEC_CTRL MSR. See Section 26.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROC-BASED_CTLS3 (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 the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).[1]
- If the local APIC is in x2APIC mode, it can accesses the local APIC's registers using the RDMSR and WRMSR instructions (see the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).
- 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:

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

— 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.

- **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 toC0001FFFH. 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 toC0001FFFH. 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 paging-structure (an EPT PML4 table with 4-level EPT and an EPT PML5 table with 5-level EPT)[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.

25.6.26 Fields Controlling Virtualization of the IA32_SPEC_CTRL MSR

On processors that support the 1-setting of the "virtualize IA32_SPEC_CTRL" VM-execution control, the VM-execution control fields include the following 64-bit fields:

- **IA32_SPEC_CTRL mask.** Setting a bit in this field prevents guest software from modifying the corresponding bit in the IA32_SPEC_CTRL MSR.
- **IA32_SPEC_CTRL shadow.** This field contains the value that guest software expects to be in the IA32_SPEC_CTRL MSR.

Section 26.3 discusses how these fields are used in VMX non-root operation.

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: ▪ 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).

Table 25-13. Definitions of Primary VM-Exit Controls (Contd.)

Bit Position(s)	Name	Description
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.
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 CR0.PG and IA32_EFER.LME, and since CR0.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 entries and VM exits 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.

Table 25-14 lists the secondary VM-exit controls. See Chapter 28 for more details of how these controls affect VM exits.

Table 25-14. Definitions of Secondary VM-Exit Controls

Bit Position(s)	Name	Description
3	Prematurely busy shadow stack	If this control is 1, VM exits that cause a shadow stack to become prematurely busy (see Section 17.2.3, "Supervisor Shadow Stack Token," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1) indicate this fact and save additional information into the VMCS.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_EX-IT_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 25-15. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 25-15. 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:

- **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 25-15). 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 25-16 lists the controls supported. See Chapter 25 for how these controls affect VM entries.

Table 25-16. 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.

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).

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. Definitions of VM-Entry Controls (Contd.)

Bit Position(s)	Name	Description
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.
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 25-15. 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

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).

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 25-17 describes the field.

Table 25-17. 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 *n*) 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.

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*.

On some processors, attempts to write to these fields with VMWRITE fail (see "VMWRITE—Write Field to Virtual-Machine Control Structure" in Chapter 31).[1]

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 25-18.

Table 25-18. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason.
16	Always cleared to 0.
24:17	Not currently defined.
25	A VM exit saves this bit as 1 to indicate that the VM exit caused a shadow stack to become prematurely busy.
26	A VM exit saves this bit as 1 to indicate that the VM exit occurred after assertion of a bus lock while the "VMM bus-lock detection" VM-execution control was 1.
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 25 is set to 1 if the "prematurely busy shadow stack" VM-exit control is 1 and the VM exit caused a shadow stack to become prematurely busy (see Section 26.4.3). Otherwise, the bit is cleared.
  - Bit 26 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 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-

1. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

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 25-19 describes this field.

Table 25-19. 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:

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.

- **IDT-vectoring information** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 25-20 describes this field.

Table 25-20. 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
11	Error code valid (0 = invalid; 1 = valid)
30:12	Not currently defined
31	Valid

- **IDT-vectoring error code** (32 bits). For VM exits the 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).

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).

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).
- 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

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.

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).

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 25-21.

Table 25-21. 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:

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).

- 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.[1,2]

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. 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, 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 violet 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:

- Updated Section 26.2, "Other Causes of VM Exits," to clarify that conditions necessary for some of the VM exits identified in this section may hold immediately after VM entry. Previously, this statement was tied to only two causes, though it can be applied to several. Additionally, an incorrect cross-reference was corrected under the "External interrupts" item.

CHAPTER 26
VMX NON-ROOT OPERATION

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, ENQCMDS.** 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 1and 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 30.6 for an exception.) Otherwise, the processor handles the interrupt is 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).

  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).

  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.

Conditions necessary for some of the VM exits identified in this section may hold immediately after VM entry. If they do, a corresponding VM exit occurs at that time.

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 ENQCMDS 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 ENQCMDS), 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 contains 48H (indicating the IA32_SPEC_CTRL MSR), the value returned by the instruction is determined by the setting of the "virtualize IA32_SPEC_CTRL" VM-execution control:

- If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_SPEC_CTRL MSR.
- If the control is 1, the value returned is that of the IA32_SPEC_CTRL shadow field in the VMCS.

— 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_UMWAIT_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_UMWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UMWAIT_CONTROL,FFFFFFFCH).

      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.
- **UMWAIT.** Behavior of the UMWAIT 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, UMWAIT 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_UMWAIT_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_UMWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UMWAIT_CONTROL,FFFFFFFCH).

      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, UMWAIT 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 48H (indicating the IA32_SPEC_CTRL MSR), instruction behavior depends on the setting of the "virtualize IA32_SPEC_CTRL" VM-execution control:

  - If the control is 0, WRMSR operates normally, loading the IA32_SPEC_CTRL MSR with the value in EAX:EDX.
  - If the control is 1, instruction will attempt to write the IA32_SPEC_CTRL MSR using the instruction's source operand, but it will attempt to modify only those bits in positions corresponding to bits cleared in the IA32_SPEC_CTRL mask field in the VMCS.

    Specifically, the instruction attempts to write the MSR with the following value:

    (MSR_VAL & ISC_MASK) OR (SRC & NOT ISC_MASK),

    where MSR_VAL is the original value of the MSR, ISC_MASK is the IA32_SPEC_CTRL mask, and SRC is the instruction's source operand.

    Any fault that would result from writing that value to the MSR (e.g., due to a reserved-bit violation) occurs normally. Otherwise, the value is written to the MSR.

    Such a write to the MSR will have any side effects that would occur normally had the MSR been written with the value indicated above (including any side effects that may result from writing unchanged values to the masked bits).

    If the write completes without a fault, the unmodified value of the source operand is written to the IA32_SPEC_CTRL shadow field in the VMCS.

  — 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).

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).

— 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 certain shadow-stack updates may 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).

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.

26.4.3 Shadow-Stack Updates

As noted in Section 17.2.3, "Supervisor Shadow Stack Token," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, a switch of shadow stack may occur as part of IDT event delivery or an execution of far CALL that changes the CPL, or as part of IDT event delivery that uses the interrupt stack table (IST).

As part of the shadow-stack switch, the processor gains exclusive access to the new stack through manipulation of the **supervisor shadow stack token** located at the base of the new shadow stack. The processor reads the token and, among other things, confirms that bit 0 of the token (its **busy bit**) is 0. If the busy bit is already 1, the transition (event delivery or far CALL) cause a general-protection fault and does not complete. If the busy bit is 0, the transition sets the busy bit by writing to the token in memory. (The update is atomic with the original read of the token.)

If the transition commenced with CPL < 3, it will follow the token update by pushing three items on the new shadow stack (for the old values of the CS selector, instruction pointer, and shadow-stack pointer). As noted in Section 17.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, if any of the pushes causes a VM exit, the processor will revert to the old shadow stack and the busy bit in the new shadow stack's token remains set. The new shadow stack is said to be prematurely busy.

If the "prematurely busy shadow stack" VM-exit control is 1, a VM exit that results in a shadow stack becoming prematurely busy will indicate that fact through information saved in the VMCS. See Section 28.2.1.

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).

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.

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.

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 0[1] 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). The EPTP value in the selected entry is evaluated to determine whether it is valid for EPTP switching: a value is valid if (1) it is the same as the current EPTP value in bits 5:3 (these bits specify the EPT page-walk length); and (2) it would not cause VM entry to fail (see Section 27.2.1.1). If the value is invalid, a VM exit occurs. Otherwise, the value 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 or change EPT page-walk length)
            THEN VM exit;
            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.

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.

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 EPTRTA values, where EPTRTA 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 paging[3] 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).

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.

---

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.

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 did not cause a shadow stack to become prematurely busy (see Section 26.4.3);
- 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.

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.

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.)

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.

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)

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.
- Bits 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 unpaged 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 unpaged 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.
- 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.

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.

  - 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 violet 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:

- Canonicality requirements for VMX transitions on LDTR.base in cases when that value is undefined have been updated. As a result, Section 27.3.2.2, "Loading Guest Segment Registers and Descriptor-Table Registers," had a sub-item on page 27-17 removed. This item previously read: "On processors that support Intel 64 architecture, the base address for LDTR is set to an undefined but canonical value."

CHAPTER 27
VM ENTRIES

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. MSRs 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.ZF[1] (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 MSRs 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 MSRs.

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 must contain a value 1 less than an EPT page-walk length supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Section 29.3.2 and Appendix A.10).
  - 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 25-17 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_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.
3. If the capability MSR IA32_VMX_CR0_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_CR0_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 25-17 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.
  - If the "load UINV" VM-entry control is 1, UINV is loaded with the low 8 bits of the UINV field. UINV is represented in bits 39:32 of the IA32_UINTR_MISC MSR. The remainder of the MSR is not modified.

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.

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.

- 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, 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 EPTRTA values (EPTRTA 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 25-17), 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 a 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_CR0_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 the 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[SENTER]. 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 & ~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:

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.

    - 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.

15. Updates to Chapter 28, Volume 3C

Change bars and violet 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:

Canonicality requirements for VMX transitions on LDTR.base in cases when that value is undefined have been updated.

- As a result, Section 28.3.2, "Saving Segment Registers and Descriptor-Table Registers," had a sub-item on page 28-24 removed. This item previously read: "LDTR. The value saved for the base address is always canonical."
- The parenthetical text was removed from the paragraph on page 28-31. This paragraph previously read: "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)."

CHAPTER 28
VM EXITS

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.NT[1]; 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_MCG_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 25 is set if the "prematurely busy shadow stack" VM-exit control is 1 and the VM exit caused a shadow stack become prematurely busy (see Section 26.4.3). Otherwise, the bit is cleared.
  - 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 24:16) 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); SPP-related events (see Section 29.3.4); and instruction timeout (see Section 26.2). 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.

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
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	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

Table 28-2. Exit Qualification for Task Switches (Contd.)

Bit Position(s)	Contents
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
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."

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 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."
- 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)

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
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
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	▪ 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	Access type: 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 Other values not used
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" (see Section 28.2.3). 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

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.

reports "NMI unblocking due to IRET" (see Section 28.2.3). 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.

— If the "PASID translation" VM-execution control, PASID translation is performed for executions of the ENQCMD and ENQCMDS 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 ENQCMDS, the exit qualification contains the low 32 bits of the instruction's source operand (which had been read from memory prior to PASID translation).

— For a VM exit due to an instruction timeout (Section 26.2), bit 0 indicates (if set) that the context of the virtual machine is invalid and that the VM should not be resumed. Bit 12 of the exit qualification reports "NMI unblocking due to IRET" (see Section 28.2.3). All other bits of the exit qualification are undefined.

- **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).[2]
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.[2] 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.[3]
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).

Table 28-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
8	If bit 7 is 1: ▪ 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).
9	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations,[4] 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 CR0.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,[4] 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 CR0.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,[4] 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 CR0.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 either (1) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (2) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 29.3.2).
3. Bit 6 is cleared to 0 if (1) the "mode-based execute control" VM-execution control is 1; and (2) either (a) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (b) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 29.3.2).
4. 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.

— If the "prematurely busy shadow stack" VM-exit control is 1, certain VM exits (besides those noted above) save the linear address that pertains to the VM exit if the VM exit caused a shadow stack to become prematurely busy (see Section 26.4.3). This is true for VM exits due for these reasons: EPT misconfiguration, page-modification log-full event, and instruction timeout. (A VM exit due to instruction timeout that sets bit 0 of the exit qualification, indicating that VM context is invalid, does not save a valid linear address.)

— 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 25-19). 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).

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.

— 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.

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, SPP-related events, or instruction timeouts.

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, SPP-related events, and instruction timeouts, 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).

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).

- 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 original event was a software interrupt (INT *n*) executed in virtual-8086 mode with EFLAGS.IOPL < 3 and the VM exit was due to a general-protection exception (#GP) that occurred because either CR4.VME = 0 or bit *n* of the software interrupt redirection bit map in the TSS is set.

The following items detail the use of these fields:

- IDT-vectoring information (format given in Table 25-20). 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.

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.

— 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, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LOADIWKEY, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, PCONFIG, RDMSR, RDPMC, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, TPAUSE, UMWAIT, 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 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.

---

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.

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.
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, TPAUSE, UMWAIT, 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 or RDSEED, the field has the format is given in Table 28-12.
  - For VM exits due to attempts to execute TPAUSE or UMWAIT, the field has the format is given in Table 28-13.
  - For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 28-14.
  - For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 28-15.
  - For VM exits due to attempts to execute LOADIWKEY, the field has the format is given in Table 28-16.

  For all other VM exits, the field is undefined, unless the VM exit occurred in enclave mode, in which case the field is cleared.

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).

- **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.
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.
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

Table 28-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)

Bit Position(s)	Content
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.
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).

Table 28-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR (Contd.)

Bit Position(s)	Content
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 and RDSEED

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (destination 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)
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 TPAUSE and UMWAIT

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (source 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)
31:7	Undefined.

Table 28-14. 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.
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.

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.

Table 28-15. 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.
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)

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.

Table 28-16. 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).

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.
- 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

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.

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).

— 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 gate[1] 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:

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.

- The activity-state field is saved with the logical processor's activity state before the VM exit.[1] 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.

    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",[2] "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.

1. 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.
2. This item includes VM exits that occur as a result of certain VM entries (Section 27.7.7).

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:

- 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:

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.

- 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.

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.)

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.

    - 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.

    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.

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.

28.5.2 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.

    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 MSRs.
  - 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 FFFFFH 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.

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.

— 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.

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.

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.

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.

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 EPTRTA values (EPTRTA 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.

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. Note the following about processors that support Intel 64 architecture. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CR0.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.

1. There was a failure in saving guest MSRs (see Section 28.4).
2. Host checking of the page-directory-pointer-table entries (PDPTEs) 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).
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 the 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:[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 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:

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 Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

  - 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.

- 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 ≠ 0; otherwise, no pending user interrupt is recognized.

16. Updates to Chapter 33, Volume 3C

Change bars and violet text show changes to Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

Changes to this chapter:

- Updated Section 33.3.7, "Decoder Synchronization (PSB+)." Previously this section indicated that PSB+ can include "MODE.Exec, if PacketEn=1." This has been updated to indicate that PSB+ can include "MODE.Exec, if PacketEn=1 or (ContextEn=1 and IA32_RTIT_CTL.EventEn=1)."

CHAPTER 33
INTEL® PROCESSOR TRACE

33.1 OVERVIEW

Intel® Processor Trace (**Intel PT**) is an extension of Intel® Architecture that captures information about software execution using dedicated hardware facilities that cause only minimal performance perturbation to the software being traced. This information is collected in **data packets**. The initial implementations of Intel PT offer **control flow tracing**, which generates a variety of packets to be processed by a software decoder. The packets include timing, program flow information (e.g., branch targets, branch taken/not taken indications) and program-induced mode related information (e.g., Intel TSX state transitions, CR3 changes). These packets may be buffered internally before being sent to the memory subsystem or other output mechanism available in the platform. Debug software can process the trace data and reconstruct the program flow.

Intel Processor Trace was first introduced in Intel® processors based on Broadwell microarchitecture and Intel Atom® processors based on Goldmont microarchitecture. Later generations include additional trace sources, including software trace instrumentation using PTWRITE, and Power Event tracing.

33.1.1 Features and Capabilities

Intel PT's control flow trace generates a variety of packets that, when combined with the binaries of a program by a post-processing tool, can be used to produce an exact execution trace. The packets record flow information such as instruction pointers (IP), indirect branch targets, and directions of conditional branches within contiguous code regions (basic blocks).

Intel PT can also be configured to log software-generated packets using PTWRITE, and packets describing processor power management events. Further, Precise Event-Based Sampling (PEBS) can be configured to log PEBS records in the Intel PT trace; see Section 20.5.5.2.

In addition, the packets record other contextual, timing, and bookkeeping information that enables both functional and performance debugging of applications. Intel PT has several control and filtering capabilities available to customize the tracing information collected and to append other processor state and timing information to enable debugging. For example, there are modes that allow packets to be filtered based on the current privilege level (CPL) or the value of CR3.

Configuration of the packet generation and filtering capabilities are programmed via a set of MSRs. The MSRs generally follow the naming convention of IA32_RTIT_*. The capability provided by these configuration MSRs are enumerated by CPUID, see Section 33.3. Details of the MSRs for configuring Intel PT are described in Section 33.2.8.

33.1.1.1 Packet Summary

After a tracing tool has enabled and configured the appropriate MSRs, the processor will collect and generate trace information in the following categories of packets (for more details on the packets, see Section 33.4):

- Packets about basic information on program execution; these include:
  - Packet Stream Boundary (PSB) packets: PSB packets act as 'heartbeats' that are generated at regular intervals (e.g., every 4K trace packet bytes). These packets allow the packet decoder to find the packet boundaries within the output data stream; a PSB packet should be the first packet that a decoder looks for when beginning to decode a trace.
  - Paging Information Packet (PIP): PIPs record modifications made to the CR3 register. This information, along with information from the operating system on the CR3 value of each process, allows the debugger to attribute linear addresses to their correct application source.
  - Time-Stamp Counter (TSC) packets: TSC packets aid in tracking wall-clock time, and contain some portion of the software-visible time-stamp counter.
  - Core Bus Ratio (CBR) packets: CBR packets contain the core:bus clock ratio.

  - Mini Time Counter (MTC) packets: MTC packets provide periodic indication of the passing of wall-clock time.
  - Cycle Count (CYC) packets: CYC packets provide indication of the number of processor core clock cycles that pass between packets.
  - Overflow (OVF) packets: OVF packets are sent when the processor experiences an internal buffer overflow, resulting in packets being dropped. This packet notifies the decoder of the loss and can help the decoder to respond to this situation.
- Packets about control flow information:
  - Taken Not-Taken (TNT) packets: TNT packets track the "direction" of direct conditional branches (taken or not taken).
  - Target IP (TIP) packets: TIP packets record the target IP of indirect branches, exceptions, interrupts, and other branches or events. These packets can contain the IP, although that IP value may be compressed by eliminating upper bytes that match the last IP. There are various types of TIP packets; they are covered in more detail in Section 33.4.2.2.
  - Flow Update Packets (FUP): FUPs provide the source IP addresses for asynchronous events (interrupt and exceptions), as well as other cases where the source address cannot be determined from the binary.
  - MODE packets: These packets provide the decoder with important processor execution information so that it can properly interpret the dis-assembled binary and trace log. MODE packets have a variety of formats that indicate details such as the execution mode (16-bit, 32-bit, or 64-bit).
- Packets inserted by software:
  - PTWRITE (PTW) packets: includes the value of the operand passed to the PTWRITE instruction (see "PTWRITE—Write Data to a Processor Trace Packet" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B).
- Packets about processor power management events:
  - MWAIT packets: Indicate successful completion of an MWAIT operation to a C-state deeper than C0.0.
  - Power State Entry (PWRE) packets: Indicate entry to a C-state deeper than C0.0.
  - Power State Exit (PWRX) packets: Indicate exit from a C-state deeper than C0.0, returning to C0.
  - Execution Stopped (EXSTOP) packets: Indicate that software execution has stopped, due to events such as P-state change, C-state change, or thermal throttling.
- Packets containing groups of processor state values:
  - Block Begin Packets (BBP): Indicate the type of state held in the following group.
  - Block Item Packets (BIP): Indicate the state values held in the group.
  - Block End Packets (BEP): Indicate the end of the current group.

33.2 INTEL® PROCESSOR TRACE OPERATIONAL MODEL

This section describes the overall Intel Processor Trace mechanism and the essential concepts relevant to how it operates.

33.2.1 Change of Flow Instruction (COFI) Tracing

A basic program block is a section of code where no jumps or branches occur. The instruction pointers (IPs) in this block of code need not be traced, as the processor will execute them from start to end without redirecting code flow. Instructions such as branches, and events such as exceptions or interrupts, can change the program flow. These instructions and events that change program flow are called Change of Flow Instructions (COFI). There are three categories of COFI:

- Direct transfer COFI.
- Indirect transfer COFI.
- Far transfer COFI.

The following subsections describe the COFI events that result in trace packet generation. Table 33-1 lists branch instruction by COFI types. For detailed description of specific instructions, see the Intel® 64 and IA-32 Architectures Software Developer's Manual.

Table 33-1. COFI Type for Branch Instructions

COFI Type	Instructions
Conditional Branch	JA, JAE, JB, JBE, JC, JCXZ, JECXZ, JRCXZ, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ
Unconditional Direct Branch	JMP (E9 xx, EB xx), CALL (E8 xx)
Indirect Branch	JMP (FF /4), CALL (FF /2), RET (C3, C2 xx)
Far Transfers	INT1, INT3, INT *n*, INTO, IRET, IRETD, IRETQ, JMP (EA xx, FF /5), CALL (9A xx, FF /3), RET (CB, CA xx), SYSCALL, SYSRET, SYSENTER, SYSEXIT, VMLAUNCH, VMRESUME

33.2.1.1 Direct Transfer COFI

Direct Transfer COFI are relative branches. This means that their target is an IP whose offset from the current IP is embedded in the instruction bytes. It is not necessary to indicate target of these instructions in the trace output since it can be obtained through the source disassembly. Conditional branches need to indicate only whether the branch is taken or not. Unconditional branches do not need any recording in the trace output. There are two sub-categories:

- **Conditional Branch (Jcc, J*CXZ) and LOOP**

  To track this type of instruction, the processor encodes a single bit (taken or not taken — TNT) to indicate the program flow after the instruction.

  Jcc, J*CXZ, and LOOP can be traced with TNT bits. To improve the trace packet output efficiency, the processor will compact several TNT bits into a single packet.

- **Unconditional Direct Jumps**

  There is no trace output required for direct unconditional jumps (like JMP near relative or CALL near relative) since they can be directly inferred from the application assembly. Direct unconditional jumps do not generate a TNT bit or a Target IP packet, though TIP.PGD and TIP.PGE packets can be generated by unconditional direct jumps that toggle Intel PT enables (see Section 33.2.6).

33.2.1.2 Indirect Transfer COFI

Indirect transfer instructions involve updating the IP from a register or memory location. Since the register or memory contents can vary at any time during execution, there is no way to know the target of the indirect transfer until the register or memory contents are read. As a result, the disassembled code is not sufficient to determine the target of this type of COFI. Therefore, tracing hardware must send out the destination IP in the trace packet for debug software to determine the target address of the COFI. Note that this IP may be a linear or effective address (see Section 33.3.1.1).

An indirect transfer instruction generates a Target IP Packet (TIP) that contains the target address of the branch. There are two sub-categories:

- **Near JMP Indirect and Near Call Indirect**

  As previously mentioned, the target of an indirect COFI resides in the contents of either a register or memory location. Therefore, the processor must generate a packet that includes this target address to allow the decoder to determine the program flow.

- **Near RET**

  When a CALL instruction executes, it pushes onto the stack the address of the next instruction following the CALL. Upon completion of the call procedure, the RET instruction is often used to pop the return address off of the call stack and redirect code flow back to the instruction following the CALL.

  A RET instruction simply transfers program flow to the address it popped off the stack. Because a called procedure may change the return address on the stack before executing the RET instruction, debug software

can be misled if it assumes that code flow will return to the instruction following the last CALL. Therefore, even for near RET, a Target IP Packet may be sent.

— **RET Compression**

A special case is applied if the target of the RET is consistent with what would be expected from tracking the CALL stack. If it is assured that the decoder has seen the corresponding CALL (with "corresponding" defined as the CALL with matching stack depth), and the RET target is the instruction after that CALL, the RET target may be "compressed". In this case, only a single TNT bit of "taken" is generated instead of a Target IP Packet. To ensure that the decoder will not be confused in cases of RET compression, only RETs that correspond to CALLs which have been seen since the last PSB packet may be compressed in a given logical processor. For details, see "Indirect Transfer Compression for Returns (RET)" in Section 33.4.2.2.

33.2.1.3 Far Transfer COFI

All operations that change the instruction pointer and are not near jumps are "far transfers". This includes exceptions, interrupts, traps, TSX aborts, and instructions that do far transfers.

All far transfers will produce a Target IP (TIP) packet, which provides the destination IP address. For those far transfers that cannot be inferred from the binary source (e.g., asynchronous events such as exceptions and interrupts), the TIP will be preceded by a Flow Update packet (FUP), which provides the source IP address at which the event was taken. Table 33-23 indicates exactly which IP will be included in the FUP generated by a far transfer.

33.2.2 Software Trace Instrumentation with PTWRITE

PTWRITE provides a mechanism by which software can instrument the Intel PT trace. PTWRITE is a ring3-accessible instruction that can be passed to a register or memory variable, see "PTWRITE—Write Data to a Processor Trace Packet" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B, for details. The contents of that variable will be used as the payload for the PTW packet (see Table 33-40 "PTW Packet Definition"), inserted at the time of PTWRITE retirement, assuming PTWRITE is enabled and all other filtering conditions are met. Decode and analysis software will then be able to determine the meaning of the PTWRITE packet based on the IP of the associated PTWRITE instruction.

PTWRITE is enabled via IA32_RTIT_CTL.PTWEn[12] (see Table 33-6). Optionally, the user can use IA32_RTIT_CTL.FUPonPTW[5] to enable PTW packets to be followed by FUP packets containing the IP of the associated PTWRITE instruction. Support for PTWRITE is introduced in Intel Atom processors based on the Goldmont Plus microarchitecture.

33.2.3 Power Event Tracing

Power Event Trace is a capability that exposes core- and thread-level sleep state and power down transition information. When this capability is enabled, the trace will expose information about:

— Scenarios where software execution stops.
  - Due to sleep state entry, frequency change, or other powerdown.
  - Includes the IP, when in the tracing context.

— The requested and resolved hardware thread C-state.
  - Including indication of hardware autonomous C-state entry.

— The last and deepest core C-state achieved during a sleep session.

— The reason for C-state wake.

This information is in addition to the bus ratio (CBR) information provided by default after any powerdown, and the timing information (TSC, TMA, MTC, CYC) provided during or after a powerdown state.

Power Event Trace is enabled via IA32_RTIT_CTL.PwrEvtEn[4]. Support for Power Event Tracing is introduced in Intel Atom processors based on the Goldmont Plus microarchitecture.

33.2.4 Event Tracing

Event Trace is a capability that exposes details about the asynchronous events, when they are generated, and when their corresponding software event handler completes execution. These include:

- Interrupts, including NMI and SMI, including the interrupt vector when defined.
- Faults, exceptions including the fault vector.
  - Page faults additionally include the page fault address, when in context.
- Event handler returns, including IRET and RSM.
- VM exits and VM entries.[1]
  - VM exits include the values written to the "exit reason" and "exit qualification" VMCS fields.
- INIT and SIPI events.
- TSX aborts, including the abort status returned for the RTM instructions.
- Shutdown.

Additionally, it provides indication of the status of the Interrupt Flag (IF), to indicate when interrupts are masked.

Event Trace is enabled via IA32_RTIT_CTL.EventEn[31]. Event Trace information is conveyed in Control Flow Event (CFE) and Event Data (EVD) packets, as well as the legacy MODE.Exec packet. See Section 33.4.2 for packet details. Support for Event Trace is introduced in Intel® processors based on Gracemont microarchitecture.

33.2.5 Trace Filtering

Intel Processor Trace provides filtering capabilities, by which the debug/profile tool can control what code is traced.

33.2.5.1 Filtering by Current Privilege Level (CPL)

Intel PT provides the ability to configure a logical processor to generate trace packets only when CPL = 0, when CPL > 0, or regardless of CPL.

CPL filtering ensures that no IPs or other architectural state information associated with the filtered CPL can be seen in the log. For example, if the processor is configured to trace only when CPL > 0, and software executes SYSCALL (changing the CPL to 0), the destination IP of the SYSCALL will be suppressed from the generated packet (see the discussion of TIP.PGD in Section 33.4.2.5).

It should be noted that CPL is always 0 in real-address mode and that CPL is always 3 in virtual-8086 mode. To trace code in these modes, filtering should be configured accordingly.

When software is executing in a non-enabled CPL, ContextEn is cleared. See Section 33.2.6.1 for details.

33.2.5.2 Filtering by CR3

Intel PT supports a CR3-filtering mechanism by which the generation of packets containing architectural states can be enabled or disabled based on the value of CR3. A debugger can use CR3 filtering to trace only a single application without context switching the state of the RTIT MSRs. For the reconstruction of traces from software with multiple threads, debug software may wish to context-switch for the state of the RTIT MSRs (if the operating system does not provide context-switch support) to separate the output for the different threads (see Section 33.3.5, "Context Switch Consideration").

To trace for only a single CR3 value, software can write that value to the IA32_RTIT_CR3_MATCH MSR, and set IA32_RTIT_CTL.CR3Filter. When CR3 value does not match IA32_RTIT_CR3_MATCH and IA32_RTIT_CTL.CR3Filter is 1, ContextEn is forced to 0, and packets containing architectural states will not be generated. Some other packets can be generated when ContextEn is 0; see Section 33.2.6.3 for details. When CR3 does match IA32_RTIT_CR3_MATCH (or when IA32_RTIT_CTL.CR3Filter is 0), CR3 filtering does not force ContextEn to 0 (although it could be 0 due to other filters or modes).

1. Logging of VMX transitions depends on VMCS configuration, see Section 33.5.1.

CR3 matches IA32_RTIT_CR3_MATCH if the two registers are identical for bits 63:12, or 63:5 when in PAE paging mode; the lower 5 bits of CR3 and IA32_RTIT_CR3_MATCH are ignored. CR3 filtering is independent of the value of CR0.PG.

When CR3 filtering is in use, PIP packets may still be seen in the log if the processor is configured to trace when CPL = 0 (IA32_RTIT_CTL.OS = 1). If not, no PIP packets will be seen.

33.2.5.3 Filtering by IP

Trace packet generation with configurable filtering by IP is supported if CPUID.(EAX=14H, ECX=0):EBX[bit 2] = 1. Intel PT can be configured to enable the generation of packets containing architectural states only when the processor is executing code within certain IP ranges. If the IP is outside of these ranges, generation of some packets is blocked.

IP filtering is enabled using the ADDRn_CFG fields in the IA32_RTIT_CTL MSR (Section 33.2.8.2), where the digit 'n' is a zero-based number that selects which address range is being configured. Each ADDRn_CFG field configures the use of the register pair IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B (Section 33.2.8.5). IA32_RTIT_ADDRn_A defines the base and IA32_RTIT_ADDRn_B specifies the limit of the range in which tracing is enabled. Thus each range, referred to as the ADDRn range, is defined by [IA32_RTIT_ADDRn_A, IA32_RTIT_ADDRn_B]. There can be multiple such ranges, software can query CPUID (Section 33.3.1) for the number of ranges supported on a processor.

Default behavior (ADDRn_CFG=0) defines no IP filter range, meaning FilterEn is always set. In this case code at any IP can be traced, though other filters, such as CR3 or CPL, could limit tracing. When ADDRn_CFG is set to enable IP filtering (see Section 33.3.1), tracing will commence when a taken branch or event is seen whose target address is in the ADDRn range.

While inside a tracing region and with FilterEn is set, leaving the tracing region may only be detected once a taken branch or event with a target outside the range is retired. If an ADDRn range is entered or exited by executing the next sequential instruction, rather than by a control flow transfer, FilterEn may not toggle immediately. See Section 33.2.6.5 for more details on FilterEn.

Note that these address range base and limit values are inclusive, such that the range includes the first and last instruction whose first instruction byte is in the ADDRn range.

Depending upon processor implementation, IP filtering may be based on linear or effective address. This can cause different behavior between implementations if CSbase is not equal to zero or in real mode. See Section 33.3.1.1 for details. Software can query CPUID to determine filters are based on linear or effective address (Section 33.3.1).

Note that some packets, such as MTC (Section 33.3.7) and other timing packets, do not depend on FilterEn. For details on which packets depend on FilterEn, and hence are impacted by IP filtering, see Section 33.4.1.

TraceStop

The ADDRn ranges can also be configured to cause tracing to be disabled upon entry to the specified region. This is intended for cases where unexpected code is executed, and the user wishes to immediately stop generating packets in order to avoid overwriting previously written packets.

The TraceStop mechanism works much the same way that IP filtering does, and uses the same address comparison logic. The TraceStop region base and limit values are programmed into one or more ADDRn ranges, but IA32_RTIT_CTL.ADDRn_CFG is configured with the TraceStop encoding. Like FilterEn, TraceStop is detected when a taken branch or event lands in a TraceStop region.

Further, TraceStop requires that TriggerEn=1 at the beginning of the branch/event, and ContextEn=1 upon completion of the branch/event. When this happens, the CPU will set IA32_RTIT_STATUS.Stopped, thereby clearing TriggerEn and hence disabling packet generation. This may generate a TIP.PGD packet with the target IP of the branch or event that entered the TraceStop region. Finally, a TraceStop packet will be inserted, to indicate that the condition was hit.

If a TraceStop condition is encountered during buffer overflow (Section 33.3.8), it will not be dropped, but will instead be signaled once the overflow has resolved.

Note that a TraceStop event does not guarantee that all internally buffered packets are flushed out of internal buffers. To ensure that this has occurred, the user should clear TraceEn.

To resume tracing after a TraceStop event, the user must first disable Intel PT by clearing IA32_RTIT_CTL.TraceEn before the IA32_RTIT_STATUS.Stopped bit can be cleared. At this point Intel PT can be reconfigured, and tracing resumed.

Note that the IA32_RTIT_STATUS.Stopped bit can also be set using the ToPA STOP bit. See Section 33.2.7.2.

IP Filtering Example

The following table gives an example of IP filtering behavior. Assume that IA32_RTIT_ADDRn_A = the IP of RangeBase, and that IA32_RTIT_ADDRn_B = the IP of RangeLimit, while IA32_RTIT_CTL.ADDRn_CFG = 0x1 (enable ADDRn range as a FilterEn range).

Table 33-2. IP Filtering Packet Example

Code Flow	Packets

`Bar:`
`  jmp RangeBase // jump into filter range`
`RangeBase:`
`  jcc Foo // not taken`
`  add eax, 1`
`Foo:`
`  jmp RangeLimit+1 // jump out of filter range`
`RangeLimit:`
`  nop`
`  jcc Bar`
	TIP.PGE(RangeBase)
	TNT(0)
	TIP.PGD(RangeLimit+1)

IP Filtering and TraceStop

It is possible for the user to configure IP filter range(s) and TraceStop range(s) that overlap. In this case, code executing in the non-overlapping portion of either range will behave as would be expected from that range. Code executing in the overlapping range will get TraceStop behavior.

33.2.6 Packet Generation Enable Controls

Intel Processor Trace includes a variety of controls that determine whether a packet is generated. In general, most packets are sent only if Packet Enable (**PacketEn**) is set. PacketEn is an internal state maintained in hardware in response to software configurable enable controls, PacketEn is not visible to software directly. The relationship of PacketEn to the software-visible controls in the configuration MSRs is described in this section.

33.2.6.1 Packet Enable (PacketEn)

When PacketEn is set, the processor is in the mode that Intel PT is monitoring. PacketEn is composed of other states according to this relationship:

```
PacketEn := TriggerEn AND ContextEn AND FilterEn AND BranchEn
```

These constituent controls are detailed in the following subsections.

PacketEn ultimately determines when the processor is tracing. When PacketEn is set, all control flow packets are enabled. When PacketEn is clear, no control flow packets are generated, though other packets (timing and bookkeeping packets) may still be sent. See Section 33.2.7 for details of PacketEn and packet generation.

Note that, on processors that do not support IP filtering (i.e., CPUID.(EAX=14H, ECX=0):EBX[bit 2] = 0), FilterEn is treated as always set.

33.2.6.2 Trigger Enable (TriggerEn)

Trigger Enable (**TriggerEn**) is the primary indicator that trace packet generation is active. TriggerEn is set when IA32_RTIT_CTL.TraceEn is set, and cleared by any of the following conditions:

- TraceEn is cleared by software.
- A TraceStop condition is encountered and IA32_RTIT_STATUS.Stopped is set.
- IA32_RTIT_STATUS.Error is set due to an operational error (see Section 33.3.10).

Software can discover the current TriggerEn value by reading the IA32_RTIT_STATUS.TriggerEn bit. When TriggerEn is clear, tracing is inactive and no packets are generated.

33.2.6.3 Context Enable (ContextEn)

Context Enable (**ContextEn**) indicates whether the processor is in the state or mode that software configured hardware to trace. For example, if execution with CPL = 0 code is not being traced (IA32_RTIT_CTL.OS = 0), then ContextEn will be 0 when the processor is in CPL0.

Software can discover the current ContextEn value by reading the IA32_RTIT_STATUS.ContextEn bit. ContextEn is defined as follows:

```
ContextEn = !((IA32_RTIT_CTL.OS = 0 AND CPL = 0) OR
(IA32_RTIT_CTL.USER = 0 AND CPL > 0) OR (IS_IN_A_PRODUCTION_ENCLAVE[1]) OR
(IA32_RTIT_CTL.CR3Filter = 1 AND IA32_RTIT_CR3_MATCH does not match CR3)
```

If the clearing of ContextEn causes PacketEn to be cleared, a Packet Generation Disable (TIP.PGD) packet is generated, but its IP payload is suppressed. If the setting of ContextEn causes PacketEn to be set, a Packet Generation Enable (TIP.PGE) packet is generated.

When ContextEn is 0, control flow packets (TNT, FUP, TIP.*, MODE.*) are not generated, and no Linear Instruction Pointers (LIPs) are exposed. However, some packets, such as MTC and PSB (see Section 33.4.2.16 and Section 33.4.2.17), may still be generated while ContextEn is 0. For details of which packets are generated only when ContextEn is set, see Section 33.4.1.

The processor does not update ContextEn when TriggerEn = 0.

The value of ContextEn will toggle only when TriggerEn = 1.

33.2.6.4 Branch Enable (BranchEn)

This value is based purely on the IA32_RTIT_CTL.BranchEn value. If **BranchEn** is not set, then relevant COFI packets (TNT, TIP*, FUP, MODE.*) are suppressed. Other packets related to timing (TSC, TMA, MTC, CYC), as well as PSB, will be generated normally regardless. Further, PIP and VMCS continue to be generated, as indicators of what software is running.

33.2.6.5 Filter Enable (FilterEn)

Filter Enable indicates that the Instruction Pointer (IP) is within the range of IPs that Intel PT is configured to watch. Software can get the state of Filter Enable by a RDMSR of IA32_RTIT_STATUS.FilterEn. For details on configuration and use of IP filtering, see Section 33.2.5.3.

On clearing of FilterEn that also clears PacketEn, a Packet Generation Disable (TIP.PGD) will be generated, but unlike the ContextEn case, the IP payload may not be suppressed. For direct, unconditional branches, as well as for indirect branches (including RETs), the PGD generated by leaving the tracing region and clearing FilterEn will contain the target IP. This means that IPs from outside the configured range can be exposed in the trace, as long as they are within context.

When FilterEn is 0, control flow packets are not generated (e.g., TNT, TIP). However, some packets, such as PIP, MTC, and PSB, may still be generated while FilterEn is clear. For details on packet enable dependencies, see Section 33.4.1.

---

1. Trace packets generation is disabled in a production enclave, see Section 33.2.9.5. See the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D, about differences between a production enclave and a debug enclave.

After TraceEn is set, FilterEn is set to 1 at all times if there is no IP filter range configured by software (IA32_RTIT_CTL.ADDRn_CFG != 1, for all n), or if the processor does not support IP filtering (i.e., CPUID.(EAX=14H, ECX=0):EBX[bit 2] = 0). FilterEn will toggle only when TraceEn=1 and ContextEn=1, and when at least one range is configured for IP filtering.

33.2.7 Trace Output

Intel PT output should be viewed independently from trace content and filtering mechanisms. The options available for trace output can vary across processor generations and platforms.

Trace output is written out using one of the following output schemes, as configured by the ToPA and FabricEn bit fields of IA32_RTIT_CTL (see Section 33.2.8.2):

- A single, contiguous region of physical address space.
- A collection of variable-sized regions of physical memory. These regions are linked together by tables of pointers to those regions, referred to as Table of Physical Addresses (**ToPA**). The trace output stores bypass the caches and the TLBs, but are not serializing. This is intended to minimize the performance impact of the output.
- A platform-specific trace transport subsystem.

Regardless of the output scheme chosen, Intel PT stores bypass the processor caches by default. This ensures that they don't consume precious cache space, but they do not have the serializing aspects associated with un-cacheable (UC) stores. Software should avoid using MTRRs to mark any portion of the Intel PT output region as UC, as this may override the behavior described above and force Intel PT stores to UC, thereby incurring severe performance impact.

There is no guarantee that a packet will be written to memory or other trace endpoint after some fixed number of cycles after a packet-producing instruction executes. The only way to assure that all packets generated have reached their endpoint is to clear TraceEn and follow that with a store, fence, or serializing instruction; doing so ensures that all buffered packets are flushed out of the processor.

33.2.7.1 Single Range Output

When IA32_RTIT_CTL.ToPA and IA32_RTIT_CTL.FabricEn bits are clear, trace packet output is sent to a single, contiguous memory (or MMIO if DRAM is not available) range defined by a base address in IA32_RTIT_OUTPUT_BASE (Section 33.2.8.7) and mask value in IA32_RTIT_OUTPUT_MASK_PTRS (Section 33.2.8.8). The current write pointer in this range is also stored in IA32_RTIT_OUTPUT_MASK_PTRS. This output range is circular, meaning that when the writes wrap around the end of the buffer they begin again at the base address.

This output method is best suited for cases where Intel PT output is either:

- Configured to be directed to a sufficiently large contiguous region of DRAM.
- Configured to go to an MMIO debug port, in order to route Intel PT output to a platform-specific trace endpoint (e.g., JTAG). In this scenario, a specific range of addresses is written in a circular manner, and SoC will intercept these writes and direct them to the proper device. Repeated writes to the same address do not overwrite each other, but are accumulated by the debugger, and hence no data is lost by the circular nature of the buffer.

The processor will determine the address to which to write the next trace packet output byte as follows:

```
OutputBase[63:0] := IA32_RTIT_OUTPUT_BASE[63:0]

OutputMask[63:0] := ZeroExtend64(IA32_RTIT_OUTPUT_MASK_PTRS[31:0])

OutputOffset[63:0] := ZeroExtend64(IA32_RTIT_OUTPUT_MASK_PTRS[63:32])

trace_store_phys_addr := (OutputBase & ~OutputMask) + (OutputOffset & OutputMask)
```

Single-Range Output Errors

If the output base and mask are not properly configured by software, an operational error (see Section 33.3.10) will be signaled, and tracing disabled. Error scenarios with single-range output are:

- Mask value is non-contiguous.

  IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTablePointer value has a 0 in a less significant bit position than the most significant bit containing a 1.

- Base address and Mask are mis-aligned, and have overlapping bits set.

  IA32_RTIT_OUTPUT_BASE && IA32_RTIT_OUTPUT_MASK_PTRS[31:0] > 0.

- Illegal Output Offset

  IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset is greater than the mask value IA32_RTIT_OUTPUT_MASK_PTRS[31:0].

Also note that errors can be signaled due to trace packet output overlapping with restricted memory, see Section 33.2.7.4.

33.2.7.2 Table of Physical Addresses (ToPA)

When IA32_RTIT_CTL.ToPA is set and IA32_RTIT_CTL.FabricEn is clear, the ToPA output mechanism is utilized. The ToPA mechanism uses a linked list of tables; see Figure 33-1 for an illustrative example. Each entry in the table contains some attribute bits, a pointer to an output region, and the size of the region. The last entry in the table may hold a pointer to the next table. This pointer can either point to the top of the current table (for circular array) or to the base of another table. The table size is not fixed, since the link to the next table can exist at any entry.

The processor treats the various output regions referenced by the ToPA table(s) as a unified buffer. This means that a single packet may span the boundary between one output region and the next.

The ToPA mechanism is controlled by three values maintained by the processor:

- **proc_trace_table_base.**
  This is the physical address of the base of the current ToPA table. When tracing is enabled, the processor loads this value from the IA32_RTIT_OUTPUT_BASE MSR. While tracing is enabled, the processor updates the IA32_RTIT_OUTPUT_BASE MSR with changes to proc_trace_table_base, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_base.

- **proc_trace_table_offset.**
  This indicates the entry of the current table that is currently in use. (This entry contains the address of the current output region.) When tracing is enabled, the processor loads the value from bits 31:7 (MaskOrTableOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS into bits 27:3 of proc_trace_table_offset. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTableOffset with changes to proc_trace_table_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_offset.

- **proc_trace_output_offset.**
  This a pointer into the current output region and indicates the location of the next write. When tracing is enabled, the processor loads this value from bits 63:32 (OutputOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset with changes to proc_trace_output_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_output_offset.

Figure 33-1 provides an illustration (not to scale) of the table and associated pointers.

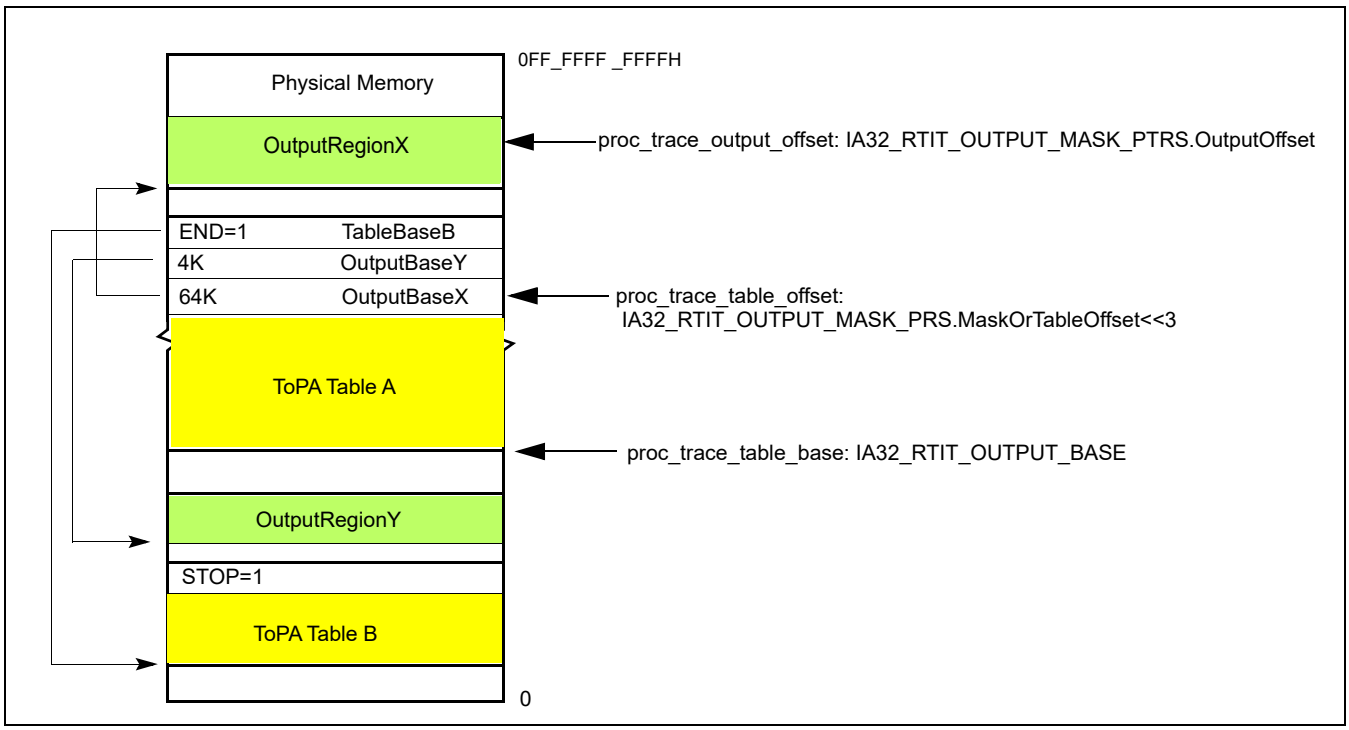


Figure 33-1. ToPA Memory Illustration

With the ToPA mechanism, the processor writes packets to the current output region (identified by proc_trace_table_base and the proc_trace_table_offset). The offset within that region to which the next byte will be written is identified by proc_trace_output_offset. When that region is filled with packet output (thus proc_trace_output_offset = RegionSize–1), proc_trace_table_offset is moved to the next ToPA entry, proc_trace_output_offset is set to 0, and packet writes begin filling the new output region specified by proc_trace_table_offset.

As packets are written out, each store derives its physical address as follows:

```
trace_store_phys_addr := Base address from current ToPA table entry +
proc_trace_output_offset
```

Eventually, the regions represented by all entries in the table may become full, and the final entry of the table is reached. An entry can be identified as the final entry because it has either the END or STOP attribute. The END attribute indicates that the address in the entry does not point to another output region, but rather to another ToPA table. The STOP attribute indicates that tracing will be disabled once the corresponding region is filled. See Table 33-3 and the section that follows for details on STOP.

When an END entry is reached, the processor loads proc_trace_table_base with the base address held in this END entry, thereby moving the current table pointer to this new table. The proc_trace_table_offset is reset to 0, as is the proc_trace_output_offset, and packet writes will resume at the base address indicated in the first entry.

If the table has no STOP or END entry, and trace-packet generation remains enabled, eventually the maximum table size will be reached (proc_trace_table_offset = 0FFFFFF8H). In this case, the proc_trace_table_offset and proc_trace_output_offset are reset to 0 (wrapping back to the beginning of the current table) once the last output region is filled.

It is important to note that processor updates to the IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs are asynchronous to instruction execution. Thus, reads of these MSRs while Intel PT is enabled may return stale values. Like all IA32_RTIT_* MSRs, the values of these MSRs should not be trusted or saved unless trace packet generation is first disabled by clearing IA32_RTIT_CTL.TraceEn. This ensures that the output MSR values account for all packets generated to that point, after which the processor will cease updating the output MSR values until tracing resumes. [1]

The processor may cache internally any number of entries from the current table or from tables that it references (directly or indirectly). If tracing is enabled, the processor may ignore or delay detection of modifications to these tables. To ensure that table changes are detected by the processor in a predictable manner, software should clear TraceEn before modifying the current table (or tables that it references) and only then re-enable packet generation.

Single Output Region ToPA Implementation

The first processor generation to implement Intel PT supports only ToPA configurations with a single ToPA entry followed by an END entry that points back to the first entry (creating one circular output buffer). Such processors enumerate CPUID.(EAX=14H,ECX=0):ECX.MENTRY[bit 1] = 0 and CPUID.(EAX=14H,ECX=0):ECX.TOPAOUT[bit 0] = 1.

If CPUID.(EAX=14H,ECX=0):ECX.MENTRY[bit 1] = 0, ToPA tables can hold only one output entry, which must be followed by an END=1 entry which points back to the base of the table. Hence only one contiguous block can be used as output.

The lone output entry can have INT or STOP set, but nonetheless must be followed by an END entry as described above. Note that, if INT=1, the PMI will actually be delivered before the region is filled.

ToPA Table Entry Format

The format of ToPA table entries is shown in Figure 33-2. The size of the address field is determined by the processor's physical-address width (MAXPHYADDR) in bits, as reported in CPUID.80000008H:EAX[7:0].

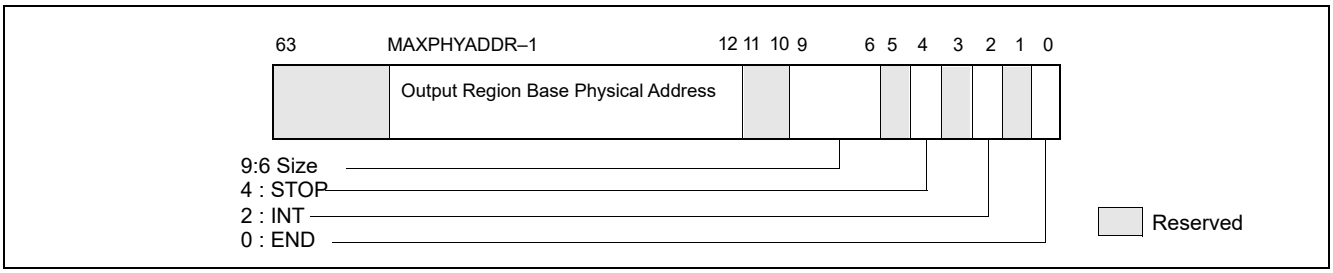


Figure 33-2. Layout of ToPA Table Entry

Table 33-3 describes the details of the ToPA table entry fields. If reserved bits are set to 1, an error is signaled.

Table 33-3. ToPA Table Entry Fields

ToPA Entry Field	Description
Output Region Base Physical Address	If END=0, this is the base physical address of the output region specified by this entry. Note that all regions must be aligned based on their size. Thus a 2M region must have bits 20:12 clear. If the region is not properly aligned, an operational error will be signaled when the entry is reached. If END=1, this is the 4K-aligned base physical address of the next ToPA table (which may be the base of the current table, or the first table in the linked list if a circular buffer is desired). If the processor supports only a single ToPA output region (see above), this address must be the value currently in the IA32_RTIT_OUTPUT_BASE MSR.

1. Although WRMSR is a serializing instruction, the execution of WRMSR that forces packet writes by clearing TraceEn does not itself cause these writes to be globally observed.

Table 33-3. ToPA Table Entry Fields (Contd.)

ToPA Entry Field	Description
Size	Indicates the size of the associated output region. Encodings are: 0: 4K, 1: 8K, 2: 16K, 3: 32K, 4: 64K, 5: 128K, 6: 256K, 7: 512K, 8: 1M, 9: 2M, 10: 4M, 11: 8M, 12: 16M, 13: 32M, 14: 64M, 15: 128M This field is ignored if END=1.
STOP	When the output region indicated by this entry is filled, software should disable packet generation. This will be accomplished by setting IA32_RTIT_STATUS.Stopped, which clears TriggerEn. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
INT	When the output region indicated by this entry is filled, signal Perfmon LVT interrupt. Note that if both INT and STOP are set in the same entry, the STOP will happen before the INT. Thus the interrupt handler should expect that the IA32_RTIT_STATUS.Stopped bit will be set, and will need to be reset before tracing can be resumed. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
END	If set, indicates that this is an END entry, and thus the address field points to a table base rather than an output region base. If END=1, INT and STOP must be set to 0; otherwise it is treated as reserved bit violation (see ToPA Errors). The Size field is ignored in this case. If the processor supports only a single ToPA output region (see above), END must be set in the second table entry.

ToPA STOP

Each ToPA entry has a STOP bit. If this bit is set, the processor will set the IA32_RTIT_STATUS.Stopped bit when the corresponding trace output region is filled. This will clear TriggerEn and thereby cease packet generation. See Section 33.2.8.4 for details on IA32_RTIT_STATUS.Stopped. This sequence is known as "ToPA Stop".

No TIP.PGD packet will be seen in the output when the ToPA stop occurs, since the disable happens only when the region is already full. When this occurs, output ceases after the last byte of the region is filled, which may mean that a packet is cut off in the middle. Any packets remaining in internal buffers are lost and cannot be recovered.

When ToPA stop occurs, the IA32_RTIT_OUTPUT_BASE MSR will hold the base address of the table whose entry had STOP=1. IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTableOffset will hold the index value for that entry, and the IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset should be set to the size of the region minus one.

Note that this means the offset pointer is pointing to the next byte after the end of the region, a configuration that would produce an operational error if the configuration remained when tracing is re-enabled with IA32_RTIT_STATUS.Stopped cleared.

ToPA PMI

Each ToPA entry has an INT bit. If this bit is set, the processor will signal a performance-monitoring interrupt (PMI) when the corresponding trace output region is filled. This interrupt is not precise, and it is thus likely that writes to the next region will occur by the time the interrupt is taken.

The following steps should be taken to configure this interrupt:

1. Enable PMI via the LVT Performance Monitor register (at MMIO offset 340H in xAPIC mode; via MSR 834H in x2APIC mode). See the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for more details on this register. For ToPA PMI, set all fields to 0, save for the interrupt vector, which can be selected by software.
2. Set up an interrupt handler to service the interrupt vector that a ToPA PMI can raise.
3. Set the interrupt flag by executing STI.
4. Set the INT bit in the ToPA entry of interest and enable packet generation, using the ToPA output option. Thus, TraceEn=ToPA=1 in the IA32_RTIT_CTL MSR.

Once the INT region has been filled with packet output data, the interrupt will be signaled. This PMI can be distinguished from others by checking bit 55 (Trace_ToPA_PMI) of the IA32_PERF_GLOBAL_STATUS MSR (MSR 38EH). Once the ToPA PMI handler has serviced the relevant buffer, writing 1 to bit 55 of the MSR at 390H (IA32_GLOBAL_STATUS_RESET) clears IA32_PERF_GLOBAL_STATUS.Trace_ToPA_PMI.

Intel PT is not frozen on PMI, and thus the interrupt handler will be traced (though filtering can prevent this). The Freeze_Perfmon_on_PMI and Freeze_LBRs_on_PMI settings in IA32_DEBUGCTL will be applied on ToPA PMI just as on other PMIs, and hence Perfmon counters are frozen.

Assuming the PMI handler wishes to read any buffered packets for persistent output, or wishes to modify any Intel PT MSRs, software should first disable packet generation by clearing TraceEn. This ensures that all buffered packets are written to memory and avoids tracing of the PMI handler. The configuration MSRs can then be used to determine where tracing has stopped. If packet generation is disabled by the handler, it should then be manually re-enabled before the IRET if continued tracing is desired.

In rare cases, it may be possible to trigger a second ToPA PMI before the first is handled. This can happen if another ToPA region with INT=1 is filled before, or shortly after, the first PMI is taken, perhaps due to EFLAGS.IF being cleared for an extended period of time. This can manifest in two ways: either the second PMI is triggered before the first is taken, and hence only one PMI is taken, or the second is triggered after the first is taken, and thus will be taken when the handler for the first completes. Software can minimize the likelihood of the second case by clearing TraceEn at the beginning of the PMI handler. Further, it can detect such cases by then checking the Interrupt Request Register (IRR) for PMI pending, and checking the ToPA table base and off-set pointers (in IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS) to see if multiple entries with INT=1 have been filled.

PMI Preservation

In some cases a ToPA PMI may be taken after completion of an XSAVES instruction that saves Intel PT state, and in such cases any modification of Intel PT MSRs within the PMI handler will not persist when the saved Intel PT context is later restored with XRSTORS. To account for such a scenario, the PMI Preservation feature has been added. Support for this feature is indicated by CPUID.(EAX=14H, ECX=0):EBX[bit 6].

When IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, PMI preservation is enabled. When a ToPA region with INT=1 is filled, a PMI is pended and the new IA32_RTIT_STATUS.PendToPAPMI[7] is set to 1. If this bit is set when Intel PT is enabled, such that IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a ToPA PMI is pended. This behavior ensures that any ToPA PMI that is pended during XSAVES, and hence can't be properly handled, will be re-pended when the saved PT state is restored.

When this feature is enabled, the PMI handler should take the following actions:

1. Ignore ToPA PMIs that are taken when TraceEn = 0. This indicates that the PMI was pended during Intel PT disable, and the PendToPAPMI flag will ensure that the PMI is re-pended once Intel PT is re-enabled in the same context. For this reason, the PendToPAPMI bit should be left set to 1.
2. If TraceEn=1 and the PMI can be properly handled, clear the new PendTopaPMI bit. This will ensure that additional, spurious ToPA PMIs are not taken. It is required that PendToPAPMI is cleared before the PMI LVT mask is cleared in the APIC, and before any clearing of either LBRS_FROZEN or COUNTERS_FROZEN in IA32_PERF_GLOBAL_STATUS.

ToPA PMI and Single Output Region ToPA Implementation

A processor that supports only a single ToPA output region implementation (such that only one output region is supported; see above) will attempt to signal a ToPA PMI interrupt before the output wraps and overwrites the top of the buffer. To support this functionality, the PMI handler should disable packet generation as soon as possible.

Due to PMI skid, it is possible that, in rare cases, the wrap will have occurred before the PMI is delivered. Software can avoid this by setting the STOP bit in the ToPA entry (see Table 33-3); this will disable tracing once the region is filled, and no wrap will occur. This approach has the downside of disabling packet generation so that some of the instructions that led up to the PMI will not be traced. If the PMI skid is significant enough to cause the region to fill and tracing to be disabled, the PMI handler will need to clear the IA32_RTIT_STATUS.Stopped indication before tracing can resume.

ToPA PMI and XSAVES/XRSTORS State Handling

In some cases the ToPA PMI may be taken after completion of an XSAVES instruction that switches Intel PT state, and in such cases any modification of Intel PT MSRs within the PMI handler will not persist when the saved Intel PT context is later restored with XRSTORS. To account for such a scenario, it is recommended that the Intel PT output configuration be modified by altering the ToPA tables themselves, rather than the Intel PT output MSRs. On processors that support PMI preservation (CPUID.(EAX=14H, ECX=0):EBX[bit 6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PMI that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendTopaPMI[7] = 1. A PMI will then be pended when the saved PT context is later restored.

Table 33-4 depicts a recommended PMI handler algorithm for managing multi-region ToPA output and handling ToPA PMIs that may arrive between XSAVES and XRSTORS, if PMI preservation is not in use. This algorithm is flexible to allow software to choose between adding entries to the current ToPA table, adding a new ToPA table, or using the current ToPA table as a circular buffer. It assumes that the ToPA entry that triggers the PMI is not the last entry in the table, which is the recommended treatment.

Table 33-4. Algorithm to Manage Intel PT ToPA PMI and XSAVES/XRSTORS

Pseudo Code Flow

```
IF (IA32_PERF_GLOBAL_STATUS.ToPA)
    Save IA32_RTIT_CTL value;
    IF ( IA32_RTIT_CTL.TraceEN )
        Disable Intel PT by clearing TraceEn;
    FI;
    IF ( there is space available to grow the current ToPA table )
        Add one or more ToPA entries after the last entry in the ToPA table;
        Point new ToPA entry address field(s) to new output region base(s);
    ELSE
        Modify an upcoming ToPA entry in the current table to have END=1;
        IF (output should transition to a new ToPA table )
            Point the address of the "END=1" entry of the current table to the new table base;
        ELSE
            /* Continue to use the current ToPA table, make a circular. */
            Point the address of the "END=1"l entry to the base of the current table;
            Modify the ToPA entry address fields for filled output regions to point to new, unused output regions;
            /* Filled regions are those with index in the range of 0 to (IA32_RTIT_MASK_PTRS.MaskOrTableOffset -1). */
        FI;
      FI;
      Restore saved IA32_RTIT_CTL.value;
FI;
```

ToPA Errors

When a malformed ToPA entry is found, an **operational error** results (see Section 33.3.10). A malformed entry can be any of the following:

1. **ToPA entry reserved bit violation**.
   This describes cases where a bit marked as reserved in Section 33.2.7.2 above is set to 1.
2. **ToPA alignment violation**.
   This includes cases where illegal ToPA entry base address bits are set to 1:
   a. ToPA table base address is not 4KB-aligned. The table base can be from a WRMSR to IA32_RTIT_OUTPUT_BASE, or from a ToPA entry with END=1.
   b. ToPA entry base address is not aligned to the ToPA entry size (e.g., a 2MB region with base address[20:12] not equal to 0), for ToPA entries with END=0.
   c. ToPA entry base address sets upper physical address bits not supported by the processor.

3. **Illegal ToPA Output Offset**.
   IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset is greater than or equal to the size of the current ToPA output region size.
4. **ToPA rules violations**.
   These are similar to ToPA entry reserved bit violations; they are cases when a ToPA entry is encountered with illegal field combinations. They include the following:
   a. Setting the STOP or INT bit on an entry with END=1.
   b. Setting the END bit in entry 0 of a ToPA table.
   c. On processors that support only a single ToPA entry (see above), two additional illegal settings apply:
      i) ToPA table entry 1 with END=0.
      ii) ToPA table entry 1 with base address not matching the table base.

In all cases, the error will be logged by setting IA32_RTIT_STATUS.Error, thereby disabling tracing when the problematic ToPA entry is reached (when proc_trace_table_offset points to the entry containing the error). Any packet bytes that are internally buffered when the error is detected may be lost.

Note that operational errors may also be signaled due to attempts to access restricted memory. See Section 33.2.7.4 for details.

A tracing software have a range of flexibility using ToPA to manage the interaction of Intel PT with application buffers, see Section 33.4.2.26.

33.2.7.3 Trace Transport Subsystem

When IA32_RTIT_CTL.FabricEn is set, the IA32_RTIT_CTL.ToPA bit is ignored, and trace output is written to the trace transport subsystem. The endpoints of this transport are platform-specific, and details of configuration options should refer to the specific platform documentation. The FabricEn bit is available to be set if CPUID(EAX=14H,ECX=0):EBX[bit 3] = 1.

33.2.7.4 Restricted Memory Access

Packet output cannot be directed to any regions of memory that are restricted by the platform. In particular, all memory accesses on behalf of packet output are checked against the SMRR regions. If there is any overlap with these regions, trace data collection will not function properly. Exact processor behavior is implementation-dependent; Table 33-5 summarizes several scenarios.

Table 33-5. Behavior on Restricted Memory Access

Scenario	Description
ToPA output region overlaps with SMRR	Stores to the restricted memory region will be dropped, and that packet data will be lost. Any attempt to read from that restricted region will return all 1s. The processor also may signal an error (Section 33.3.10) and disable tracing when the output pointer reaches the restricted region. If packet generation remains enabled, then packet output may continue once stores are no longer directed to restricted memory (on wrap, or if the output region is larger than the restricted memory region).
ToPA table overlaps with SMRR	The processor will signal an error (Section 33.3.10) and disable tracing when the ToPA write pointer (IA32_RTIT_OUTPUT_BASE + proc_trace_table_offset) enters the restricted region.

It should also be noted that packet output should not be routed to the 4KB APIC MMIO region, as defined by the IA32_APIC_BASE MSR. For details about the APIC, refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A. No error is signaled for this case.

Modifications to Restricted Memory Regions

It is recommended that software disable packet generation before modifying the SMRRs to change the scope of the SMRR regions. This is because the processor reserves the right to cache any number of ToPA table entries internally, after checking them against restricted memory ranges. Once cached, the entries will not be checked again, meaning one could potentially route packet output to a newly restricted region. Software can ensure that any cached entries are written to memory by clearing IA32_RTIT_CTL.TraceEn.

33.2.8 Enabling and Configuration MSRs

33.2.8.1 General Considerations

Trace packet generation is enabled and configured by a collection of model-specific registers (MSRs), which are detailed below. Some notes on the configuration MSR behavior:

- If Intel Processor Trace is not supported by the processor (see Section 33.3.1), RDMSR or WRMSR of the IA32_RTIT_* MSRs will cause #GP.
- A WRMSR to any of the IA32_RTIT_* configuration MSRs while packet generation is enabled (IA32_RTIT_CTL.TraceEn=1) will generate a #GP exception. Packet generation must be disabled before the configuration MSRs can be changed.

  Note: Software may write the same value back to IA32_RTIT_CTL without #GP, even if TraceEn=1.
- All configuration MSRs for Intel PT are duplicated per logical processor
- For each configuration MSR, any MSR write that attempts to change bits marked reserved, or utilize encodings marked reserved, will cause a #GP fault.
- All configuration MSRs for Intel PT are cleared on a warm or cold RESET.
  - If CPUID.(EAX=14H, ECX=0):EBX[bit 2] = 1, only the TraceEn bit is cleared on warm RESET; though this may have the impact of clearing other bits in IA32_RTIT_STATUS. Other MSR values of the trace configuration MSRs are preserved on warm RESET.
- The semantics of MSR writes to trace configuration MSRs in this chapter generally apply to explicit WRMSR to these registers, using VMexit or VM entry MSR load list to these MSRs, XRSTORS with requested feature bit map including XSAVE map component of state_8 (corresponding to IA32_XSS[bit 8]), and the write to IA32_RTIT_CTL.TraceEn by XSAVES (Section 33.3.5.2).

33.2.8.2 IA32_RTIT_CTL MSR

IA32_RTIT_CTL, at address 570H, is the primary enable and control MSR for trace packet generation. Bit positions are listed in Table 33-6.

Table 33-6. IA32_RTIT_CTL MSR

Position	Bit Name	At Reset	Bit Description
0	TraceEn	0	If 1, enables tracing; else tracing is disabled. When this bit transitions from 1 to 0, all buffered packets are flushed out of internal buffers. A further store, fence, or architecturally serializing instruction may be required to ensure that packet data can be observed at the trace endpoint. See Section 33.2.8.3 for details of enabling and disabling packet generation. Note that the processor will clear this bit on #SMI (Section 33.2.9.3) and warm reset. Other MSR bits of IA32_RTIT_CTL (and other trace configuration MSRs) are not impacted by these events.
1	CYCEn	0	0: Disables CYC Packet (see Section 33.4.2.14). 1: Enables CYC Packet. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 1] = 0.
2	OS	0	0: Packet generation is disabled when CPL = 0. 1: Packet generation may be enabled when CPL = 0.
3	User	0	0: Packet generation is disabled when CPL > 0. 1: Packet generation may be enabled when CPL > 0.
4	PwrEvtEn	0	0: Power Event Trace packets are disabled. 1: Power Event Trace packets are enabled (see Section 33.2.3, “Power Event Tracing”).

Table 33-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
5	FUPonPTW	0	0: PTW packets are not followed by FUPs. 1: PTW packets are followed by FUPs. This bit is reserved when CPUID.(EAX=14H, ECX=0):EBX[bit 4] ("PTWRITE Supported") is 0.
6	FabricEn	0	0: Trace output is directed to the memory subsystem, mechanism depends on IA32_RTIT_CTL.ToPA. 1: Trace output is directed to the trace transport subsystem, IA32_RTIT_CTL.ToPA is ignored. This bit is reserved if CPUID.(EAX=14H, ECX=0):ECX[bit 3] = 0.
7	CR3Filter	0	0: Disables CR3 filtering. 1: Enables CR3 filtering. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 0] ("CR3 Filtering Support") is 0.
8	ToPA	0	0: Single-range output scheme enabled if CPUID.(EAX=14H, ECX=0):ECX.SNGLRGNOUT[bit 2] = 1 and IA32_RTIT_CTL.FabricEn=0. 1: ToPA output scheme enabled (see Section 33.2.7.2) if CPUID.(EAX=14H, ECX=0):ECX.TOPA[bit 0] = 1, and IA32_RTIT_CTL.FabricEn=0. Note: WRMSR to IA32_RTIT_CTL that sets TraceEn but clears this bit and FabricEn would cause #GP, if CPUID.(EAX=14H, ECX=0):ECX.SNGLRGNOUT[bit 2] = 0. WRMSR to IA32_RTIT_CTL that sets this bit causes #GP, if CPUID.(EAX=14H, ECX=0):ECX.TOPA[bit 0] = 0.
9	MTCEn	0	0: Disables MTC Packet (see Section 33.4.2.16). 1: Enables MTC Packet. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 3] = 0.
10	TSCEn	0	0: Disable TSC packets. 1: Enable TSC packets (see Section 33.4.2.11).
11	DisRETC	0	0: Enable RET compression. 1: Disable RET compression (see Section 33.2.1.2).
12	PTWEn	0	0: PTWRITE packet generation disabled. 1: PTWRITE packet generation enabled (see Table 33-40 "PTW Packet Definition"). This bit is reserved when CPUID.(EAX=14H, ECX=0):EBX[bit 4] ("PTWRITE Supported") is 0.
13	BranchEn	0	0: Disable COFI-based packets. 1: Enable COFI-based packets: FUP, TIP, TIP.PGE, TIP.PGD, TNT, MODE.Exec, MODE.TSX. See Section 33.2.6.4 for details on BranchEn.
17:14	MTCFreq	0	Defines MTC packet Frequency, which is based on the core crystal clock, or Always Running Timer (ART). MTC will be sent each time the selected ART bit toggles. The following Encodings are defined: 0: ART(0), 1: ART(1), 2: ART(2), 3: ART(3), 4: ART(4), 5: ART(5), 6: ART(6), 7: ART(7), 8: ART(8), 9: ART(9), 10: ART(10), 11: ART(11), 12: ART(12), 13: ART(13), 14: ART(14), 15: ART(15) Software must use CPUID to query the supported encodings in the processor, see Section 33.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 3] = 0.
18	Reserved	0	Must be 0.

Table 33-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
22:19	CycThresh	0	CYC packet threshold, see Section 33.3.6 for details. CYC packets will be sent with the first eligible packet after N cycles have passed since the last CYC packet. If CycThresh is 0 then N=0, otherwise N is defined as $2^{(CycThresh-1)}$. The following Encodings are defined: 0: 0, 1: 1, 2: 2, 3: 4, 4: 8, 5: 16, 6: 32, 7: 64, 8: 128, 9: 256, 10: 512, 11: 1024, 12: 2048, 13: 4096, 14: 8192, 15: 16384 Software must use CPUID to query the supported encodings in the processor, see Section 33.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 1] = 0.
23	Reserved	0	Must be 0.
27:24	PSBFreq	0	Indicates the frequency of PSB packets. PSB packet frequency is based on the number of Intel PT packet bytes output, so this field allows the user to determine the increment of IA32_IA32_RTIT_STATUS.PacketByteCnt that should cause a PSB to be generated. Note that PSB insertion is not precise, but the average output bytes per PSB should approximate the SW selected period. The following Encodings are defined: 0: 2K, 1: 4K, 2: 8K, 3: 16K, 4: 32K, 5: 64K, 6: 128K, 7: 256K, 8: 512K, 9: 1M, 10: 2M, 11: 4M, 12: 8M, 13: 16M, 14: 32M, 15: 64M Software must use CPUID to query the supported encodings in the processor, see Section 33.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 1] = 0.
30:28	Reserved	0	Must be 0.
31	EventEn	0	0: Event Trace packets are disabled. 1: Event Trace packets are enabled. This bit is reserved when CPUID.(EAX=14H, ECX=0):EBX[bit 7] ("Event Trace Supported") is 0.
35:32	ADDR0_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR0_A/B based on the following encodings: 0: ADDR0 range unused. 1: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 33.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See 4.2.8 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 1.
39:36	ADDR1_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR1_A/B based on the following encodings: 0: ADDR1 range unused. 1: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 33.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 33.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 2.

Table 33-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
43:40	ADDR2_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR2_A/B based on the following encodings: 0: ADDR2 range unused. 1: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 33.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 33.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 3.
47:44	ADDR3_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR3_A/B based on the following encodings: 0: ADDR3 range unused. 1: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 33.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 33.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 4.
54:48	Reserved	0	Reserved only for future trace content enables, or address filtering configuration enables. Must be 0.
55	DisTNT	0	0: Include TNT packets in control flow trace. 1: Omit TNT packets from control flow trace. This bit is reserved when CPUID.(EAX=14H, ECX=0):EBX[bit 8] ("TNT Disable Supported") is 0. SeeSection 33.3.9 for details.
56	InjectPsbPmiOnEnable	0	1: Enables use of IA32_RTIT_STATUS bits PendPSB[6] and PendTopaPMI[7], see Section 33.2.8.4, "IA32_RTIT_STATUS MSR," for behavior of these bits. 0: IA32_RTIT_STATUS bits 6 and 7 are ignored. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 6] = 0.
59:57	Reserved	0	Reserved only for future trace content enables, or address filtering configuration enables. Must be 0.
63:60	Reserved	0	Must be 0.

33.2.8.3 Enabling and Disabling Packet Generation with TraceEn

When TraceEn transitions from 0 to 1, Intel Processor Trace is enabled, and a series of packets may be generated. These packets help ensure that the decoder is aware of the state of the processor when the trace begins, and that it can keep track of any timing or state changes that may have occurred while packet generation was disabled. A full PSB+ (see Section 33.4.2.17) will be generated if IA32_RTIT_STATUS.PacketByteCnt=0, and may be generated in other cases as well. Otherwise, timing packets will be generated, including TSC, TMA, and CBR (see Section 33.4.1.1).

In addition to the packets discussed above, if and when PacketEn (Section 33.2.6.1) transitions from 0 to 1 (which may happen immediately, depending on filtering settings), a TIP.PGE packet (Section 33.4.2.3) will be generated.

When TraceEn is set, the processor may read ToPA entries from memory and cache them internally. For this reason, software should disable packet generation before making modifications to the ToPA tables (or changing the config-

uration of restricted memory regions). See Section 33.7 for more details of packets that may be generated with modifications to TraceEn.

Disabling Packet Generation

Clearing TraceEn causes any packet data buffered within the logical processor to be flushed out, after which the output MSRs (IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS) will have stable values. When output is directed to memory, a store, fence, or architecturally serializing instruction may be required to ensure that the packet data is globally observed. No special packets are generated by disabling packet generation, though a TIP.PGD may result if PacketEn=1 at the time of disable.

Other Writes to IA32_RTIT_CTL

Any attempt to modify IA32_RTIT_CTL while TraceEn is set will result in a general-protection fault (#GP) unless the same write also clears TraceEn. However, writes to IA32_RTIT_CTL that do not modify any bits will not cause a #GP, even if TraceEn remains set.

33.2.8.4 IA32_RTIT_STATUS MSR

The IA32_RTIT_STATUS MSR is readable and writable by software, though some fields cannot be modified by software. See Table 33-7 for details. The WRMSR instruction ignores these bits in the source operand (attempts to modify these bits are ignored and do not cause WRMSR to fault).

This MSR can only be written when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP). The processor does not modify the value of this MSR while TraceEn is 0 (software can modify it with WRMSR).

Table 33-7. IA32_RTIT_STATUS MSR

Position	Bit Name	At Reset	Bit Description
0	FilterEn	0	This bit is written by the processor, and indicates that tracing is allowed for the current IP, see Section 33.2.6.5. Writes are ignored.
1	ContextEn	0	The processor sets this bit to indicate that tracing is allowed for the current context. See Section 33.2.6.3. Writes are ignored.
2	TriggerEn	0	The processor sets this bit to indicate that tracing is enabled. See Section 33.2.6.2. Writes are ignored.
3	Reserved	0	Must be 0.
4	Error	0	The processor sets this bit to indicate that an operational error has been encountered. When this bit is set, TriggerEn is cleared to 0 and packet generation is disabled. For details, see "ToPA Errors" in Section 33.2.7.2. When TraceEn is cleared, software can write this bit. Once it is set, only software can clear it. It is not recommended that software ever set this bit, except in cases where it is restoring a prior saved state.
5	Stopped	0	The processor sets this bit to indicate that a ToPA Stop condition has been encountered. When this bit is set, TriggerEn is cleared to 0 and packet generation is disabled. For details, see "ToPA STOP" in Section 33.2.7.2. When TraceEn is cleared, software can write this bit. Once it is set, only software can clear it. It is not recommended that software ever set this bit, except in cases where it is restoring a prior saved state.
6	PendPSB	0	If IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, the processor sets this bit when the threshold for a PSB+ to be inserted has been reached. The processor will clear this bit when the PSB+ has been inserted into the trace. If PendPSB = 1 and InjectPsbPmiOnEnable = 1 when IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a PSB+ will be inserted into the trace. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 6] = 0.

Table 33-7. IA32_RTIT_STATUS MSR

Position	Bit Name	At Reset	Bit Description
7	PendTopaPMI	0	If IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, the processor sets this bit when the threshold for a ToPA PMI to be inserted has been reached. Software should clear this bit once the ToPA PMI has been handled, see "ToPA PMI" for details. If PendTopaPMI = 1 and InjectPsbPmiOnEnable = 1 when IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a PMI will be pended. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 6] = 0.
31:8	Reserved	0	Must be 0.
48:32	PacketByteCnt	0	This field is written by the processor, and holds a count of packet bytes that have been sent out. The processor also uses this field to determine when the next PSB packet should be inserted. Note that the processor may clear or modify this field at any time while IA32_RTIT_CTL.TraceEn=1. It will have a stable value when IA32_RTIT_CTL.TraceEn=0. See Section 33.4.2.17 for details. This field is reserved when CPUID.(EAX=14H,ECX=0):EBX[bit 1] ("Configurable PSB and CycleAccurate Mode Supported") is 0.
63:49	Reserved	0	Must be 0.

33.2.8.5 IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B MSRs

The role of the IA32_RTIT_ADDRn_A/B register pairs, for each n, is determined by the corresponding ADDRn_CFG fields in IA32_RTIT_CTL (see Section 33.2.8.2). The number of these register pairs is enumerated by CPUID.(EAX=14H, ECX=1):EAX.RANGECNT[2:0].

- Processors that enumerate support for 1 range support:
  - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
- Processors that enumerate support for 2 ranges support:
  - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
  - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
- Processors that enumerate support for 3 ranges support:
  - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
  - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
  - IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B
- Processors that enumerate support for 4 ranges support:
  - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
  - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
  - IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B
  - IA32_RTIT_ADDR3_A, IA32_RTIT_ADDR3_B

Each register has a single 64-bit field that holds a linear address value. Writes must ensure that the address is in canonical form, otherwise a general-protection fault (#GP) fault will result.

Each MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

33.2.8.6 IA32_RTIT_CR3_MATCH MSR

The IA32_RTIT_CR3_MATCH register is compared against CR3 when IA32_RTIT_CTL.CR3Filter is 1. Bits 63:5 hold the CR3 address value to match, bits 4:0 are reserved to 0. For more details on CR3 filtering and the treatment of this register, see Section 33.2.5.2.

This MSR is accessible if CPUID.(EAX=14H, ECX=0):EBX[bit 0], "CR3 Filtering Support", is 1. This MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP). IA32_RTIT_CR3_MATCH[4:0] are reserved and must be 0; an attempt to set those bits using WRMSR causes a #GP.

33.2.8.7 IA32_RTIT_OUTPUT_BASE MSR

This MSR is used to configure the trace output destination, when output is directed to memory (IA32_RTIT_CTL.FabricEn = 0). The size of the address field is determined by the maximum physical address width (MAXPHYADDR), as reported by CPUID.80000008H:EAX[7:0].

When the ToPA output scheme is used, the processor may update this MSR when packet generation is enabled, and those updates are asynchronous to instruction execution. Therefore, the values in this MSR should be considered unreliable unless packet generation is disabled (IA32_RTIT_CTL.TraceEn = 0).

Accesses to this MSR are supported only if Intel PT output to memory is supported, hence when either CPUID.(EAX=14H, ECX=0):ECX[bit 0] or CPUID.(EAX=14H, ECX=0):ECX[bit 2] are set. Otherwise WRMSR or RDMSR cause a general-protection fault (#GP). If supported, this MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

Table 33-8. IA32_RTIT_OUTPUT_BASE MSR

Position	Bit Name	At Reset	Bit Description
6:0	Reserved	0	Must be 0.
MAXPHYADDR-1:7	BasePhysAddr	0	The base physical address. How this address is used depends on the value of IA32_RTIT_CTL.ToPA: 0: This is the base physical address of a single, contiguous physical output region. This could be mapped to DRAM or to MMIO, depending on the value. The base address should be aligned with the size of the region, such that none of the 1s in the mask value(Section 33.2.8.8) overlap with 1s in the base address. If the base is not aligned, an operational error will result (see Section 33.3.10). 1: The base physical address of the current ToPA table. The address must be 4K aligned. Writing an address in which bits 11:7 are non-zero will not cause a #GP, but an operational error will be signaled once TraceEn is set. See "ToPA Errors" in Section 33.2.7.2, as well as Section 33.3.10.
63:MAXPHYADDR	Reserved	0	Must be 0.

33.2.8.8 IA32_RTIT_OUTPUT_MASK_PTRS MSR

This MSR holds any mask or pointer values needed to indicate where the next byte of trace output should be written. The meaning of the values held in this MSR depend on whether the ToPA output mechanism is in use. See Section 33.2.7.2 for details.

The processor updates this MSR while when packet generation is enabled, and those updates are asynchronous to instruction execution. Therefore, the values in this MSR should be considered unreliable unless packet generation is disabled (IA32_RTIT_CTL.TraceEn = 0).

Accesses to this MSR are supported only if Intel PT output to memory is supported, hence when either CPUID.(EAX=14H, ECX=0):ECX[bit 0] or CPUID.(EAX=14H, ECX=0):ECX[bit 2] are set. Otherwise WRMSR or RDMSR cause a general-protection fault (#GP). If supported, this MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

Table 33-9. IA32_RTIT_OUTPUT_MASK_PTRS MSR

Position	Bit Name	At Reset	Bit Description
6:0	LowerMask	7FH	Forced to 1, writes are ignored.
31:7	MaskOrTableOffset	0	The use of this field depends on the value of IA32_RTIT_CTL.ToPA: 0: This field holds bits 31:7 of the mask value for the single, contiguous physical output region. The size of this field indicates that regions can be of size 128B up to 4GB. This value (combined with the lower 7 bits, which are reserved to 1) will be ANDed with the OutputOffset field to determine the next write address. All 1s in this field should be consecutive and starting at bit 7, otherwise the region will not be contiguous, and an operational error (Section 33.3.10) will be signaled when TraceEn is set. 1: This field holds bits 27:3 of the offset pointer into the current ToPA table. This value can be added to the IA32_RTIT_OUTPUT_BASE value to produce a pointer to the current ToPA table entry, which itself is a pointer to the current output region. In this scenario, the lower 7 reserved bits are ignored. This field supports tables up to 256 MBytes in size.
63:32	OutputOffset	0	The use of this field depends on the value of IA32_RTIT_CTL.ToPA: 0: This is bits 31:0 of the offset pointer into the single, contiguous physical output region. This value will be added to the IA32_RTIT_OUTPUT_BASE value to form the physical address at which the next byte of packet output data will be written. This value must be less than or equal to the MaskOrTableOffset field, otherwise an operational error (Section 33.3.10) will be signaled when TraceEn is set. 1: This field holds bits 31:0 of the offset pointer into the current ToPA output region. This value will be added to the output region base field, found in the current ToPA table entry, to form the physical address at which the next byte of trace output data will be written. This value must be less than the ToPA entry size, otherwise an operational error (Section 33.3.10) will be signaled when TraceEn is set.

33.2.9 Interaction of Intel® Processor Trace and Other Processor Features

33.2.9.1 Intel® Transactional Synchronization Extensions (Intel® TSX)

The operation of Intel TSX is described in Chapter 14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1. For tracing purpose, packet generation does not distinguish between hardware lock elision (HLE) and restricted transactional memory (RTM), but speculative execution does have impacts on the trace output. Specifically, packets are generated as instructions complete, even for instructions in a transactional region that is later aborted. For this reason, debugging software will need indication of the beginning and end of a transactional region; this will allow software to understand when instructions are part of a transactional region and whether that region has been committed.

To enable this, TSX information is included in a MODE packet leaf. The mode bits in the leaf are:

- **InTX**: Set to 1 on an TSX transaction begin, and cleared on transaction commit or abort.
- **TXAbort**: Set to 1 only when InTX transitions from 1 to 0 on an abort. Cleared otherwise.

If BranchEn=1, this MODE packet will be sent each time the transaction status changes. See Table 33-10 for details.

Table 33-10. TSX Packet Scenarios with BranchEn=1

TSX Event	Instruction	Packets
Transaction Begin	Either XBEGIN or XACQUIRE lock (the latter if executed transactionally)	MODE(TXAbort=0, InTX=1), FUP(CurrentIP)
Transaction Commit	Either XEND or XRELEASE lock, if transactional execution ends. This happens only on the outermost commit	MODE(TXAbort=0, InTX=0), FUP(CurrentIP)

Table 33-10. TSX Packet Scenarios with BranchEn=1

TSX Event	Instruction	Packets
Transaction Abort	XABORT or other transactional abort	MODE(TXAbort=1, InTX=0), FUP(CurrentIP), TIP(TargetIP)
Other	One of the following: ▪ Nested XBEGIN or XACQUIRE lock ▪ An outer XACQUIRE lock that doesn't begin a transaction (InTX not set) ▪ Non-outermost XEND or XRELEASE lock	None. No change to TSX mode bits for these cases.

The CurrentIP listed above is the IP of the associated instruction. The TargetIP is the IP of the next instruction to be executed; for HLE, this is the XACQUIRE lock; for RTM, this is the fallback handler.

Intel PT stores are non-transactional, and thus packet writes are not rolled back on TSX abort.

33.2.9.2 TSX and IP Filtering

A complication with tracking transactions is handling transactions that start or end outside of the tracing region. Transactions can't span across a change in ContextEn, because CPL changes and CR3 changes each cause aborts. But a transaction can start within the IP filter region and end outside it.

To assist the decoder handling this situation, MODE.TSX packets can be sent even if FilterEn=0, though there will be no FUP attached. Instead, they will merely serve to indicate to the decoder when transactions are active and when they are not. When tracing resumes (due to PacketEn=1), the last MODE.TSX preceding the TIP.PGE will indicate the current transaction status.

33.2.9.3 System Management Mode (SMM)

SMM code has special privileges that non-SMM code does not have. Intel Processor Trace can be used to trace SMM code, but special care is taken to ensure that SMM handler context is not exposed in any non-SMM trace collection. Additionally, packet output from tracing non-SMM code cannot be written into memory space that is either protected by SMRR or used by the SMM handler.

SMM is entered via a system management interrupt (SMI). SMI delivery saves the value of IA32_RTIT_CTL.TraceEn into SMRAM and then clears it, thereby disabling packet generation.

The saving and clearing of IA32_RTIT_CTL.TraceEn ensures two things:

1. All internally buffered packet data is flushed before entering SMM (see Section 33.2.8.2).
2. Packet generation ceases before entering SMM, so any tracing that was configured outside SMM does not continue into SMM. No SMM instruction pointers or other state will be exposed in the non-SMM trace.

When the RSM instruction is executed to return from SMM, the TraceEn value that was saved by SMI delivery is restored, allowing tracing to be resumed. As is done any time packet generation is enabled, ContextEn is re-evaluated, based on the values of CPL, CR3, etc., established by RSM.

Like other interrupts, delivery of an SMI produces a FUP containing the IP of the next instruction to execute. By toggling TraceEn, SMI and RSM can produce TIP.PGD and TIP.PGE packets, respectively, indicating that tracing was disabled or re-enabled. See Table 33.7 for more information about packets entering and leaving SMM.

Although #SMI and RSM change CR3, PIP packets are not generated in these cases. With #SMI tracing is disabled before the CR3 change; with RSM TraceEn is restored after CR3 is written.

TraceEn must be cleared before executing RSM, otherwise it will cause a shutdown. Further, on processors that restrict use of Intel PT with LBRs (see Section 33.3.1.2), any RSM that results in enabling of both will cause a shutdown.

Intel PT can support tracing of System Transfer Monitor operating in SMM, see Section 33.6.

33.2.9.4 Virtual-Machine Extensions (VMX)

Initial implementations of Intel Processor Trace do not support tracing in VMX operation. Such processors indicate this by returning 0 for IA32_VMX_MISC[bit 14]. On these processors, execution of the VMXON instruction clears IA32_RTIT_CTL.TraceEn and any attempt to write IA32_RTIT_CTL in VMX operation causes a general-protection exception (#GP).

Processors that support Intel Processor Trace in VMX operation return 1 for IA32_VMX_MISC[bit 14]. Details of tracing in VMX operation are described in Section 33.4.2.26.

33.2.9.5 Intel® Software Guard Extensions (Intel® SGX)

Intel SGX provides an application with the ability to instantiate a protective container (an enclave) with confidentiality and integrity (see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D). On a processor with both Intel PT and Intel SGX enabled, when executing code within a production enclave, no control flow packets are produced by Intel PT. An enclave entry will clear ContextEn, thereby blocking control flow packet generation. A TIP.PGD packet will be generated if PacketEn=1 at the time of the entry.

Upon enclave exit, ContextEn will no longer be forced to 0. If other enables are set at the time, a TIP.PGE may be generated to indicate that tracing is resumed.

During the enclave execution, Intel PT remains enabled, and periodic or timing packets such as PSB, TSC, MTC, or CBR can still be generated. No IPs or other architectural state will be exposed.

For packet generation examples on enclave entry or exit, see Section 33.7.

Debug Enclaves

Intel SGX allows an enclave to be configured with relaxed protection of confidentiality for debug purposes, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D. In a debug enclave, Intel PT continues to function normally. Specifically, ContextEn is not impacted by an enclave entry or exit. Hence, the generation of ContextEn-dependent packets within a debug enclave is allowed.

33.2.9.6 SENTER/ENTERACCS and ACM

GETSEC[SENTER] and GETSEC[ENTERACCS] instructions clear TraceEn, and it is not restored when those instruction complete. SENTER also causes TraceEn to be cleared on other logical processors when they rendezvous and enter the SENTER sleep state. In these two cases, the disabling of packet generation is not guaranteed to flush internally buffered packets. Some packets may be dropped.

When executing an authenticated code module (ACM), packet generation is silently disabled during ACRAM setup. TraceEn will be cleared, but no TIP.PGD packet is generated. After completion of the module, the TraceEn value will be restored. There will be no TIP.PGE packet, but timing packets, like TSC and CBR, may be produced.

33.2.9.7 Intel® Memory Protection Extensions (Intel® MPX)

Bounds exceptions (#BR) caused by Intel MPX are treated like other exceptions, producing FUP and TIP packets that indicate the source and destination IPs.

33.3 CONFIGURATION AND PROGRAMMING GUIDELINE

33.3.1 Detection of Intel Processor Trace and Capability Enumeration

Processor support for Intel Processor Trace is indicated by CPUID.(EAX=07H,ECX=0H):EBX[bit 25] = 1. CPUID function 14H is dedicated to enumerate the resource and capability of processors that report CPUID.(EAX=07H,ECX=0H):EBX[bit 25] = 1. Different processor generations may have architecturally-defined variation in capabilities. Table 33-11 describes details of the enumerable capabilities that software must use across generations of processors that support Intel Processor Trace.

Table 33-11. CPUID Leaf 14H Enumeration of Intel Processor Trace Capabilities

CPUID.(EAX=14H,ECX=0)		Name	Description Behavior
Register	Bits		
EAX	31:0	Maximum valid sub-leaf Index	Specifies the index of the maximum valid sub-leaf for this CPUID leaf.
EBX	0	CR3 Filtering Support	1: Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed. See Section 33.2.8. 0: Indicates that writes that set IA32_RTIT_CTL.CR3Filter to 1, or any access to IA32_RTIT_CR3_MATCH, will generate a #GP exception.
	1	Configurable PSB and Cycle-Accurate Mode Supported	1: (a) IA32_RTIT_CTL.PSBFreq can be set to a non-zero value, in order to select the preferred PSB frequency (see below for allowed values). (b) IA32_RTIT_STATUS.PacketByteCnt can be set to a non-zero value, and will be incremented by the processor when tracing to indicate progress towards the next PSB. If trace packet generation is enabled by setting TraceEn, a PSB will only be generated if PacketByteCnt=0. (c) IA32_RTIT_CTL.CYCEn can be set to 1 to enable Cycle-Accurate Mode. See Section 33.2.8. 0: (a) Any attempt to write a non-zero value to IA32_RTIT_CTL.PSBFreq or IA32_RTIT_STATUS.PacketByteCnt will generate a #GP exception. (b) If trace packet generation is enabled by setting TraceEn, a PSB is always generated. (c) Any attempt to write a non-zero value to IA32_RTIT_CTL.CYCEn or IA32_RTIT_CTL.CycThresh will generate a #GP exception.
	2	IP Filtering and TraceStop supported, and Preserve Intel PT MSRs across warm reset	1: (a) IA32_RTIT_CTL provides at one or more ADDRn_CFG field to configure the corresponding address range MSRs for IP Filtering or IP TraceStop. Each ADDRn_CFG field accepts a value in the range of 0:2 inclusive. The number of ADDRn_CFG fields is reported by CPUID.(EAX=14H, ECX=1):EAX.RANGECNT[2:0]. (b) At least one register pair IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B are provided to configure address ranges for IP filtering or IP TraceStop. (c) On warm reset, all Intel PT MSRs will retain their pre-reset values, though IA32_RTIT_CTL.TraceEn will be cleared. The Intel PT MSRs are listed in Section 33.2.8. 0: (a) An Attempt to write IA32_RTIT_CTL.ADDRn_CFG with non-zero encoding values will cause #GP. (b) Any access to IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B, will generate a #GP exception. (c) On warm reset, all Intel PT MSRs will be cleared.
	3	MTC Supported	1: IA32_RTIT_CTL.MTCEn can be set to 1, and MTC packets will be generated. See Section 33.2.8. 0: An attempt to set IA32_RTIT_CTL.MTCEn or IA32_RTIT_CTL.MTCFreq to a non-zero value will generate a #GP exception.
	4	PTWRITE Supported	1: Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets. 0: Writes that set IA32_RTIT_CTL[12] or IA32_RTIT_CTL[5] will generate a #GP exception, and PTWRITE will #UD fault.
	5	Power Event Trace Supported	1: Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation. 0: Writes that set IA32_RTIT_CTL[4] will generate a #GP exception.

Table 33-11. CPUID Leaf 14H Enumeration of Intel Processor Trace Capabilities (Contd.)

CPUID.(EAX=14H,ECX=0)		Name	Description Behavior
Register	Bits		
	6	PSB and PMI Preservation Supported	1: 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. 0: Writes that set IA32_RTIT_CTL[56], IA32_RTIT_STATUS[7], or IA32_RTIT_STATUS[6] will generate a #GP exception.
	7	Event Trace Supported	1: Writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation. 0: Writes that set IA32_RTIT_CTL[31] will generate a #GP exception.
	8	TNT Disable Supported	1: Writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation. 0: Writes that set IA32_RTIT_CTL[55] will generate a #GP exception.
	31:9	Reserved	
ECX	0	ToPA Output Supported	1: Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme (Section 33.2.7.2) IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. 0: Unless CPUID.(EAX=14H, ECX=0):ECX.SNGLRNGOUT[bit 2] = 1. writes to IA32_RTIT_OUTPUT_BASE or IA32_RTIT_OUTPUT_MASK_PTRS. MSRs will generate a #GP exception.
	1	ToPA Tables Allow Multiple Output Entries	1: ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. 0: ToPA tables can hold only one output entry, which must be followed by an END=1 entry which points back to the base of the table. Further, ToPA PMIs will be delivered before the region is filled. See ToPA PMI in Section 33.2.7.2. If there is more than one output entry before the END entry, or if the END entry has the wrong base address, an operational error will be signaled (see "ToPA Errors" in Section 33.2.7.2).
	2	Single-Range Output Supported	1: Enabling tracing (TraceEn=1) with IA32_RTIT_CTL.ToPA=0 is supported. 0: Unless CPUID.(EAX=14H, ECX=0):ECX.TOPAOUT[bit 0] = 1. writes to IA32_RTIT_OUTPUT_BASE or IA32_RTIT_OUTPUT_MASK_PTRS. MSRs will generate a #GP exception.
	3	Output to Trace Transport Subsystem Supported	1: Setting IA32_RTIT_CTL.FabricEn to 1 is supported. 0: IA32_RTIT_CTL.FabricEn is reserved. Write 1 to IA32_RTIT_CTL.FabricEn will generate a #GP exception.
	30:4	Reserved	
	31	IP Payloads are LIP	1: Generated packets which contain IP payloads have LIP values, which include the CS base component. 0: Generated packets which contain IP payloads have RIP values, which are the offset from CS base.
EDX	31:0	Reserved	

If CPUID.(EAX=14H, ECX=0):EAX reports a non-zero value, additional capabilities of Intel Processor Trace are described in the sub-leaves of CPUID leaf 14H.

Table 33-12. CPUID Leaf 14H, sub-leaf 1H Enumeration of Intel Processor Trace Capabilities

CPUID.(EAX=14H,ECX=1)		Name	Description Behavior
Register	Bits		
EAX	2:0	Number of Address Ranges	A non-zero value specifies the number ADDRn_CFG field supported in IA32_RTIT_CTL and the number of register pair IA32_RTIT_ADDRn_A/IA32_RTIT_ADDRn_B supported for IP filtering and IP TraceStop. **NOTE:** Currently, no processors support more than 4 address ranges.
	15:3	Reserved	
	31:16	Bitmap of supported MTC Period Encodings	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.MTCFreq field. This applies only if CPUID.(EAX=14H, ECX=0):EBX[bit 3] = 1 (MTC Packet generation is supported), otherwise the MTCFreq field is reserved to 0. Each bit position in this field represents 1 encoding value in the 4-bit MTCFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: MTCFreq can be assigned the associated encoding value. 0: MTCFreq cannot be assigned to the associated encoding value. A write to IA32_RTIT_CTLMTCFreq with unsupported encoding will cause #GP fault.
EBX	15:0	Bitmap of supported Cycle Threshold values	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.CycThresh field. This applies only if CPUID.(EAX=14H, ECX=0):EBX[bit 1] = 1 (Cycle-Accurate Mode is Supported), otherwise the CycThresh field is reserved to 0. See Section 33.2.8. Each bit position in this field represents 1 encoding value in the 4-bit CycThresh field (ie, bit 0 is associated with encoding value 0). For each bit: 1: CycThresh can be assigned the associated encoding value. 0: CycThresh cannot be assigned to the associated encoding value. A write to CycThresh with unsupported encoding will cause #GP fault.
	31:16	Bitmap of supported Configurable PSB Frequency encoding	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.PSBFreq field. This applies only if CPUID.(EAX=14H, ECX=0):EBX[bit 1] = 1 (Configurable PSB is supported), otherwise the PSBFreq field is reserved to 0. See Section 33.2.8. Each bit position in this field represents 1 encoding value in the 4-bit PSBFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: PSBFreq can be assigned the associated encoding value. 0: PSBFreq cannot be assigned to the associated encoding value. A write to PSBFreq with unsupported encoding will cause #GP fault.
ECX	31:0	Reserved	
EDX	31:0	Reserved	

33.3.1.1 Packet Decoding of RIP versus LIP

FUP, TIP, TIP.PGE, and TIP.PGE packets can contain an instruction pointer (IP) payload. On some processor generations, this payload will be an effective address (RIP), while on others this will be a linear address (LIP). In the former case, the payload is the offset from the current CS base address, while in the latter it is the sum of the offset and the CS base address (Note that in real mode, the CS base address is the value of CS<<4, while in protected mode the CS base address is the base linear address of the segment indicated by the CS register.). Which IP type is in use is indicated by enumeration (see CPUID.(EAX=14H, ECX=0):ECX.LIP[bit 31] in Table 33-11).

For software that executes while the CS base address is 0 (including all software executing in 64-bit mode), the difference is indistinguishable. A trace decoder must account for cases where the CS base address is not 0 and the resolved LIP will not be evident in a trace generated on a CPU that enumerates use of RIP. This is likely to cause problems when attempting to link the trace with the associated binaries.

Note that IP comparison logic, for IP filtering and TraceStop range calculation, is based on the same IP type as these IP packets. For processors that output RIP, the IP comparison mechanism is also based on RIP, and hence on those processors RIP values should be written to IA32_RTIT_ADDRn_[AB] MSRs. This can produce differing behavior if the same trace configuration setting is run on processors reporting different IP types, i.e., CPUID.(EAX=14H, ECX=0):ECX.LIP[bit 31]. Care should be taken to check CPUID when configuring IP filters.

33.3.1.2 Model Specific Capability Restrictions

Some processor generations impose restrictions that prevent use of LBRs/BTS/BTM/LERs when software has enabled tracing with Intel Processor Trace. On these processors, when TraceEn is set, updates of LBR, BTS, BTM, LERs are suspended but the states of the corresponding IA32_DEBUGCTL control fields remained unchanged as if it were still enabled. When TraceEn is cleared, the LBR array is reset, and LBR/BTS/BTM/LERs updates will resume. Further, reads of these registers will return 0, and writes will be dropped.

The list of MSRs whose updates/accesses are restricted follows.

- MSR_LASTBRANCH_x_TO_IP, MSR_LASTBRANCH_x_FROM_IP, MSR_LBR_INFO_x, MSR_LASTBRANCH_TOS
- MSR_LER_FROM_LIP, MSR_LER_TO_LIP
- MSR_LBR_SELECT

For processors with CPUID DisplayFamily_DisplayModel signatures of 06_3DH, 06_47H, 06_4EH, 06_4FH, 06_56H, and 06_5EH, the use of Intel PT and LBRs are mutually exclusive.

33.3.2 Enabling and Configuration of Trace Packet Generation

To configure trace packets, enable packet generation, and capture packets, software starts with using CPUID instruction to detect its feature flag, CPUID.(EAX=07H,ECX=0H):EBX[bit 25] = 1; followed by enumerating the capabilities described in Section 33.3.1.

Based on the capability queried from Section 33.3.1, software must configure a number of model-specific registers. This section describes programming considerations related to those MSRs.

33.3.2.1 Enabling Packet Generation

When configuring and enabling packet generation, the IA32_RTIT_CTL MSR should be written after any other Intel PT MSRs have been written, since writes to the other configuration MSRs cause a general-protection fault (#GP) if TraceEn = 1. If a prior trace collection context is not being restored, then software should first clear IA32_RTIT_STATUS. This is important since the Stopped, and Error fields are writable; clearing the MSR clears any values that may have persisted from prior trace packet collection contexts. See Section 33.2.8.2 for details of packets generated by setting TraceEn to 1.

If setting TraceEn to 1 causes an operational error (see Section 33.3.10), there may be a delay after the WRMSR completes before the error is signaled in the IA32_RTIT_STATUS MSR.

While packet generation is enabled, the values of some configuration MSRs (e.g., IA32_RTIT_STATUS and IA32_RTIT_OUTPUT_*) are transient, and reads may return values that are out of date. Only after packet generation is disabled (by clearing TraceEn) do reads of these MSRs return reliable values.

33.3.2.2 Disabling Packet Generation

After disabling packet generation by clearing IA32_RTIT_CTL, it is advisable to read the IA32_RTIT_STATUS MSR (Section 33.2.8.4):

- If the Error bit is set, an operational error was encountered, and the trace is most likely compromised. Software should check the source of the error (by examining the output MSR values), correct the source of the problem, and then attempt to gather the trace again. For details on operational errors, see Section 33.3.10. Software should clear IA32_RTIT_STATUS.Error before re-enabling packet generation.
- If the Stopped bit is set, software execution encountered an IP TraceStop (see Section 33.2.5.3) or the ToPA Stop condition (see "ToPA STOP" in Section 33.2.7.2) before packet generation was disabled.

33.3.3 Flushing Trace Output

Packets are first buffered internally and then written out asynchronously. To collect packet output for post-processing, a collector needs first to ensure that all packet data has been flushed from internal buffers. Software can ensure this by stopping packet generation by clearing IA32_RTIT_CTL.TraceEn (see "Disabling Packet Generation" in Section 33.2.8.2).

When software clears IA32_RTIT_CTL.TraceEn to flush out internally buffered packets, the logical processor issues an SFENCE operation which ensures that WC trace output stores will be ordered with respect to the next store, or serializing operation. A subsequent read from the same logical processor will see the flushed trace data, while a read from another logical processor should be preceded by a store, fence, or architecturally serializing operation on the tracing logical processor.

When the flush operations complete, the IA32_RTIT_OUTPUT_* MSR values indicate where the trace ended. While TraceEn is set, these MSRs may hold stale values. Further, if a ToPA region with INT=1 is filled, meaning a ToPA PMI has been triggered, IA32_PERF_GLOBAL_STATUS.Trace_ToPA_PMI[55] will be set by the time the flush completes.

33.3.4 Warm Reset

The MSRs software uses to program Intel Processor Trace are cleared after a power-on RESET (or cold RESET). On a warm RESET, the contents of those MSRs can retain their values from before the warm RESET with the exception that IA32_RTIT_CTL.TraceEn will be cleared (which may have the side effect of clearing some bits in IA32_RTIT_STATUS).

33.3.5 Context Switch Consideration

To facilitate construction of instruction execution traces at the granularity of a software process or thread context, software can save and restore the states of the trace configuration MSRs across the process or thread context switch boundary. The principle is the same as saving and restoring the typical architectural processor states across context switches.

33.3.5.1 Manual Trace Configuration Context Switch

The configuration can be saved and restored through a sequence of instructions of RDMSR, management of MSR content and WRMSR. To stop tracing and to ensure that all configuration MSRs contain stable values, software must clear IA32_RTIT_CTL.TraceEn before reading any other trace configuration MSRs. The recommended method for saving trace configuration context manually follows:

1. RDMSR IA32_RTIT_CTL, save value to memory
2. WRMSR IA32_RTIT_CTL with saved value from RDMSR above and TraceEn cleared
3. RDMSR all other configuration MSRs whose values had changed from previous saved value, save changed values to memory

When restoring the trace configuration context, IA32_RTIT_CTL should be restored last:

1. Read saved configuration MSR values, aside from IA32_RTIT_CTL, from memory, and restore them with WRMSR
2. Read saved IA32_RTIT_CTL value from memory, and restore with WRMSR.

33.3.5.2 Trace Configuration Context Switch Using XSAVES/XRSTORS

On processors whose XSAVE feature set supports XSAVES and XRSTORS, the Trace configuration state can be saved using XSAVES and restored by XRSTORS, in conjunction with the bit field associated with supervisory state component in IA32_XSS. See Chapter 13, "Managing State Using the XSAVE Feature Set," of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

33.3.6 Cycle-Accurate Mode

Intel PT can be run in a cycle-accurate mode which enables CYC packets (see Section 33.4.2.14) that provide low-level information in the processor core clock domain. This cycle counter data in CYC packets can be used to compute IPC (Instructions Per Cycle), or to track wall-clock time on a fine-grain level.

To enable cycle-accurate mode packet generation, software should set IA32_RTIT_CTL.CYCEn=1. It is recommended that software also set TSCEn=1 anytime cycle-accurate mode is in use. With this, all CYC-eligible packets will be preceded by a CYC packet, the payload of which indicates the number of core clock cycles since the last CYC packet. In cases where multiple CYC-eligible packets are generated in a single cycle, only a single CYC will be generated before the CYC-eligible packets, otherwise each CYC-eligible packet will be preceded by its own CYC. The CYC-eligible packets are:

- TNT, TIP, TIP.PGE, TIP.PGD, MODE.EXEC, MODE.TSX, PIP, VMCS, OVF, MTC, TSC, PTWRITE, EXSTOP

TSC packets are generated when there is insufficient information to reconstruct wall-clock time, due to tracing being disabled (TriggerEn=0), or power down scenarios like a transition to a deep-sleep MWAIT C-state. In this case, the CYC that is generated along with the TSC will indicate the number of cycles actively tracing (those powered up, with TriggerEn=1) executed between the last CYC packet and the TSC packet. And hence the amount of time spent while tracing is inactive can be inferred from the difference in time between that expected based on the CYC value, and the actual time indicated by the TSC.

Additional CYC packets may be sent stand-alone, so that the processor can ensure that the decoder is aware of the number of cycles that have passed before the internal hardware counter wraps, or is reset due to other micro-architectural condition. There is no guarantee at what intervals these standalone CYC packets will be sent, except that they will be sent before the wrap occurs. An illustration is given below.

Example 33-1. An Illustrative CYC Packet Example

Time (cycles)	Instruction Snapshot	Generated Packets	Comment
x	call %eax	CYC(?), TIP	?Elapsed cycles from the previous CYC unknown
x + 2	call %ebx	CYC(2), TIP	1 byte CYC packet; 2 cycles elapsed from the previous CYC
x + 8	jnz Foo (not taken)	CYC(6)	1 byte CYC packet
x + 9	ret (compressed)		
x + 12	jnz Bar (taken)		
x + 16	ret (uncompressed)	TNT, CYC(8), TIP	1 byte CYC packet
x + 4111		CYC(4095)	2 byte CYC packet
x + 12305		CYC(8194)	3 byte CYC packet
x + 16332	mov cr3, %ebx	CYC(4027), PIP	2 byte CYC packet

33.3.6.1 Cycle Counter

The cycle counter is implemented in hardware (independent of the time stamp counter or performance monitoring counters), and is a simple incrementing counter that does not saturate, but rather wraps. The size of the counter is implementation specific.

The cycle counter is reset to zero any time that TriggerEn is cleared, and when a CYC packet is sent. The cycle counter will continue to count when ContextEn or FilterEn are cleared, and cycle packets will still be generated. It will not count during sleep states that result in Intel PT logic being powered-down, but will count up to the point where clocks are disabled, and resume counting once they are re-enabled.

33.3.6.2 Cycle Packet Semantics

Cycle-accurate mode adheres to the following protocol:

- All packets that precede a CYC packet represent instructions or events that took place before the CYC time.
- All packets that follow a CYC packet represent instructions or events that took place at the same time as, or after, the CYC time.
- The CYC-eligible packet that immediately follows a CYC packet represents an instruction or event that took place at the same time as the CYC time.

These items above give the decoder a means to apply CYC packets to a specific instruction in the assembly stream. Most packets represent a single instruction or event, and hence the CYC packet that precedes each of those packets represents the retirement time of that instruction or event. In the case of TNT packets, up to 6 conditional branches and/or compressed RETs may be contained in the packet. In this case, the preceding CYC packet provides the retirement time of the first branch in the packet. It is possible that multiple branches retired in the same cycle as that first branch in the TNT, but the protocol will not make that obvious. Also note that a MTC packet could be generated in the same cycle as the first JCC in the TNT packet. In this case, the CYC would precede both the MTC and the TNT, and apply to both.

Note that there are times when the cycle counter will stop counting, though cycle-accurate mode is enabled. After any such scenario, a CYC packet followed by TSC packet will be sent. See Section 33.8.3.2 to understand how to interpret the payload values

Multi-packet Instructions or Events

Some operations, such as interrupts or task switches, generate multiple packets. In these cases, multiple CYC packets may be sent for the operation, preceding each CYC-eligible packet in the operation. An example, using a task switch on a software interrupt, is shown below.

Example 33-2. An Example of CYC in the Presence of Multi-Packet Operations

Time (cycles)	Instruction Snapshot	Generated Packets
x	jnz Foo (not taken)	CYC(?)
x + 2	ret (compressed)	
x + 8	jnz Bar (taken)	
x + 9	jmp %eax	TNT, CYC(9), TIP
x + 12	jnz Bar (not taken)	CYC(3)
x + 32	int3 (task gate)	TNT, FUP, CYC(10), PIP, CYC(20), MODE.Exec, TIP

33.3.6.3 Cycle Thresholds

Software can opt to reduce the frequency of cycle packets, a trade-off to save bandwidth and intrusion at the expense of precision. This is done by utilizing a cycle threshold (see Section 33.2.8.2).

IA32_RTIT_CTL.CycThresh indicates to the processor the minimum number of cycles that must pass before the next CYC packet should be sent. If this value is 0, no threshold is used, and CYC packets can be sent every cycle in which a CYC-eligible packet is generated. If this value is greater than 0, the hardware will wait until the associated

number of cycles have passed since the last CYC packet before sending another. CPUID provides the threshold options for CycThresh, see Section 33.3.1.

Note that the cycle threshold does not dictate how frequently a CYC packet will be posted, it merely assigns the maximum frequency. If the cycle threshold is 16, a CYC packet can be posted no more frequently than every 16 cycles. However, once that threshold of 16 cycles has passed, it still requires a new CYC-eligible packet to be generated before a CYC will be inserted. Table 33-13 illustrates the threshold behavior.

Table 33-13. An Illustrative CYC Packet Example

Time (cycles)	Instruction Snapshot	Threshold			
		0	16	32	64
x	jmp %eax	CYC, TIP	CYC, TIP	CYC, TIP	CYC, TIP
x + 9	call %ebx	CYC, TIP	TIP	TIP	TIP
x + 15	call %ecx	CYC, TIP	TIP	TIP	TIP
x + 30	jmp %edx	CYC, TIP	CYC, TIP	TIP	TIP
x + 38	mov cr3, %eax	CYC, PIP	PIP	CYC, PIP	PIP
x + 46	jmp [%eax]	CYC, TIP	CYC, TIP	TIP	TIP
x + 64	call %edx	CYC, TIP	CYC, TIP	TIP	CYC,TIP
x + 71	jmp %edx	CYC, TIP	TIP	CYC,TIP	TIP

33.3.7 Decoder Synchronization (PSB+)

The PSB packet (Section 33.4.2.17) serves as a synchronization point for a trace-packet decoder. It is a pattern in the trace log for which the decoder can quickly scan to align packet boundaries. No legal packet combination can result in such a byte sequence. As such, it serves as the starting point for packet decode. To decode a trace log properly, the decoder needs more than simply to be aligned: it needs to know some state and potentially some timing information as well. The decoder should never need to retain any information (e.g., LastIP, call stack, compound packet event) across a PSB; all compound packet events will be completed before a PSB, and any compression state will be reset.

When a PSB packet is generated, it is followed by a PSBEND packet (Section 33.4.2.18). One or more packets may be generated in between those two packets, and these inform the decoder of the current state of the processor. These packets, known collectively as PSB+, should be interpreted as "status only", since they do not imply any change of state at the time of the PSB, nor are they associated directly with any instruction or event. Thus, the normal binding and ordering rules that apply to these packets outside of PSB+ can be ignored when these packets are between a PSB and PSBEND. They inform the decoder of the state of the processor at the time of the PSB.

PSB+ can include:

- Timestamp (TSC), if IA32_RTIT_CTL.TSCEn=1.
- Timestamp-MTC Align (TMA), if IA32_RTIT_CTL.TSCEn=1 && IA32_RTIT_CTL.MTCEn=1.
- Paging Information Packet (PIP), if ContextEn=1 and IA32_RTIT_CTL.OS=1. The non-root bit (NR) is set if the logical processor is in VMX non-root operation and the "conceal VMX from PT" VM-execution control is 0.
- VMCS packet, if either the logical is in VMX root operation or the logical processor is in VMX non-root operation and the "conceal VMX from PT" VM-execution control is 0.
- Core Bus Ratio (CBR).
- MODE.TSX, if ContextEn=1 and BranchEn = 1.
- MODE.Exec, if PacketEn=1 or (ContextEn=1 and IA32_RTIT_CTL.EventEn=1).
- Flow Update Packet (FUP), if PacketEn=1.

PSB is generated only when TriggerEn=1; hence PSB+ has the same dependencies. The ordering of packets within PSB+ is not fixed. Timing packets such as CYC and MTC may be generated between PSB and PSBEND, and their meanings are the same as outside PSB+.

A PSB+ can be lost in some scenarios. If IA32_RTIT_STATUS.TriggerEn is cleared just as the PSB threshold is reached, e.g., due to TraceEn being cleared, the PSB+ may not be generated. On processors that support PSB preservation (CPUID.(EAX=14H, ECX=0):EBX[bit 6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PSB+ that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendPSB[6] = 1. A PSB will be inserted, and PendPSB cleared, when PT is later re-enabled while PendPSB = 1.

Note that an overflow can occur during PSB+, and this could cause the PSBEND packet to be lost. For this reason, the OVF packet should also be viewed as terminating PSB+. If IA32_RTIT_STATUS.TriggerEn is cleared just as the PSB threshold is reached, the PSB+ may not be generated. TriggerEn can be cleared by a WRMSR that clears IA32_RTIT_CTL.TraceEn, a VM exit that clears IA32_RTIT_CTL.TraceEn, an #SMI, or any time that either IA32_RTIT_STATUS.Stopped is set (e.g., by a TraceStop or ToPA stop condition) or IA32_RTIT_STATUS.Error is set (e.g., by an Intel PT output error). On processors that support PSB preservation (CPUID.(EAX=14H, ECX=0):EBX[bit 6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PSB+ that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendPSB[6] = 1. A PSB will then be pended when the saved PT context is later restored.

33.3.8 Internal Buffer Overflow

In the rare circumstances when new packets need to be generated but the processor's dedicated internal buffers are all full, an "internal buffer overflow" occurs. On such an overflow packet generation ceases (as packets would need to enter the processor's internal buffer) until the overflow resolves. Once resolved, packet generation resumes.

When the buffer overflow is cleared, an OVF packet (Section 33.4.2.16) is generated, and the processor ensures that packets which follow the OVF are not compressed (IP compression or RET compression) against packets that were lost.

If IA32_RTIT_CTL.BranchEn = 1, the OVF packet will be followed by a FUP if the overflow resolves while PacketEn=1. If the overflow resolves while PacketEn = 0 no packet is generated, but a TIP.PGE will naturally be generated later, once PacketEn = 1. The payload of the FUP or TIP.PGE will be the Current IP of the first instruction upon which tracing resumes after the overflow is cleared. If the overflow resolves while PacketEn=1, only timing packets may come between the OVF and the FUP. If the overflow resolves while PacketEn=0, any other packets that are not dependent on PacketEn may come between the OVF and the TIP.PGE.

33.3.8.1 Overflow Impact on Enables

The address comparisons to ADDRn ranges, for IP filtering and TraceStop (Section 33.2.5.3), continue during a buffer overflow, and TriggerEn, ContextEn, and FilterEn may change during a buffer overflow. Like other packets, however, any TIP.PGE or TIP.PGD packets that would have been generated will be lost. Further, IA32_RTIT_STATUS.PacketByteCnt will not increment, since it is only incremented when packets are generated.

If a TraceStop event occurs during the buffer overflow, IA32_RTIT_STATUS.Stopped will still be set, tracing will cease as a result. However, the TraceStop packet, and any TIP.PGD that result from the TraceStop, may be dropped.

33.3.8.2 Overflow Impact on Timing Packets

Any timing packets that are generated during a buffer overflow will be dropped. If only a few MTC packets are dropped, a decoder should be able to detect this by noticing that the time value in the first MTC packet after the buffer overflow incremented by more than one. If the buffer overflow lasted long enough that 256 MTC packets are lost (and thus the MTC packet 'wraps' its 8-bit CTC value), then the decoder may be unable to properly understand the trace. This is not an expected scenario. No CYC packets are generated during overflow, even if the cycle counter wraps.

Note that, if cycle-accurate mode is enabled, the OVF packet will generate a CYC packet. Because the cycle counter counts during overflows, this CYC packet can provide the duration of the overflow. However, there is a risk that the cycle counter wrapped during the overflow, which could render this CYC misleading.

33.3.9 TNT Disable

Software can opt to omit TNT packets from control flow trace (BranchEn=1) by setting IA32_RTIT_CTL.DisTNT[bit 55]. This can dramatically reduce trace size. Results will vary by workload, but trace size reductions of 40-75% are typical, which will have a corresponding reduction in performance overhead and memory bandwidth consumption from Intel PT. However, omitting TNT packets means the decoder is not able to follow the full control flow trace, since conditional branch and compressed RET results won't be known. Thus, TNT Disable should be employed only for usages that do not depend on full control flow trace.

NOTE

To avoid loss of RET results with TNT Disable, software may wish to disable RET compression by setting IA32_RTIT_CTL.DisRETC[bit 11].

33.3.10 Operational Errors

Errors are detected as a result of packet output configuration problems, which can include output alignment issues, ToPA reserved bit violations, or overlapping packet output with restricted memory. See "ToPA Errors" in Section 33.2.7.2 for details on ToPA errors, and Section 33.2.7.4 for details on restricted memory errors. Operational errors are only detected and signaled when TraceEn=1.

When an operational error is detected, tracing is disabled and the error is logged. Specifically, IA32_RTIT_STATUS.Error is set, which will cause IA32_RTIT_STATUS.TriggerEn to be 0. This will disable generation of all packets. Some causes of operational errors may lead to packet bytes being dropped.

It should be noted that the timing of error detection may not be predictable. Errors are signaled when the processor encounters the problematic configuration. This could be as soon as packet generation is enabled but could also be later when the problematic entry or field needs to be used.

Once an error is signaled, software should disable packet generation by clearing TraceEn, diagnose and fix the error condition, and clear IA32_RTIT_STATUS.Error. At this point, packet generation can be re-enabled.

33.4 TRACE PACKETS AND DATA TYPES

This section details the data packets generated by Intel Processor Trace. It is useful for developers writing the interpretation code that will decode the data packets and apply it to the traced source code.

33.4.1 Packet Relationships and Ordering

This section introduces the concept of packet "binding", which involves determining the IP in a binary disassembly at which the change indicated by a given packet applies. Some packets have the associated IP as the payload (FUP, TIP), while for others the decoder need only search for the next instance of a particular instruction (or instructions) to bind the packet (TNT). However, in many cases, the decoder will need to consider the relationship between packets, and to use this packet context to determine how to bind the packet.

Section 33.4.1.1 below provides detailed descriptions of the packets, including how packets bind to IPs in the disassembly, to other packets, or to nothing at all. Many packets listed are simple to bind, because they are generated in only a few scenarios. Those that require more consideration are typically part of "compound packet events", such as interrupts, exceptions, and some instructions, where multiple packets are generated by a single operation (instruction or event). These compound packet events frequently begin with a FUP to indicate the source address (if it is not clear from the disassembly), and are concluded by a TIP or TIP.PGD packet that indicates the destination address (if one is provided). In this scenario, the FUP is said to be "coupled" with the TIP packet.

Other packets could be in between the coupled FUP and TIP packet. Timing packets, such as TSC, MTC, CYC, or CBR, could arrive at any time, and hence could intercede in a compound packet event. If an operation changes CR3 or the processor's mode of execution, a state update packet (i.e., PIP or MODE) is generated. The state changes indicated by these intermediate packets should be applied at the IP of the TIP* packet. A summary of compound packet events is provided in Table 33-14; see Section 33.4.1.1 for more per-packet details and Section 33.7 for more detailed packet generation examples.

Table 33-14. Compound Packet Event Summary

Event Type	Beginning	Middle	End	Comment
Unconditional, uncompressed control-flow transfer	FUP or none	Any combination of PIP, VMCS, MODE.Exec, or none	TIP or TIP.PGD	FUP only for asynchronous events. Order of middle packets may vary. PIP/VMCS/MODE only if the operation modifies the state tracked by these respective packets.
TSX Update	MODE.TSX, and (FUP or none)	None	TIP, TIP.PGD, or none	FUP TIP/TIP.PGD only for TSX abort cases.
Overflow	OVF	PSB, PSBEND, or none	FUP or TIP.PGE	FUP if overflow resolves while ContextEn=1, else TIP.PGE.

33.4.1.1 Packet Blocks

Packet blocks are a means to dump one or more groups of state values. Packet blocks begin with a Block Begin Packet (BBP), which indicates what type of state is held within the block. Following each BBP there may be one or more Block Item Packets (BIPs), which contain the state values. The block is terminated by either a Block End Packet (BEP) or another BBP indicating the start of a new block.

The BIP packet includes an ID value that, when combined with the Type field from the BBP that preceded it, uniquely identifies the state value held in the BIP payload. The size of each BIP packet payload is provided by the Size field in the preceding BBP packet.

Each block type can have up to 32 items defined for it. There is no guarantee, however, that each block of that type will hold all 32 items. For more details on which items to expect, see documentation on the specific block type of interest.

See the BBP packet description (Section 33.4.2.26) for details on packet block generation scenarios.

Packet blocks are entirely generated within an instruction or between instructions, which dictates the types of packets (aside from BIPs) that may be seen within a packet block. Packets that indicate control flow changes, or other indication of instruction completion, cannot be generated within a block. These are listed in the following table. Other packets, including timing packets, may occur between BBP and BEP.

Table 33-15. Packets Forbidden Between BBP and BEP

TNT
TIP, TIP.PGE, TIP.PGD
MODE.Exec, MODE.TSX
PIP, VMCS
TraceStop
PSB, PSBEND
PTW
MWAIT

It is possible to encounter an internal buffer overflow in the middle of a block. In such a case, it is guaranteed that packet generation will not resume in the middle of a block, and hence the OVF packet terminates the current block. Depending on the duration of the overflow, subsequent blocks may also be lost.

Decoder Implications

When a Block Begin Packet (BBP) is encountered, the decoder will need to decode some packets within the block differently from those outside a block. The Block Item Packet (BIP) header byte has the same encoding as a TNT packet outside of a block, but must be treated as a BIP header (with following payload) within one.

When an OVF packet is encountered, the decoder should treat that as a block ending condition. Packet generation will not resume within a block.

33.4.2 Packet Definitions

The following description of packet definitions are in tabular format. Figure 33-3 explains how to interpret them. Packet bits listed as "RSVD" are not guaranteed to be 0.

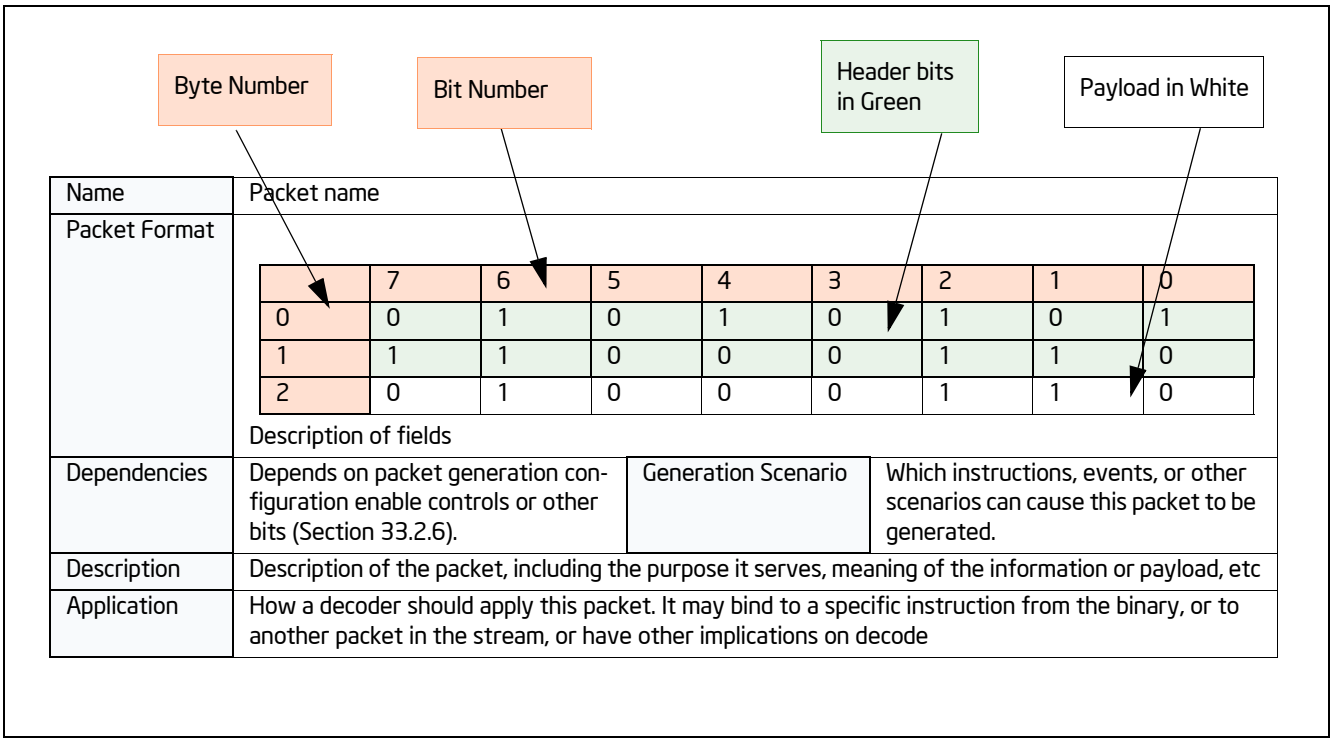


Figure 33-3. Interpreting Tabular Definition of Packet Format

33.4.2.1 Taken/Not-taken (TNT) Packet

Table 33-16. TNT Packet Definition

<table>
<tr><td>Name</td><td colspan="3">Taken/Not-taken (TNT) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>0</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>B_5</td><td>B_6</td><td>0</td><td>Short TNT</td></tr>
</table>
</td></tr>
<tr><td></td><td colspan="3">B1...BN represent the last N conditional branch or compressed RET (Section 33.4.2.2) results, such that B1 is oldest and BN is youngest. The short TNT packet can contain from 1 to 6 TNT bits. The long TNT packet can contain from 1 to 47 TNT bits.</td></tr>
<tr><td></td><td colspan="3">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td rowspan="8">Long TNT</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>B_{40}</td><td>B_{41}</td><td>B_{42}</td><td>B_{43}</td><td>B_{44}</td><td>B_{45}</td><td>B_{46}</td><td>B_{47}</td></tr>
<tr><td>3</td><td>B_{32}</td><td>B_{33}</td><td>B_{34}</td><td>B_{35}</td><td>B_{36}</td><td>B_{37}</td><td>B_{38}</td><td>B_{39}</td></tr>
<tr><td>4</td><td>B_{24}</td><td>B_{25}</td><td>B_{26}</td><td>B_{27}</td><td>B_{28}</td><td>B_{29}</td><td>B_{30}</td><td>B_{31}</td></tr>
<tr><td>5</td><td>B_{16}</td><td>B_{17}</td><td>B_{18}</td><td>B_{19}</td><td>B_{20}</td><td>B_{21}</td><td>B_{22}</td><td>B_{23}</td></tr>
<tr><td>6</td><td>B_8</td><td>B_9</td><td>B_{10}</td><td>B_{11}</td><td>B_{12}</td><td>B_{13}</td><td>B_{14}</td><td>B_{15}</td></tr>
<tr><td>7</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>B_5</td><td>B_6</td><td>B_7</td></tr>
</table>
</td></tr>
<tr><td></td><td colspan="3">Irrespective of how many TNT bits is in a packet, the last valid TNT bit is followed by a trailing 1, or Stop bit, as shown above. If the TNT packet is not full (fewer than 6 TNT bits for the Short TNT, or fewer than 47 TNT bits for the Long TNT), the Stop bit moves up, and the trailing bits of the packet are filled with 0s. Examples of these “partial TNTs” are shown below. An implementation may choose to use long TNTs, short TNTs, or both.
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>0</td><td>Short TNT</td></tr>
</table>
</td></tr>
<tr><td></td><td colspan="3">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>Long TNT</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td></td></tr>
<tr><td>2</td><td>B_{24}</td><td>B_{25}</td><td>B_{26}</td><td>B_{27}</td><td>B_{28}</td><td>B_{29}</td><td>B_{30}</td><td>B_{31}</td><td></td></tr>
<tr><td>3</td><td>B_{16}</td><td>B_{17}</td><td>B_{18}</td><td>B_{19}</td><td>B_{20}</td><td>B_{21}</td><td>B_{22}</td><td>B_{23}</td><td></td></tr>
<tr><td>4</td><td>B_8</td><td>B_9</td><td>B_{10}</td><td>B_{11}</td><td>B_{12}</td><td>B_{13}</td><td>B_{14}</td><td>B_{15}</td><td></td></tr>
<tr><td>5</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>B_5</td><td>B_6</td><td>B_7</td><td></td></tr>
<tr><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr>
<tr><td>7</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>PacketEn &&
~IA32_RTIT_CTL.DisTNT</td><td>Generation Scenario</td><td>On a conditional branch or compressed RET, if it fills the TNT. Also, partial TNTs may be generated at any time, as a result of other packets being generated, or certain micro-architectural conditions occurring, before the TNT is full.</td></tr>
</table>

Table 33-16. TNT Packet Definition (Contd.)

Description	Provides the taken/not-taken results for the last 1..6 (Short TNT) or 1..47 (Long TNT) conditional branches (Jcc, J*CXZ, or LOOP) or compressed RETs (Section 33.4.2.2). The TNT payload bits should be interpreted as follows: ▪ 1 indicates a taken conditional branch, or a compressed RET ▪ 0 indicates a not-taken conditional branch TNT payload bits are stored internal to the processor in a TNT buffer, until either the buffer is filled or another packet is to be generated. In either case a TNT packet holding the buffered bits will be emitted, and the TNT buffer will be marked as empty.
Application	Each valid payload bit (that is, bits between the header bits and the trailing Stop bit) applies to an upcoming conditional branch or RET instruction. Once a decoder consumes a TNT packet with N valid payload bits, these bits should be applied to (and hence provide the destination for) the next N conditional branches or RETs.

33.4.2.2 Target IP (TIP) Packet

Table 33-17. TIP Packet Definition

Name	Target IP (TIP) Packet		
Packet Format	<table><tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr><tr><td>0</td><td colspan="4">IPBytes</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td colspan="8">TargetIP[7:0]</td></tr><tr><td>2</td><td colspan="8">TargetIP[15:8]</td></tr><tr><td>3</td><td colspan="8">TargetIP[23:16]</td></tr><tr><td>4</td><td colspan="8">TargetIP[31:24]</td></tr><tr><td>5</td><td colspan="8">TargetIP[39:32]</td></tr><tr><td>6</td><td colspan="8">TargetIP[47:40]</td></tr><tr><td>7</td><td colspan="8">TargetIP[55:48]</td></tr><tr><td>8</td><td colspan="8">TargetIP[63:56]</td></tr></table>		
Dependencies	PacketEn	Generation Scenario	Indirect branch (including un-compressed RET), far branch, interrupt, exception, INIT, SIPI, VM exit, VM entry, TSX abort, EENTER, EEXIT, ERESUME, AEX[1].
Description	Provides the target for some control flow transfers		
Application	Anytime a TIP is encountered, it indicates that control was transferred to the IP provided in the payload.		
	The source of this control flow change, and hence the IP or instruction to which it binds, depends on the packets that precede the TIP. If a TIP is encountered and all preceding packets have already been bound, then the TIP will apply to the upcoming indirect branch, far branch, or VMRESUME. However, if there was a preceding FUP that remains unbound, it will bind to the TIP. Here, the TIP provides the target of an asynchronous event or TSX abort that occurred at the IP given in the FUP payload. Note that there may be other packets, in addition to the FUP, which will bind to the TIP packet. See the packet application descriptions for other packets for details.		

NOTES:

1. EENTER, EEXIT, ERESUME, AEX would be possible only for a debug enclave.

IP Compression

The IP payload in a TIP. FUP, TIP.PGE, or TIP.PGD packet can vary in size, based on the mode of execution, and the use of IP compression. IP compression is an optional compression technique the processor may choose to employ to reduce bandwidth. With IP compression, the IP to be represented in the payload is compared with the last IP sent out, via any of FUP, TIP, TIP.PGE, or TIP.PGD. If that previous IP had the same upper (most significant) address bytes, those matching bytes may be suppressed in the current packet. The processor maintains an internal state of the "Last IP" that was encoded in trace packets, thus the decoder will need to keep track of the "Last IP" state in

software, to match fidelity with packets generated by hardware. "Last IP" is initialized to zero, hence if the first IP in the trace may be compressed if the upper bytes are zeroes.

The "IPBytes" field of the IP packets (FUP, TIP, TIP.PGE, TIP.PGD) serves to indicate how many bytes of payload are provided, and how the decoder should fill in any suppressed bytes. The algorithm for reconstructing the IP for a TIP/FUP packet is shown in the table below.

Table 33-18. FUP/TIP IP Reconstruction

IPBytes	Uncompressed IP Value							
	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
000b	None, IP is out of context							
001b	Last IP[63:16]						IP Payload[15:0]	
010b	Last IP[63:32]				IP Payload[31:0]			
011b	IP Payload[47] extended		IP Payload[47:0]					
100b	Last IP [63:48]		IP Payload[47:0]					
101b	Reserved							
110b	IP Payload[63:0]							
111b	Reserved							

The processor-internal Last IP state is guaranteed to be reset to zero when a PSB is sent out. This means that the IP that follows the PSB with either be un-compressed (011b or 110b, see Table 33-18), or compressed against zero.

At times, "IPbytes" will have a value of 0. As shown above, this does not mean that the IP payload matches the full address of the last IP, but rather that the IP for this packet was suppressed. This is used for cases where the IP that applies to the packet is out of context. An example is the TIP.PGD sent on a SYSCALL, when tracing only USR code. In that case, no TargetIP will be included in the packet, since that would expose an instruction point at CPL = 0. When the IP payload is suppressed in this manner, Last IP is not cleared, and instead refers to the last IP packet with a non-zero IPBytes field.

On processors that support a maximum linear address size of 32 bits, IP payloads may never exceed 32 bits (IPBytes <= 010b).

Indirect Transfer Compression for Returns (RET)

In addition to IP compression, TIP packets for near return (RET) instructions can also be compressed. If the RET target matches the next IP of the corresponding CALL, then the TIP packet is unneeded, since the decoder can deduce the target IP by maintaining a CALL/RET stack of its own.

When a RET is compressed, a Taken indication is added to the TNT buffer. Because the RET generates no TIP packet, it also does not update the internal Last IP value, and thus the decoder should treat it the same way. If the RET is not compressed, it will generate a TIP packet (just like when RET compression is disabled, via IA32_RTIT_CTL.DisRETC).

A CALL/RET stack can be maintained by the decoder by doing the following:

1. Allocate space to store 64 RET targets.
2. For near CALLs, push the Next IP onto the stack. Once the stack is full, new CALLs will force the oldest entry off the end of the stack, such that only the youngest 64 entries are stored. Note that this excludes zero-length CALLs, which are direct near CALLs with displacement zero (to the next IP). These CALLs typically don't have matching RETs.
3. For near RETs, pop the top (youngest) entry off the stack. This will be the expected target of the RET.

In cases where a RET is compressed, the RET target is guaranteed to match the expected target from 3) above. If the target is not compressed, a TIP packet will be generated with the RET target, which may differ from the expected target in some cases.

The hardware ensures that packets read by the decoder will always have seen the CALL that corresponds to any compressed RET. The processor will never compress a RET across a PSB, a buffer overflow, or scenario where PacketEn=0. This means that a RET whose corresponding CALL executed while PacketEn=0, or before the last PSB, etc., will not be compressed.

If the CALL/RET stack is manipulated or corrupted by software, and thereby causes a RET to transfer control to a target that is inconsistent with the CALL/RET stack, then the RET will not be compressed, and will produce a TIP packet. This can happen, for example, if software executes a PUSH instruction to push a target onto the stack, and a later RET uses this target.

For processors that employ deferred TIPs (Section 33.4.2.3), an uncompressed RET will not be deferred, and hence will force out any accumulated TNTs or TIPs. This serves to avoid ambiguity, and make clear to the decoder whether the near RET was compressed, and hence a bit in the in-progress TNT should be consumed, or uncompressed, in which case there will be no in-progress TNT and thus a TIP should be consumed.

Note that in the unlikely case that a RET executes in a different execution mode than the associated CALL, the decoder will need to model the same behavior with its CALL stack. For instance, if a CALL executes in 64-bit mode, a 64-bit IP value will be pushed onto the software stack. If the corresponding RET executes in 32-bit mode, then only the lower 32 target bits will be popped off of the stack, which may mean that the RET does not go to the CALL's Next IP. This is architecturally correct behavior, and this RET could be compressed, thus the decoder should match this behavior.

33.4.2.3 Deferred TIPs

The processor may opt to defer sending out the TNT when TIPs are generated. Thus, rather than sending a partial TNT followed by a TIP, both packets will be deferred while the TNT accumulates more Jcc/RET results. Any number of TIP packets may be accumulated this way, such that only once the TNT is filled, or once another packet (e.g., FUP) is generated, the TNT will be sent, followed by all the deferred TIP packets, and finally terminated by the other packet(s) that forced out the TNT and TIP packets. Generation of many other packets (see list below) will force out the TNT and any accumulated TIP packets. This is an optional optimization in hardware to reduce the bandwidth consumption, and hence the performance impact, incurred by tracing.

Table 33-19. TNT Examples with Deferred TIPs

Code Flow	Packets, Non-Deferred TIPS	Packets, Deferred TIPS
0x1000 cmp %rcx, 0 0x1004 jnz Foo // not-taken 0x1008 jmp %rdx	TNT(0b0), TIP(0x1308)	
0x1308 cmp %rcx, 1 0x130c jnz Bar // not-taken 0x1310 cmp %rcx, 2 0x1314 jnz Baz // taken 0x1500 cmp %eax, 7 0x1504 jg Exit // not-taken 0x1508 jmp %r15	TNT(0b010), TIP(0x1100)	
0x1100 cmp %rbx, 1 0x1104 jg Start // not-taken 0x1108 add %rcx, %eax 0x110c ... // **an asynchronous Interrupt arrives** INThandler: 0xcc00 pop %rdx	TNT(0b0), FUP(0x110c), TIP(0xcc00)	TNT(0b00100), TIP(0x1308), TIP(0x1100), FUP(0x110c), TIP(0xcc00)

33.4.2.4 Packet Generation Enable (TIP.PGE) Packet

Table 33-20. TIP.PGE Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Target IP - Packet Generation Enable (TIP.PGE) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td colspan="3">IPBytes</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td colspan="8">TargetIP[7:0]</td></tr>
<tr><td>2</td><td colspan="8">TargetIP[15:8]</td></tr>
<tr><td>3</td><td colspan="8">TargetIP[23:16]</td></tr>
<tr><td>4</td><td colspan="8">TargetIP[31:24]</td></tr>
<tr><td>5</td><td colspan="8">TargetIP[39:32]</td></tr>
<tr><td>6</td><td colspan="8">TargetIP[47:40]</td></tr>
<tr><td>7</td><td colspan="8">TargetIP[55:48]</td></tr>
<tr><td>8</td><td colspan="8">TargetIP[63:56]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>PacketEn transitions to 1</td><td>Generation Scenario</td><td colspan="2">Any branch instruction, control flow transfer, or MOV CR3 that sets PacketEn, a WRMSR that enables packet generation and sets PacketEn</td></tr>
<tr><td>Description</td><td colspan="4">Indicates that PacketEn has transitioned to 1. It provides the IP at which the tracing begins.
This can occur due to any of the enables that comprise PacketEn transitioning from 0 to 1, as long as all the others are asserted. Examples:
▪ TriggerEn: This is set on software write to set IA32_RTIT_CTL.TraceEn as long as the Stopped and Error bits in IA32_RTIT_STATUS are clear. The IP payload will be the Next IP of the WRMSR.
▪ FilterEn: This is set when software jumps into the tracing region. This region is defined by enabling IP filtering in IA32_RTIT_CTL.ADDRn_CFG, and defining the range in IA32_RTIT_ADDRn_[AB], see. Section 33.2.5.3. The IP payload will be the target of the branch.
▪ ContextEn: This is set on a CPL change, a CR3 write or any other means of changing ContextEn. The IP payload will be the Next IP of the instruction that changes context if it is not a branch, otherwise it will be the target of the branch.</td></tr>
<tr><td>Application</td><td colspan="4">TIP.PGE packets bind to the instruction at the IP given in the payload.</td></tr>
</table>

33.4.2.5 Packet Generation Disable (TIP.PGD) Packet

Table 33-21. TIP.PGD Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Target IP - Packet Generation Disable (TIP.PGD) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td colspan="3">IPBytes</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td colspan="8">TargetIP[7:0]</td></tr>
<tr><td>2</td><td colspan="8">TargetIP[15:8]</td></tr>
<tr><td>3</td><td colspan="8">TargetIP[23:16]</td></tr>
<tr><td>4</td><td colspan="8">TargetIP[31:24]</td></tr>
<tr><td>5</td><td colspan="8">TargetIP[39:32]</td></tr>
<tr><td>6</td><td colspan="8">TargetIP[47:40]</td></tr>
<tr><td>7</td><td colspan="8">TargetIP[55:48]</td></tr>
<tr><td>8</td><td colspan="8">TargetIP[63:56]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>PacketEn transitions to 0</td><td>Generation Scenario</td><td colspan="2">Any branch instruction, control flow transfer, or MOV CR3 that clears PacketEn, a WRMSR that disables packet generation and clears PacketEn</td></tr>
<tr><td>Description</td><td colspan="4">Indicates that PacketEn has transitioned to 0. It will include the IP at which the tracing ends, unless ContextEn= 0 or TraceEn=0 at the conclusion of the instruction or event that cleared PacketEn.
PacketEn can be cleared due to any of the enables that comprise PacketEn transitioning from 1 to 0. Examples:
▪ TriggerEn: This is cleared on software write to clear IA32_RTIT_CTL.TraceEn, or when IA32_RTIT_STATUS.Stopped is set, or on operational error. The IP payload will be suppressed in this case, and the “IPBytes” field will have the value 0.
▪ FilterEn: This is cleared when software jumps out of the tracing region. This region is defined by enabling IP filtering in IA32_RTIT_CTL.ADDRn_CFG, and defining the range in IA32_RTIT_ADDRn_[AB], see. Section 33.2.5.3. The IP payload will depend on the type of the branch. For conditional branches, the payload is suppressed (IPBytes = 0), and in this case the destination can be inferred from the disassembly. For any other type of branch, the IP payload will be the target of the branch.
▪ ContextEn: This can happen on a CPL change, a CR3 write or any other means of changing ContextEn. See Section 33.2.5.3 for details. In this case, when ContextEn is cleared, there will be no IP payload. The “IPBytes” field will have value 0.
Note that, in cases where a branch that would normally produce a TIP packet (i.e., far transfer, indirect branch, interrupt, etc) or TNT update (conditional branch or compressed RT) causes PacketEn to transition from 1 to 0, the TIP or TNT bit will be replaced with TIP.PGD. The payload of the TIP.PGD will be the target of the branch, unless the result of the instruction causes TraceEn or ContextEn to be cleared (ie, SYSCALL when IA32_RTIT_CTL.OS=0, In the case where a conditional branch clears FilterEn and hence PacketEn, there will be no TNT bit for this branch, replaced instead by the TIP.PGD.</td></tr>
<tr><td>Application</td><td colspan="4">TIP.PGD can be produced by any branch instructions, as well as some non-branch instructions, that clear PacketEn. When produced by a branch, it replaces any TIP or TNT update that the branch would normally produce.
In cases where there is an unbound FUP preceding the TIP.PGD, then the TIP.PGD is part of compound operation (i.e., asynchronous event or TSX abort) which cleared PacketEn. For most such cases, the TIP.PGD is simply replacing a TIP, and should be treated the same way. The TIP.PGD may or may not have an IP payload, depending on whether the operation cleared ContextEn.
If there is not an associated FUP, the binding will depend on whether there is an IP payload. If there is an IP payload, then the TIP.PGD should be applied to either the next direct branch whose target matches the TIP.PGD payload, or the next branch that would normally generate a TIP or TNT packet. If there is no IP payload, then the TIP.PGD should apply to the next branch or MOV CR3 instruction.</td></tr>
</table>

33.4.2.6 Flow Update (FUP) Packet

Table 33-22. FUP Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Flow Update (FUP) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td colspan="3">IPBytes</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td colspan="8">IP[7:0]</td></tr>
<tr><td>2</td><td colspan="8">IP[15:8]</td></tr>
<tr><td>3</td><td colspan="8">IP[23:16]</td></tr>
<tr><td>4</td><td colspan="8">IP[31:24]</td></tr>
<tr><td>5</td><td colspan="8">IP[39:32]</td></tr>
<tr><td>6</td><td colspan="8">IP[47:40]</td></tr>
<tr><td>7</td><td colspan="8">IP[55:48]</td></tr>
<tr><td>8</td><td colspan="8">IP[63:56]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn. (Typically depends on BranchEn and FilterEn as well, see Section 33.2.5, Section 33.4.2.21, and Section 33.4.2.22 for details.)</td><td>Generation Scenario</td><td colspan="2">Asynchronous Events (interrupts, exceptions, INIT, SIPI, SMI, VM exit, #MC), PSB+, XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, EENTER, EEXIT, ERESUME, EEE, AEX,[1], INTO, INT1, INT3, INT n, a WRMSR that disables packet generation.</td></tr>
<tr><td>Description</td><td colspan="4">Provides the source address for asynchronous events, and some other instructions. Is never sent alone, always sent with an associated TIP or MODE packet, and potentially others.</td></tr>
<tr><td>Application</td><td colspan="4">FUP packets provide the IP to which they bind. However, they are never standalone, but are coupled with other packets.
In TSX cases, the FUP is immediately preceded by a MODE.TSX, which binds to the same IP. A TIP will follow only in the case of TSX aborts, see Section 33.4.2.8 for details.
Otherwise, FUPs are part of compound packet events (see Section 33.4.1). In these compound cases, the FUP provides the source IP for an instruction or event, while a following TIP (or TIP.PGD) packet will provide the destination IP. Other packets may be included in the compound event between the FUP and TIP.</td></tr>
</table>

NOTES:

1. EENTER, EEXIT, ERESUME, EEE, AEX apply only if Intel Software Guard Extensions is supported.

FUP IP Payload

Flow Update Packet gives the source address of an instruction when it is needed. In general, branch instructions do not need a FUP, because the source address is clear from the disassembly. For asynchronous events, however, the source address cannot be inferred from the source, and hence a FUP will be sent. Table 33-23 illustrates cases where FUPs are sent, and which IP can be expected in those cases.

Table 33-23. FUP Cases and IP Payload

Event	Flow Update IP	Comment
External Interrupt, NMI/SMI, Traps, Machine Check (trap-like), INIT/SIPI	Address of next instruction (Next IP) that would have been executed	Functionally, this matches the LBR FROM field value and also the EIP value which is saved onto the stack.
Exceptions/Faults, Machine check (fault-like)	Address of the instruction which took the exception/fault (Current IP)	This matches the similar functionality of LBR FROM field value and also the EIP value which is saved onto the stack.
Software Interrupt	Address of the software interrupt instruction (Current IP)	This matches the similar functionality of LBR FROM field value, but does not match the EIP value which is saved onto the stack (Next Linear Instruction Pointer - NLIP).
EENTER, EEXIT, ERESUME, Enclave Exiting Event (EEE), AEX[1]	Current IP of the instruction	This matches the LBR FROM field value and also the EIP value which is saved onto the stack.
XACQUIRE	Address of the X* instruction	
XRELEASE, XBEGIN, XEND, XABORT, other transactional abort	Current IP	
#SMI	IP that is saved into SMRAM	
WRMSR that clears TraceEn, PSB+	Current IP	

NOTES:

1. Information on EENTER, EEXIT, ERESUME, EEE, Asynchronous Enclave eXit (AEX) can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D.

On a canonical fault due to sequentially fetching an instruction in non-canonical space (as opposed to jumping to non-canonical space), the IP of the fault (and thus the payload of the FUP) will be a non-canonical address. This is consistent with what is pushed on the stack for such faulting cases.

If there are post-commit task switch faults, the IP value of the FUP will be the original IP when the task switch started. This is the same value as would be seen in the LBR_FROM field. But it is a different value as is saved on the stack or VMCS.

33.4.2.7 Paging Information (PIP) Packet

Table 33-24. PIP Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Paging Information (PIP) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td colspan="6">CR3[11:5] or 0</td><td>RSVD/NR</td></tr>
<tr><td>3</td><td colspan="7">CR3[19:12]</td></tr>
<tr><td>4</td><td colspan="7">CR3[27:20]</td></tr>
<tr><td>5</td><td colspan="7">CR3[35:28]</td></tr>
<tr><td>6</td><td colspan="7">CR3[43:36]</td></tr>
<tr><td>7</td><td colspan="7">CR3[51:44]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn && IA32_RTIT_CTL.OS</td><td>Generation Scenario</td><td colspan="2">MOV CR3, Task switch, INIT, SIPI, PSB+, VM exit, VM entry</td></tr>
<tr><td>Description</td><td colspan="4">The CR3 payload shown includes only the address portion of the CR3 value. For PAE paging, CR3[11:5] are thus included. For other paging modes (32-bit and 4-level paging[1]), these bits are 0.
This packet holds the CR3 address value. It will be generated on operations that modify CR3:
▪ MOV CR3 operation
▪ Task Switch
▪ INIT and SIPI
▪ VM exit, if “conceal VMX from PT” VM-exit control is 0 (see Section 33.5.1)
▪ VM entry, if “conceal VMX from PT” VM-entry control is 0
PIPs are not generated, despite changes to CR3, on SMI and RSM. This is due to the special behavior on these operations, see Section 33.2.9.3 for details. Note that, for some cases of task switch where CR3 is not modified, no PIP will be produced.
The purpose of the PIP is to indicate to the decoder which application is running, so that it can apply the proper binaries to the linear addresses that are being traced.
The PIP packet contains the new CR3 value when CR3 is written.
PIPs generated by VM entries set the NR bit. PIPs generated in VMX non-root operation set the NR bit if the “conceal VMX from PT” VM-execution control is 0 (see Section 33.5.1). All other PIPs clear the NR bit.</td></tr>
<tr><td>Application</td><td colspan="4">The purpose of the PIP packet is to help the decoder uniquely identify what software is running at any given time. When a PIP is encountered, a decoder should do the following:
1) If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this PIP is part of a compound packet event (Section 33.4.1). Find the ending TIP and apply the new CR3/NR values to the TIP payload IP.
2) Otherwise, look for the next MOV CR3, far branch, or VMRESUME/VMLAUNCH in the disassembly, and apply the new CR3 to the next (or target) IP.
For examples of the packets generated by these flows, see Section 33.7.</td></tr>
</table>

NOTES:

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

33.4.2.8 MODE Packets

MODE packets keep the decoder informed of various processor modes about which it needs to know in order to properly manage the packet output, or to properly disassemble the associated binaries. MODE packets include a header and a mode byte, as shown below.

Table 33-25. General Form of MODE Packets

	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1
1	Leaf ID			Mode				

The MODE Leaf ID indicates which set of mode bits are held in the lower bits.

MODE.Exec Packet

Table 33-26. MODE.Exec Packet Definition

<table>
<tr><td>Name</td><td colspan="3">MODE.Exec Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td colspan="2">Reserved</td><td>IF</td><td>CS.D</td><td>(CS.L & LMA)</td></tr>
</table>
MODE Leaf ID is ‘000.
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn && FilterEn</td><td>Generation Scenario</td><td>Any operation that changes the CS.L, CS.D, or EFER.LMA, if IA32_RTIT_CTL.BranchEn=1.
Any operation that changes RFLAGS.IF, if IA32_RTIT_CTL.EventEn=1.
Any TIP.PGE scenario, such that any of the mode bits tracked may have changed since the last MODE.Exec.</td></tr>
<tr><td>Description</td><td colspan="3">Indicates whether software is in 16, 32, or 64-bit mode, by providing the CS.D and (CS.L & IA32_EFER.LMA) values. Essential for the decoder to properly disassemble the associated binary. Further, if CPUID.0x14.0.EBX[6]=1 (“Event Trace Support”), it indicates when interrupts are masked by providing the RFLAGS.IF value.

MODE.Exec is sent at the time of a mode change, if dependencies are met at the time, otherwise it is sent when tracing resumes. In the former case, the MODE.Exec packet is generated along with other packets that result from the operation that changes the mode, and is guaranteed to be followed by a TIP or TIP.PGE for branch operations, or a FUP for non-branch operations (CLI, STI, or POPF if EventEn=1). In cases where the mode changes while filtering dependencies are not met, the processor ensures that the decoder doesn’t lose track of the mode by sending any needed MODE.Exec once tracing resumes (preceding the TIP.PGE, if BranchEn=1). The processor may opt to suppress the MODE.Exec when tracing resumes if the mode matches that of the last MODE.Exec packet.
MODE.Exec packets are generated on CS.L, CS.D, or EFER.LMA changes only if control flow tracing is enabled (BranchEn=1). This is essential for the decoder to properly disassemble the associated binary.

<table>
<tr><td>CS.D</td><td>(CS.L & IA32_EFER.LMA)</td><td>Addressing Mode</td></tr>
<tr><td>1</td><td>1</td><td>N/A</td></tr>
<tr><td>0</td><td>1</td><td>64-bit mode</td></tr>
<tr><td>1</td><td>0</td><td>32-bit mode</td></tr>
<tr><td>0</td><td>0</td><td>16-bit mode</td></tr>
</table>

MODE.Exec packets are generated on interrupt flag (RFLAGS.IF) changes only if event tracing is enabled (EventEn=1).</td></tr>
<tr><td>Application</td><td colspan="3">MODE.Exec always precedes an IP packet (TIP, TIP.PGE, or FUP). The mode change applies to the IP address in the payload of the IP packet. When MODE.Exec is followed by a FUP, it is a stand-alone FUP and should be consumed by the MODE.Exec.</td></tr>
</table>

MODE.TSX Packet

Table 33-27. MODE.TSX Packet Definition

<table>
<tr><td>Name</td><td colspan="4">MODE.TSX Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>TXAbort</td><td>InTX</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn</td><td>Generation Scenario</td><td colspan="2">XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, if InTX changes, Asynchronous TSX Abort, PSB+</td></tr>
<tr><td>Description</td><td colspan="4">Indicates when a TSX transaction (either HLE or RTM) begins, commits, or aborts. Instructions executed transactionally will be “rolled back” if the transaction is aborted.
<table>
<tr><td>TXAbort</td><td>InTX</td><td>Implication</td></tr>
<tr><td>1</td><td>1</td><td>N/A</td></tr>
<tr><td>0</td><td>1</td><td>Transaction begins, or executing transactionally</td></tr>
<tr><td>1</td><td>0</td><td>Transaction aborted</td></tr>
<tr><td>0</td><td>0</td><td>Transaction committed, or not executing transactionally</td></tr>
</table>
</td></tr>
<tr><td>Application</td><td colspan="4">If PacketEn=1, MODE.TSX always immediately precedes a FUP. If the TXAbort bit is zero, then the mode change applies to the IP address in the payload of the FUP. If TXAbort=1, then the FUP will be followed by a TIP, and the mode change will apply to the IP address in the payload of the TIP.
MODE.TSX packets may be generated when PacketEn=0, due to FilterEn=0. In this case, only the last MODE.TSX generated before TIP.PGE need be applied.</td></tr>
</table>

33.4.2.9 TraceStop Packet

Table 33-28. TraceStop Packet Definition

<table>
<tr><td>Name</td><td colspan="4">TraceStop Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4"><table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn</td><td>Generation Scenario</td><td colspan="2">Taken branch with target in TraceStop IP region, MOV CR3 in TraceStop IP region, or WRMSR that sets TraceEn in TraceStop IP region.</td></tr>
<tr><td>Description</td><td colspan="4">Indicates when software has entered a user-configured TraceStop region.
When the IP matches a TraceStop range while ContextEn and TriggerEn are set, a TraceStop action occurs. This disables tracing by setting IA32_RTIT_STATUS.Stopped, thereby clearing TriggerEn, and causes a TraceStop packet to be generated.
The TraceStop action also forces FilterEn to 0. Note that TraceStop may not force a flush of internally buffered packets, and thus trace packet generation should still be manually disabled by clearing IA32_RTIT_CTL.TraceEn before examining output. See Section 33.2.5.3 for more details.</td></tr>
<tr><td>Application</td><td colspan="4">If TraceStop follows a TIP.PGD (before the next TIP.PGE), then it was triggered either by the instruction that cleared PacketEn, or it was triggered by some later instruction that executed while FilterEn=0. In either case, the TraceStop can be applied at the IP of the TIP.PGD (if any).
If TraceStop follows a TIP.PGE (before the next TIP.PGD), it should be applied at the last known IP.</td></tr>
</table>

33.4.2.10 Core:Bus Ratio (CBR) Packet

Table 33-29. CBR Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Core:Bus Ratio (CBR) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4"><table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td colspan="7">Core:Bus Ratio</td></tr>
<tr><td>3</td><td colspan="7">Reserved</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">After any frequency change, on C-state wake up, PSB+, and after enabling trace packet generation.</td></tr>
<tr><td>Description</td><td colspan="4">Indicates the core:bus ratio of the processor core. Useful for correlating wall-clock time and cycle time.</td></tr>
<tr><td>Application</td><td colspan="4">The CBR packet indicates the point in the trace when a frequency transition has occurred. On some implementations, software execution will continue during transitions to a new frequency, while on others software execution ceases during frequency transitions. There is not a precise IP provided, to which to bind the CBR packet.</td></tr>
</table>

33.4.2.11 Timestamp Counter (TSC) Packet

Table 33-30. TSC Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Timestamp Counter (TSC) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td colspan="8">SW TSC[7:0]</td></tr>
<tr><td>2</td><td colspan="8">SW TSC[15:8]</td></tr>
<tr><td>3</td><td colspan="8">SW TSC[23:16]</td></tr>
<tr><td>4</td><td colspan="8">SW TSC[31:24]</td></tr>
<tr><td>5</td><td colspan="8">SW TSC[39:32]</td></tr>
<tr><td>6</td><td colspan="8">SW TSC[47:40]</td></tr>
<tr><td>7</td><td colspan="8">SW TSC[55:48]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.TSCEn && TriggerEn</td><td>Generation Scenario</td><td colspan="2">Sent after any event that causes the processor clocks or Intel PT timing packets (such as MTC or CYC) to stop, This may include P-state changes, wake from C-state, or clock modulation. Also on transition of TraceEn from 0 to 1.</td></tr>
<tr><td>Description</td><td colspan="4">When enabled by software, a TSC packet provides the lower 7 bytes of the current TSC value, as returned by the RDTSC instruction. This may be useful for tracking wall-clock time, and synchronizing the packets in the log with other timestamped logs.</td></tr>
<tr><td>Application</td><td colspan="4">TSC packet provides a wall-clock proxy of the event which generated it (packet generation enable, sleep state wake, etc). In all cases, TSC does not precisely indicate the time of any control flow packets; however, all preceding packets represent instructions that executed before the indicated TSC time, and all subsequent packets represent instructions that executed after it. There is not a precise IP to which to bind the TSC packet.</td></tr>
</table>

33.4.2.12 Mini Time Counter (MTC) Packet

Table 33-31. MTC Packet Definition

<table>
<tr><td>Name</td><td colspan="3">Mini time Counter (MTC) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td colspan="8">CTC[N+7:N]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.MTCEn && TriggerEn</td><td>Generation Scenario</td><td>Periodic, based on the core crystal clock, or Always Running Timer (ART).</td></tr>
<tr><td>Description</td><td colspan="3">When enabled by software, an MTC packet provides a periodic indication of wall-clock time. The 8-bit CTC (Common Timestamp Copy) payload value is set to (ART >> N) & FFH. The frequency of the ART is related to the Maximum Non-Turbo frequency, and the ratio can be determined from CPUID leaf 15H, as described in Section 33.8.3.
Software can select the threshold N, which determines the MTC frequency by setting the IA32_RTIT_CTL.MTCFreq field (see Section 33.2.8.2) to a supported value using the lookup enumerated by CPUID (see Section 33.3.1).
See Section 33.8.3 for details on how to use the MTC payload to track TSC time.
MTC provides 8 bits from the ART, starting with the bit selected by MTCFreq to dictate the frequency of the packet. Whenever that 8-bit range being watched changes, an MTC packet will be sent out with the new value of that 8-bit range. This allows the decoder to keep track of how much wall-clock time has elapsed since the last TSC packet was sent, by keeping track of how many MTC packets were sent and what their value was. The decoder can infer the truncated bits, CTC[N-1:0], are 0 at the time of the MTC packet.
There are cases in which MTC packet can be dropped, due to overflow or other micro-architectural conditions. The decoder should be able to recover from such cases by checking the 8-bit payload of the next MTC packet, to determine how many MTC packets were dropped. It is not expected that >256 consecutive MTC packets should ever be dropped.</td></tr>
<tr><td>Application</td><td colspan="3">MTC does not precisely indicate the time of any other packet, nor does it bind to any IP. However, all preceding packets represent instructions or events that executed before the indicated ART time, and all subsequent packets represent instructions that executed after, or at the same time as, the ART time.</td></tr>
</table>

33.4.2.13 TSC/MTC Alignment (TMA) Packet

Table 33-32. TMA Packet Definition

<table>
<tr><td>Name</td><td colspan="4">TSC/MTC Alignment (TMA) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td colspan="8">CTC[7:0]</td></tr>
<tr><td>3</td><td colspan="8">CTC[15:8]</td></tr>
<tr><td>4</td><td colspan="7">Reserved</td><td>0</td></tr>
<tr><td>5</td><td colspan="8">FastCounter[7:0]</td></tr>
<tr><td>6</td><td colspan="7">Reserved</td><td>FC[8]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.MTCEn && IA32_RTIT_CTL.TSCEn && TriggerEn</td><td>Generation Scenario</td><td colspan="2">Sent with any TSC packet.</td></tr>
<tr><td>Description</td><td colspan="4">The TMA packet serves to provide the information needed to allow the decoder to correlate MTC packets with TSC packets. With this packet, when a MTC packet is encountered, the decoder can determine how many timestamp counter ticks have passed since the last TSC or MTC packet. See Section 33.8.3.2 for details on how to make this calculation.</td></tr>
<tr><td>Application</td><td colspan="4">TMA is always sent immediately following a TSC packet, and the payload values are consistent with the TSC payload value. Thus the application of TMA matches that of TSC.</td></tr>
</table>

33.4.2.14 Cycle Count (CYC) Packet

Table 33-33. Cycle Count Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Cycle Count (CYC) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td colspan="5">Cycle Counter[4:0]</td><td>Exp</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td colspan="7">Cycle Counter[11:5]</td><td>Exp</td></tr>
<tr><td>2</td><td colspan="7">Cycle Counter[18:12]</td><td>Exp</td></tr>
<tr><td>...</td><td colspan="8">... (if Exp = 1 in the previous byte)</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.CYCEn && TriggerEn</td><td>Generation Scenario</td><td colspan="2">Can be sent at any time, though a maximum of one CYC packet is sent per core clock cycle. See Section 33.3.6 for CYC-eligible packets.</td></tr>
<tr><td>Description</td><td colspan="4">The Cycle Counter field increments at the same rate as the processor core clock ticks, but with a variable length format (using a trailing EXP bit field) and a range-capped byte length.
If the CYC value is less than 32, a 1-byte CYC will be generated, with Exp=0. If the CYC value is between 32 and 4095 inclusive, a 2-byte CYC will be generated, with byte 0 Exp=1 and byte 1 Exp=0. And so on.
CYC provides the number of core clocks that have passed since the last CYC packet. CYC can be configured to be sent in every cycle in which an eligible packet is generated, or software can opt to use a threshold to limit the number of CYC packets, at the expense of some precision. These settings are configured using the IA32_RTIT_CTL.CycThresh field (see Section 33.2.8.2). For details on Cycle-Accurate Mode, IPC calculation, etc, see Section 33.3.6.
When CycThresh=0, and hence no threshold is in use, then a CYC packet will be generated in any cycle in which any CYC-eligible packet is generated. The CYC packet will precede the other packets generated in the cycle, and provides the precise cycle time of the packets that follow.
In addition to these CYC packets generated with other packets, CYC packets can be sent stand-alone. These packets serve simply to update the decoder with the number of cycles passed, and are used to ensure that a wrap of the processor's internal cycle counter doesn't cause cycle information to be lost. These stand-alone CYC packets do not indicate the cycle time of any other packet or operation, and will be followed by another CYC packet before any other CYC-eligible packet is seen.
When CycThresh>0, CYC packets are generated only after a minimum number of cycles have passed since the last CYC packet. Once this threshold has passed, the behavior above resumes, where CYC will either be sent in the next cycle that produces other CYC-eligible packets, or could be sent stand-alone.
When using CYC thresholds, only the cycle time of the operation (instruction or event) that generates the CYC packet is truly known. Other operations simply have their execution time bounded: they completed at or after the last CYC time, and before the next CYC time.</td></tr>
<tr><td>Application</td><td colspan="4">CYC provides the offset cycle time (since the last CYC packet) for the CYC-eligible packet that follows. If another CYC is encountered before the next CYC-eligible packet, the cycle values should be accumulated and applied to the next CYC-eligible packet.
If a CYC packet is generated by a TNT, note that the cycle time provided by the CYC packet applies to the first branch in the TNT packet.</td></tr>
</table>

33.4.2.15 VMCS Packet

Table 33-34. VMCS Packet Definition

<table>
<tr><td>Name</td><td colspan="4">VMCS Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>2</td><td colspan="8">VMCS pointer [19:12]</td></tr>
<tr><td>3</td><td colspan="8">VMCS pointer [27:20]</td></tr>
<tr><td>4</td><td colspan="8">VMCS pointer [35:28]</td></tr>
<tr><td>5</td><td colspan="8">VMCS pointer [43:36]</td></tr>
<tr><td>6</td><td colspan="8">VMCS pointer [51:44]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn; Also in VMX operation.</td><td>Generation Scenario</td><td colspan="2">Generated on successful VMPTRLD, and optionally on PSB+, SMM VM exits, and VM entries that return from SMM (see Section 33-53).</td></tr>
<tr><td>Description</td><td colspan="4">The VMCS packet provides a VMCS pointer for a decoder to determine the transition of code contexts:
• On a successful VMPTRLD (i.e., a VMPTRLD that doesn't fault, fail, or VM exit), the VMCS packet contains the logical processor's VMCS pointer established by VMPTRLD (for subsequent execution of a VM guest context).
• An SMM VM exit loads the logical processor's VMCS pointer with the SMM-transfer VMCS pointer. If the "conceal VMX from PT" VM-exit control is 0 (see Section 33.5.1), a VMCS packet provides this pointer. See Section 33.6 on tracing inside and outside STM.
• A VM entry that returns from SMM loads the logical processor's VMCS pointer from a field in the SMM-transfer VMCS. If the "conceal VMX from PT" VM-entry control is 0, a VMCS packet provides this pointer. Whether the VM entry is to VMX root operation or VMX non-root operation is indicated by the PIP.NR bit.
A VMCS packet generated before a VMCS pointer has been loaded, or after the VMCS pointer has been cleared will set all 64 bits in the VMCS pointer field.
VMCS packets will not be seen on processors with IA32_VMX_MISC[bit 14]=0, as these processors do not allow TraceEn to be set in VMX operation.</td></tr>
<tr><td>Application</td><td colspan="4">The purpose of the VMCS packet is to help the decoder uniquely identify changes in the executing software context in situations that CR3 may not be unique.
When a VMCS packet is encountered, a decoder should do the following:
▪ If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this VMCS is part of a compound packet event (Section 33.4.1). Find the ending TIP and apply the new VMCS base pointer value to the TIP payload IP.
▪ Otherwise, look for the next VMPTRLD, VMRESUME, or VMLAUNCH in the disassembly, and apply the new VMCS base pointer on the next VM entry.
For examples of the packets generated by these flows, see Section 33.7.</td></tr>
</table>

33.4.2.16 Overflow (OVF) Packet

Table 33-35. OVF Packet Definition

<table>
<tr><td>Name</td><td colspan="3">Overflow (OVF) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3"><table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td>On resolution of internal buffer overflow</td></tr>
<tr><td>Description</td><td colspan="3">OVF simply indicates to the decoder that an internal buffer overflow occurred, and packets were likely lost. If BranchEN= 1, OVF is followed by a FUP or TIP.PGE which will provide the IP at which packet generation resumes. See Section 33.3.8.</td></tr>
<tr><td>Application</td><td colspan="3">When an OVF packet is encountered, the decoder should skip to the IP given in the subsequent FUP or TIP.PGE. The cycle counter for the CYC packet will be reset at the time the OVF packet is sent.
Software should reset its call stack depth on overflow, since no RET compression is allowed across an overflow. Similarly, any IP compression that follows the OVF is guaranteed to use as a reference LastIP the IP payload of an IP packet that preceded the overflow.</td></tr>
</table>

33.4.2.17 Packet Stream Boundary (PSB) Packet

Table 33-36. PSB Packet Definition

<table>
<tr><td>Name</td><td>Packet Stream Boundary (PSB) Packet</td></tr>
<tr><td>Packet Format</td><td><table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>3</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>5</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>7</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>8</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>9</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>10</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>11</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>12</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>13</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>14</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>15</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
</table></td></tr>
</table>

Table 33-36. PSB Packet Definition (Contd.)

Dependencies	TriggerEn	Generation Scenario	Periodic, based on the number of output bytes generated while tracing. PSB is sent when IA32_RTIT_STATUS.PacketByteCnt=0, and each time it crosses the software selected threshold after that. May be sent for other micro-architectural conditions as well.
Description	PSB is a unique pattern in the packet output log, and hence serves as a sync point for the decoder. It is a pattern that the decoder can search for in order to get aligned on packet boundaries. This packet is periodic, based on the number of output bytes, as indicated by IA32_RTIT_STATUS.PacketByteCnt. The period is chosen by software, via IA32_RTIT_CTL.PSBFreq (see Section 33.2.8.2). Note, however, that the PSB period is not precise, it simply reflects the average number of output bytes that should pass between PSBs. The processor will make a best effort to insert PSB as quickly after the selected threshold is reached as possible. The processor also may send extra PSB packets for some micro-architectural conditions. PSB also serves as the leading packet for a set of "status-only" packets collectively known as PSB+ (Section 33.3.7).		
Application	When a PSB is seen, the decoder should interpret all following packets as "status only", until either a PSBEND or OVF packet is encountered. "Status only" implies that the binding and ordering rules to which these packets normally adhere are ignored, and the state they carry can instead be applied to the IP payload in the FUP packet that is included.		

33.4.2.18 PSBEND Packet

Table 33-37. PSBEND Packet Definition

Name	PSBEND Packet		
Packet Format			

	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	1	1

Dependencies	TriggerEn	Generation Scenario	Always follows PSB packet, separated by PSB+ packets
Description	PSBEND is simply a terminator for the series of "status only" (PSB+) packets that follow PSB (Section 33.3.7).		
Application	When a PSBEND packet is seen, the decoder should cease to treat packets as "status only".		

33.4.2.19 Maintenance (MNT) Packet

Table 33-38. MNT Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Maintenance (MNT) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>3</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>4</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>5</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>6</td><td colspan="8">Payload[31:24]</td></tr>
<tr><td>7</td><td colspan="8">Payload[39:32]</td></tr>
<tr><td>8</td><td colspan="8">Payload[47:40]</td></tr>
<tr><td>9</td><td colspan="8">Payload[55:48]</td></tr>
<tr><td>10</td><td colspan="8">Payload[63:56]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">Implementation specific.</td></tr>
<tr><td>Description</td><td colspan="4">This packet is generated by hardware, the payload meaning is model-specific.</td></tr>
<tr><td>Application</td><td colspan="4">Unless a decoder has been extended for a particular family/model/stepping to interpret MNT packet payloads, this packet should simply be ignored. It does not bind to any IP.</td></tr>
</table>

33.4.2.20 PAD Packet

Table 33-39. PAD Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PAD Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">Implementation specific</td></tr>
<tr><td>Description</td><td colspan="4">PAD is simply a NOP packet. Processor implementations may choose to add pad packets to improve packet alignment or for implementation-specific reasons.</td></tr>
<tr><td>Application</td><td colspan="4">Ignore PAD packets.</td></tr>
</table>

33.4.2.21 PTWRITE (PTW) Packet

Table 33-40. PTW Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PTW Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>IP</td><td colspan="2">PayloadBytes</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>3</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>4</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>5</td><td colspan="8">Payload[31:24]</td></tr>
<tr><td>6</td><td colspan="8">Payload[39:32]</td></tr>
<tr><td>7</td><td colspan="8">Payload[47:40]</td></tr>
<tr><td>8</td><td colspan="8">Payload[55:48]</td></tr>
<tr><td>9</td><td colspan="8">Payload[63:56]</td></tr>
</table>
</td></tr>
<tr><td></td><td colspan="4">The PayloadBytes field indicates the number of bytes of payload that follow the header bytes. Encodings are as follows:
<table>
<tr><th>PayloadBytes</th><th>Bytes of Payload</th></tr>
<tr><td>‘00</td><td>4</td></tr>
<tr><td>‘01</td><td>8</td></tr>
<tr><td>‘10</td><td>Reserved</td></tr>
<tr><td>‘11</td><td>Reserved</td></tr>
</table>
IP bit indicates if a FUP, whose payload will be the IP of the PTWRITE instruction, will follow.</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn && FilterEn && PTWEn</td><td>Generation Scenario</td><td colspan="2">PTWRITE Instruction</td></tr>
<tr><td>Description</td><td colspan="4">Contains the value held in the PTWRITE operand.
This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.</td></tr>
<tr><td>Application</td><td colspan="4">Binds to the associated PTWRITE instruction. The IP of the PTWRITE will be provided by a following FUP, when PTW.IP=1.</td></tr>
</table>

33.4.2.22 Execution Stop (EXSTOP) Packet

Table 33-41. EXSTOP Packet Definition

<table>
<tr><td>Name</td><td colspan="4">EXSTOP Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>IP</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
</table>
IP bit indicates if a FUP will follow.
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn</td><td>Generation Scenario</td><td colspan="2">C-state entry, P-state change, or other processor clock power-down. Includes :
▪ Entry to C-state deeper than C0.0
▪ TM1/2
▪ STPCLK#
▪ Frequency change due to IA32_CLOCK_MODULATION, Turbo</td></tr>
<tr><td>Description</td><td colspan="4">This packet indicates that software execution has stopped due to processor clock powerdown. Later packets will indicate when execution resumes.
If EXSTOP is generated while ContextEn is set, the IP bit will be set, and EXSTOP will be followed by a FUP packet containing the IP at which execution stopped. More precisely, this will be the IP of the oldest instruction that has not yet completed.
This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.</td></tr>
<tr><td>Application</td><td colspan="4">If a FUP follows EXSTOP (hence IP bit set), the EXSTOP can be bound to the FUP IP. Otherwise the IP is not known. Time of powerdown can be inferred from the preceding CYC, if CYCEn=1. Combined with the TSC at the time of wake (if TSCEn=1), this can be used to determine the duration of the powerdown.</td></tr>
</table>

33.4.2.23 MWAIT Packet

Table 33-42. MWAIT Packet Definition

<table>
<tr><td>Name</td><td colspan="5">MWAIT Packet</td></tr>
<tr><td>Packet Format</td><td colspan="5">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td colspan="8">MWAIT Hints[7:0]</td></tr>
<tr><td>3</td><td colspan="8">Reserved</td></tr>
<tr><td>4</td><td colspan="8">Reserved</td></tr>
<tr><td>5</td><td colspan="8">Reserved</td></tr>
<tr><td>6</td><td colspan="6">Reserved</td><td colspan="2">EXT[1:0]</td></tr>
<tr><td>7</td><td colspan="8">Reserved</td></tr>
<tr><td>8</td><td colspan="8">Reserved</td></tr>
<tr><td>9</td><td colspan="8">Reserved</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn && ContextEn</td><td>Generation Scenario</td><td colspan="3">MWAIT, UMWAIT, or TPAUSE instructions, or I/O redirection to MWAIT, that complete without fault or VMexit.</td></tr>
<tr><td>Description</td><td colspan="5">Indicates that an MWAIT operation to C-state deeper than C0.0 completed. The MWAIT hints and extensions passed in by software are exposed in the payload. For UMWAIT and TPAUSE, the EXT field holds the input register value that determines the optimized state requested.
For entry to some highly optimized C0 sub-C-states, such as C0.1, no MWAIT packet is generated.
This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.</td></tr>
<tr><td>Application</td><td colspan="5">The binding for the upcoming EXSTOP packet also applies to the MWAIT packet. See Section 33.4.2.22.</td></tr>
</table>

33.4.2.24 Power Entry (PWRE) Packet

Table 33-43. PWRE Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PWRE Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>HW</td><td colspan="7">Reserved</td></tr>
<tr><td>3</td><td colspan="4">Resolved Thread C-State</td><td colspan="4">Resolved Thread Sub C-State</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn</td><td>Generation Scenario</td><td colspan="2">Transition to a C-state deeper than C0.0.</td></tr>
<tr><td>Description</td><td colspan="4">Indicates processor entry to the resolved thread C-state and sub C-state indicated. The processor will remain in this C-state until either another PWRE indicates the processor has moved to a C-state deeper than C0.0, or a PWRX packet indicates a return to C0.0.
For entry to some highly optimized C0 sub-C-states, such as C0.1, no PWRE packet is generated.
Note that some CPUs may allow MWAIT to request a deeper C-state than is supported by the core. These deeper C-states may have platform-level implications that differentiate them. However, the PWRE packet will provide only the resolved thread C-state, which will not exceed that supported by the core.
If the C-state entry was initiated by hardware, rather than a direct software request (such as MWAIT, UMWAIT, TPAUSE, HLT, or shutdown), the HW bit will be set to indicate this. Hardware Duty Cycling (see Section 15.5, "Hardware Duty Cycling (HDC)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B) is an example of such a case.</td></tr>
<tr><td>Application</td><td colspan="4">When transitioning from C0.0 to a deeper C-state, the PWRE packet will be followed by an EXSTOP. If that EXSTOP packet has the IP bit set, then the following FUP will provide the IP at which the C-state entry occurred. Subsequent PWRE packets generated before the next PWRX should bind to the same IP.</td></tr>
</table>

33.4.2.25 Power Exit (PWRX) Packet

Table 33-44. PWRX Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PWRX Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td colspan="4">Last Core C-State</td><td colspan="4">Deepest Core C-State</td></tr>
<tr><td>3</td><td colspan="4">Reserved</td><td colspan="4">Wake Reason</td></tr>
<tr><td>4</td><td colspan="8">Reserved</td></tr>
<tr><td>5</td><td colspan="8">Reserved</td></tr>
<tr><td>6</td><td colspan="8">Reserved</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn</td><td>Generation Scenario</td><td colspan="2">Transition from a C-state deeper than C0.0 to C0.</td></tr>
<tr><td>Description</td><td colspan="4">Indicates processor return to thread C0 from a C-state deeper than C0.0.
For return from some highly optimized C0 sub-C-states, such as C0.1, no PWRX packet is generated.
The Last Core C-State field provides the MWAIT encoding for the core C-state at the time of the wake. The Deepest Core C-State provides the MWAIT encoding for the deepest core C-state achieved during the sleep session, or since leaving thread C0. MWAIT encodings for C-states can be found in Table 4-11 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B. Note that these values reflect only the core C-state, and hence will not exceed the maximum supported core C-state, even if deeper C-states can be requested.
The Wake Reason field is one-hot, encoded as follows:
<table>
<tr><th>Bit</th><th>Field</th><th>Meaning</th></tr>
<tr><td>0</td><td>Interrupt</td><td>Wake due to external interrupt received.</td></tr>
<tr><td>1</td><td>Timer Deadline</td><td>Wake due to timer expiration, such as UMWAIT/TPAUSE TSC-quanta.</td></tr>
<tr><td>2</td><td>Store to Monitored Address</td><td>Wake due to store to monitored address.</td></tr>
<tr><td>3</td><td>HW Wake</td><td>Wake due to hardware autonomous condition, such as HDC.</td></tr>
</table>
</td></tr>
<tr><td>Application</td><td colspan="4">PWRX will always apply to the same IP as the PWRE. The time of wake can be discerned from (optional) timing packets that precede PWRX.</td></tr>
</table>

33.4.2.26 Block Begin Packet (BBP)

Table 33-45. Block Begin Packet Definition

<table>
<tr><td>Name</td><td colspan="4">BBP</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>SZ</td><td colspan="3">Reserved</td><td colspan="4">Type[4:0]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">PEBS event, if IA32_PEBS_ENABLE.OUTPUT=1.</td></tr>
<tr><td>Description</td><td colspan="4">This packet indicates the beginning of a block of packets which are collectively tied to a single event or instruction. The size of the block item payloads within this block is provided by the Size (SZ) bit:
SZ=0: 8-byte block items
SZ=1: 4-byte block items
The meaning of the BIP payloads is provided by the Type field:
<table>
<tr><th>BBP.Type</th><th>Block name</th></tr>
<tr><td>0x00</td><td>Reserved</td></tr>
<tr><td>0x01</td><td>General-Purpose Registers</td></tr>
<tr><td>0x02..0x03</td><td>Reserved</td></tr>
<tr><td>0x04</td><td>PEBS Basic</td></tr>
<tr><td>0x05</td><td>PEBS Memory</td></tr>
<tr><td>0x06..0x07</td><td>Reserved</td></tr>
<tr><td>0x08</td><td>LBR Block 0</td></tr>
<tr><td>0x09</td><td>LBR Block 1</td></tr>
<tr><td>0x0A</td><td>LBR Block 2</td></tr>
<tr><td>0x0B..0x0F</td><td>Reserved</td></tr>
<tr><td>0x10</td><td>XMM Registers</td></tr>
<tr><td>0x11..0x1F</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>Application</td><td colspan="4">A BBP will always be followed by a Block End Packet (BEP), and when the block is generated while ContextEn=1 that BEP will have IP=1 and be followed by a FUP that provides the IP to which the block should be bound. Note that, in addition to BEP, a block can be terminated by a BBP (indicating the start of a new block) or an OVF packet.</td></tr>
</table>

33.4.2.27 Block Item Packet (BIP)

Table 33-46. Block Item Packet Definition

<table>
<tr><td>Name</td><td colspan="4">BIP</td></tr>
<tr><td>Packet Format</td><td colspan="4">If the preceding BBP.SZ=0:
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td colspan="5">ID[5:0]</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>2</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>3</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>4</td><td colspan="8">Payload[31:24]</td></tr>
<tr><td>5</td><td colspan="8">Payload[39:32]</td></tr>
<tr><td>6</td><td colspan="8">Payload[47:40]</td></tr>
<tr><td>7</td><td colspan="8">Payload[55:48]</td></tr>
<tr><td>8</td><td colspan="8">Payload[63:56]</td></tr>
</table>
If the preceding BBP.SZ=1:
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td colspan="5">ID[5:0]</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>2</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>3</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>4</td><td colspan="8">Payload[31:24]</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">See BBP.</td></tr>
<tr><td>Description</td><td colspan="4">The size of the BIP payload is determined by the Size field in the preceding BBP packet.
The BIP header provides the ID value that, when combined with the Type field from the preceding BBP, uniquely identifies the state value held in the BIP payload. See Table 33-47 below for the complete list.</td></tr>
<tr><td>Application</td><td colspan="4">See BBP.</td></tr>
</table>

BIP State Value Encodings

The table below provides the encoding values for all defined block items. State items that are larger than 8 bytes, such as XMM register values, are broken into multiple 8-byte components. BIP packets with Size=1 (4 byte payload) will provide only the lower 4 bytes of the associated state value.

Table 33-47. BIP Encodings

BBP.Type	BIP.ID	State Value
General-Purpose Registers		
0x01	0x00	R/EFLAGS
0x01	0x01	R/EIP
0x01	0x02	R/EAX
0x01	0x03	R/ECX

Table 33-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x01	0x04	R/EDX
0x01	0x05	R/EBX
0x01	0x06	R/ESP
0x01	0x07	R/EBP
0x01	0x08	R/ESI
0x01	0x09	R/EDI
0x01	0x0A	R8
0x01	0x0B	R9
0x01	0x0C	R10
0x01	0x0D	R11
0x01	0x0E	R12
0x01	0x0F	R13
0x01	0x10	R14
0x01	0x11	R15
PEBS Basic Info (Section 20.9.2.2.1)		
0x04	0x00	Instruction Pointer
0x04	0x01	Applicable Counters
0x04	0x02	Timestamp
PEBS Memory Info (Section 20.9.2.2.2)		
0x05	0x00	MemAccessAddress
0x05	0x01	MemAuxInfo
0x05	0x02	MemAccessLatency
0x05	0x03	TSXAuxInfo
LBR_0		
0x08	0x00	LBR[TOS-0]_FROM_IP
0x08	0x01	LBR[TOS-0]_TO_IP
0x08	0x02	LBR[TOS-0]_INFO
0x08	0x03	LBR[TOS-1]_FROM_IP
0x08	0x04	LBR[TOS-1]_TO_IP
0x08	0x05	LBR[TOS-1]_INFO
0x08	0x06	LBR[TOS-2]_FROM_IP
0x08	0x07	LBR[TOS-2]_TO_IP
0x08	0x08	LBR[TOS-2]_INFO
0x08	0x09	LBR[TOS-3]_FROM_IP
0x08	0x0A	LBR[TOS-3]_TO_IP
0x08	0x0B	LBR[TOS-3]_INFO
0x08	0x0C	LBR[TOS-4]_FROM_IP
0x08	0x0D	LBR[TOS-4]_TO_IP

Table 33-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x08	0x0E	LBR[TOS-4]_INFO
0x08	0x0F	LBR[TOS-5]_FROM_IP
0x08	0x10	LBR[TOS-5]_TO_IP
0x08	0x11	LBR[TOS-5]_INFO
0x08	0x12	LBR[TOS-6]_FROM_IP
0x08	0x13	LBR[TOS-6]_TO_IP
0x08	0x14	LBR[TOS-6]_INFO
0x08	0x15	LBR[TOS-7]_FROM_IP
0x08	0x16	LBR[TOS-7]_TO_IP
0x08	0x17	LBR[TOS-7]_INFO
0x08	0x18	LBR[TOS-8]_FROM_IP
0x08	0x19	LBR[TOS-8]_TO_IP
0x08	0x1A	LBR[TOS-8]_INFO
0x08	0x1B	LBR[TOS-9]_FROM_IP
0x08	0x1C	LBR[TOS-9]_TO_IP
0x08	0x1D	LBR[TOS-9]_INFO
0x08	0x1E	LBR[TOS-10]_FROM_IP
0x08	0x1F	LBR[TOS-10]_TO_IP
LBR_1		
0x09	0x00	LBR[TOS-10]_INFO
0x09	0x01	LBR[TOS-11]_FROM_IP
0x09	0x02	LBR[TOS-11]_TO_IP
0x09	0x03	LBR[TOS-11]_INFO
0x09	0x04	LBR[TOS-12]_FROM_IP
0x09	0x05	LBR[TOS-12]_TO_IP
0x09	0x06	LBR[TOS-12]_INFO
0x09	0x07	LBR[TOS-13]_FROM_IP
0x09	0x08	LBR[TOS-13]_TO_IP
0x09	0x09	LBR[TOS-13]_INFO
0x09	0x0A	LBR[TOS-14]_FROM_IP
0x09	0x0B	LBR[TOS-14]_TO_IP
0x09	0x0C	LBR[TOS-14]_INFO
0x09	0x0D	LBR[TOS-15]_FROM_IP
0x09	0x0E	LBR[TOS-15]_TO_IP
0x09	0x0F	LBR[TOS-15]_INFO
0x09	0x10	LBR[TOS-16]_FROM_IP
0x09	0x11	LBR[TOS-16]_TO_IP
0x09	0x12	LBR[TOS-16]_INFO

Table 33-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x09	0x13	LBR[TOS-17]_FROM_IP
0x09	0x14	LBR[TOS-17]_TO_IP
0x09	0x15	LBR[TOS-17]_INFO
0x09	0x16	LBR[TOS-18]_FROM_IP
0x09	0x17	LBR[TOS-18]_TO_IP
0x09	0x18	LBR[TOS-18]_INFO
0x09	0x19	LBR[TOS-19]_FROM_IP
0x09	0x1A	LBR[TOS-19]_TO_IP
0x09	0x1B	LBR[TOS-19]_INFO
0x09	0x1C	LBR[TOS-20]_FROM_IP
0x09	0x1D	LBR[TOS-20]_TO_IP
0x09	0x1E	LBR[TOS-20]_INFO
0x09	0x1F	LBR[TOS-21]_FROM_IP
LBR_2		
0x0A	0x00	LBR[TOS-21]_TO_IP
0x0A	0x01	LBR[TOS-21]_INFO
0x0A	0x02	LBR[TOS-22]_FROM_IP
0x0A	0x03	LBR[TOS-22]_TO_IP
0x0A	0x04	LBR[TOS-22]_INFO
0x0A	0x05	LBR[TOS-23]_FROM_IP
0x0A	0x06	LBR[TOS-23]_TO_IP
0x0A	0x07	LBR[TOS-23]_INFO
0x0A	0x08	LBR[TOS-24]_FROM_IP
0x0A	0x09	LBR[TOS-24]_TO_IP
0x0A	0x0A	LBR[TOS-24]_INFO
0x0A	0x0B	LBR[TOS-25]_FROM_IP
0x0A	0x0C	LBR[TOS-25]_TO_IP
0x0A	0x0D	LBR[TOS-25]_INFO
0x0A	0x0E	LBR[TOS-26]_FROM_IP
0x0A	0x0F	LBR[TOS-26]_TO_IP
0x0A	0x10	LBR[TOS-26]_INFO
0x0A	0x11	LBR[TOS-27]_FROM_IP
0x0A	0x12	LBR[TOS-27]_TO_IP
0x0A	0x13	LBR[TOS-27]_INFO
0x0A	0x14	LBR[TOS-28]_FROM_IP
0x0A	0x15	LBR[TOS-28]_TO_IP
0x0A	0x16	LBR[TOS-28]_INFO
0x0A	0x17	LBR[TOS-29]_FROM_IP

Table 33-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x0A	0x18	LBR[TOS-29]_TO_IP
0x0A	0x19	LBR[TOS-29]_INFO
0x0A	0x1A	LBR[TOS-30]_FROM_IP
0x0A	0x1B	LBR[TOS-30]_TO_IP
0x0A	0x1C	LBR[TOS-30]_INFO
0x0A	0x1D	LBR[TOS-31]_FROM_IP
0x0A	0x1E	LBR[TOS-31]_TO_IP
0x0A	0x1F	LBR[TOS-31]_INFO
XMM Registers		
0x10	0x00	XMM0_Q0
0x10	0x01	XMM0_Q1
0x10	0x02	XMM1_Q0
0x10	0x03	XMM1_Q1
0x10	0x04	XMM2_Q0
0x10	0x05	XMM2_Q1
0x10	0x06	XMM3_Q0
0x10	0x07	XMM3_Q1
0x10	0x08	XMM4_Q0
0x10	0x09	XMM4_Q1
0x10	0x0A	XMM5_Q0
0x10	0x0B	XMM5_Q1
0x10	0x0C	XMM6_Q0
0x10	0x0D	XMM6_Q1
0x10	0x0E	XMM7_Q0
0x10	0x0F	XMM7_Q1
0x10	0x10	XMM8_Q0
0x10	0x11	XMM8_Q1
0x10	0x12	XMM9_Q0
0x10	0x13	XMM9_Q1
0x10	0x14	XMM10_Q0
0x10	0x15	XMM10_Q1
0x10	0x16	XMM11_Q0
0x10	0x17	XMM11_Q1
0x10	0x18	XMM12_Q0
0x10	0x19	XMM12_Q1
0x10	0x1A	XMM13_Q0
0x10	0x1B	XMM13_Q1
0x10	0x1C	XMM14_Q0

Table 33-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x10	0x1D	XMM14_Q1
0x10	0x1E	XMM15_Q0
0x10	0x1F	XMM15_Q1

33.4.2.28 Block End Packet (BEP)

Table 33-48. Block End Packet Definition

<table>
<tr><td>Name</td><td colspan="4">BEP</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>IP</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">See BBP.</td></tr>
<tr><td>Description</td><td colspan="4">Indicates the end of a packet block. The IP bit indicates if a FUP will follow, and will be set if ContextEn=1.</td></tr>
<tr><td>Application</td><td colspan="4">The block, from initial BBP to the BEP, binds to the FUP IP, if IP=1, and consumes the FUP.</td></tr>
</table>

33.4.2.29 Control Flow Event (CFE) Packet

Table 33-49. Control Flow Event Packet Definition

<table>
<tr><td>Name</td><td colspan="4">CFE</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>IP</td><td colspan="2">Reserved</td><td colspan="4">Type[4:0]</td></tr>
<tr><td>3</td><td colspan="8">Vector[7:0]</td></tr>
</table>
IP bit indicates if a stand-alone FUP will follow.
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.EventEn && TriggerEn && ContextEn
On ContextEn transitions, the CFE will be generated regardless of direction (1⟶0 or 0⟶1). VM exit is an exception, where CFE.VMEXIT depends only on the prior value of ContextEn.</td><td>Generation Scenario</td><td colspan="2">Software interrupt, external interrupt, user interrupt, or exception, including those injected on VM entry.
INIT, SIPI, SMI, RSM, IRET, Shutdown.
VM exit, if “Conceal VMX in PT” VMCS exit control is 0.
VM entry, if “Conceal VMX in PT” VMCS entry control is 0.
TSX Abort.</td></tr>
<tr><td>Description</td><td colspan="4">This packet indicates that an asynchronous event or related event (see list above) has occurred. The type of event is provided in the packet (see Table 33-50 below), and, if the IP bit is set, the IP at which the event occurred is provided in a stand-alone FUP packet that follows. Further, in the case of an interrupt or exception, the vector field provides the vector of the event.
The IP bit will be set only when ContextEn=1 before the event is taken, and either BranchEn=0 or else no FUP is generated for this event by BranchEn=1. There are some cases, such as SIPI and RSM, where no FUP is generated. Note that events that are not delivered to software, such as nested events or events which cause a VM exit, do not generate CFE packets.</td></tr>
<tr><td>Application</td><td colspan="4">If the IP bit is set, a FUP will follow that is stand-alone (not part of a compound packet event), and the CFE consumes the FUP. If the IP bit is not set, the CFE binds to the next FUP if PacketEn=1 (hence the CFE comes after a TIP.PGE but before the next TIP.PGD), and is stand-alone if PacketEn=0.</td></tr>
</table>

CFE Packet Type and Vector Fields

Every CFE has a Type field, which provides the type of event which generated the packet. For a subset of CFE Types, the CFE.Vector field may be valid. Details on these fields, as well as the IP to be expected in any following FUP packet, are provided in the table below.

Table 33-50. CFE Packet Type and Vector Fields Details

CFE Subtype	Type	Vector	FUP IP	Details
INTR	0x1	Event Vector	Varies	Used for interrupts (external and software), Exceptions, Faults, and NMI. FUP contains that address of the instruction that has not completed (NLIP for trap events, CLIP for fault events).
IRET	0x2	Invalid	CLIP	
SMI	0x3	Invalid	NLIP	
RSM	0x4	Invalid	None	
SIPI	0x5	SIPI Vector	None	
INIT	0x6	Invalid	NLIP	
VMENTRY	0x7	Invalid	CLIP	FUP contains IP of VMLAUNCH/VMRESUME.

Table 33-50. CFE Packet Type and Vector Fields Details (Contd.)

CFE Subtype	Type	Vector	FUP IP	Details
VMEXIT	0x8	Invalid	Varies	FUP IP varies depending on type of VM exit, but will be the address of the instruction that has not completed. Will be consistent with Guest IP saved in VMCS.
VMEXIT_INTR	0x9	Event Vector	Varies	Sent in cases where VM exit was caused by an INTR event (interrupt, exception, fault, or NMI). Vector provided is for the event which caused the VM exit. FUP IP behavior matches that of INTR type above.
SHUTDOWN	0xa	Invalid	Varies	FUP IP varies depending on the type of event that caused shutdown, but will be the address of the instruction that has not completed.
Reserved	0xb	N/A	N/A	
UINTR	0xc	User Interrupt Vector	NLIP	User interrupt delivered.
UIRET	0xd	Invalid	CLIP	Exiting from user interrupt routine.
Reserved	0xe...0x1f	N/A	N/A	Reserved

33.4.2.30 Event Data (EVD) Packet

Table 33-51. Event Data Packet Definition

<table>
<tr><td>Name</td><td colspan="4">EVD</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td colspan="2">Reserved</td><td colspan="6">Type[5:0]</td></tr>
<tr><td>3</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>4</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>5</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>6</td><td colspan="8">Payload[31:24]</td></tr>
<tr><td>7</td><td colspan="8">Payload[39:32]</td></tr>
<tr><td>8</td><td colspan="8">Payload[47:40]</td></tr>
<tr><td>9</td><td colspan="8">Payload[55:48]</td></tr>
<tr><td>10</td><td colspan="8">Payload[63:56]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.EventEn && TriggerEn && ContextEn</td><td>Generation Scenario</td><td colspan="2">Page fault, including those injected on VM entry.
VM exit, if “Suppress VMX packets on exit” VMCS exit control is 0.</td></tr>
<tr><td>Description</td><td colspan="4">Provides additional data about the event that caused the following CFE. The Payload field is dictated by the Type.
<table>
<tr><th>Type</th><th>Payload</th></tr>
<tr><td>‘000000</td><td>Page Fault Linear Address, same as CR2 (PFA)</td></tr>
<tr><td>‘000001</td><td>VMX Exit Qualification (VMXQ)</td></tr>
<tr><td>‘000010</td><td>VMX Exit Reason (VMXR)</td></tr>
<tr><td>‘000011 - ‘111111</td><td>Reserved</td></tr>
</table>
EVD packets are never generated in cases where a CFE is not.</td></tr>
<tr><td>Application</td><td colspan="4">EVD packets bind to the same IP (if any) as the subsequent CFE packet.</td></tr>
</table>

33.5 TRACING IN VMX OPERATION

On processors that IA32_VMX_MISC[bit 14] reports 1, TraceEn can be set in VMX operation. The VMM can configure specific VMX controls to control what virtualization-specific data is included within the trace packets (see Section 33.5.1 for details). The VMM can also configure the VMCS to limit tracing to non-root operation, or to trace across both root and non-root operation. The VMCS controls exist to simplify virtualization of Intel PT for guest use, including the “Clear IA32_RTIT_CTL” exit control (See Section 25.7.1), “Load IA32_RTIT_CTL” entry control (See Section 25.8.1), and “Intel PT uses guest physical addresses” execution control (See Section 26.5.3).

For older processors that do not support these VMCS controls, the MSR-load areas used by VMX transitions can be employed by the VMM to restrict tracing to the desired context. See Section 33.5.2 for details. Tracing with SMM Transfer Monitor is described in Section 33.6.

33.5.1 VMX-Specific Packets and VMCS Controls

In all of the usages of VMX and Intel PT, a decoder in the host or VMM context can identify the occurrences of VMX transitions with the aid of VMX-specific packets. There are four kinds of packets relevant to VMX:

- **VMCS packet.** The VMX transitions of individual VMs can be distinguished by a decoder using the VMCS-pointer field in a VMCS packet. A VMCS packet is sent on a successful execution of VMPTRLD, and its VMCS-pointer field stores the VMCS pointer loaded by that execution. See Section 33.4.2.15 for details.
- **The NR (non-root) bit in a PIP packet.** Normally, the NR bit is set in any PIP packet generated in VMX non-root operation. In addition, PIP packets are generated with each VM entry and VM exit. Thus a transition of the NR bit from 0 to 1 indicates the occurrence of a VM entry, and a transition of 1 to 0 indicates the occurrence of a VM exit.
- **CFE packet.** Identifies VM exit and VM entry operations.
- **EVD packet.** Provides the exit reason and exit qualification for VM exits.

There are VMX controls that a VMM can set to conceal some of this VMX-specific information (by suppressing its recording) and thereby prevent it from leaking across virtualization boundaries. There is one of these controls (each of which is called "conceal VMX from PT") of each type of VMX control.

Table 33-52. VMX Controls For Intel Processor Trace

Type of VMX Control	Bit Position[1]	Value	Behavior
Secondary processor-based VM-execution control	19	0	Each PIP generated in VM non-root operation will set the NR bit. PSB+ in VMX non-root operation will include the VMCS packet, to ensure that the decoder knows which guest is currently in use.
		1	Each PIP generated in VMX non-root operation will clear the NR bit. PSB+ in VMX non-root operation will not include the VMCS packet.
VM-exit control	24	0	Each VM exit generates a PIP in which the NR bit is clear, and a CFE/EVD if Event Trace is enabled. In addition, SMM VM exits generate VMCS packets.
		1	VM exits do not generate PIPs, CFEs, or EVDs, and no VMCS packets are generated on SMM VM exits.
VM-entry control	17	0	Each VM entry generates a PIP in which the NR bit is set (except VM entries that return from SMM to VMX root operation), and a CFE if Event Trace is enabled. In addition, VM entries that return from SMM generate VMCS packets.
		1	VM entries do not generate PIPs or CFEs, and no VMCS packets are generated on VM entries that return from SMM.

NOTES:

1. These are the positions of the control bits in the relevant VMX control fields.

The 0-settings of these VMX controls enable all VMX-specific packet information. The scenarios that would use these default settings also do not require the VMM to use VMX MSR-load areas to enable and disable trace-packet generation across VMX transitions.

If IA32_VMX_MISC[bit 14] reports 0, the 1-settings of the VMX controls in Table 33-52 are not supported, and VM entry will fail on any attempt to set them.

33.5.2 Managing Trace Packet Generation Across VMX Transitions

In tracing scenarios that collect packets for both VMX root operation and VMX non-root operation, a host executive can manage the MSRs associated with trace packet generation directly. The states of these MSRs need not be modified across VMX transitions.

For tracing scenarios that collect packets only within VMX root operation or only within VMX non-root operation, the VMM can toggle IA32_RTIT_CTL.TraceEn on VMX transitions.

33.5.2.1 System-Wide Tracing

When a host or VMM configures Intel PT to collect trace packets of the entire system, it can leave the relevant VMX controls clear to allow VMX-specific packets to provide information across VMX transitions.

The decoder will desire to identify the occurrence of VMX transitions. The packets of interests to a decoder are shown in Table 33-53.

Table 33-53. Packets on VMX Transitions (System-Wide Tracing)

Event	Packets	Enable	Description
VM exit	EVD.VMXR, EVD.VMXQ, CFE.VMEXIT*	EventEn	The CFE identifies the transfer as a VM exit, while the associated EVDs provide the exit reason and exit qualification.
	FUP(GuestIP)	BranchEn or EventEn	The FUP indicates at which point in the guest flow the VM exit occurred. This is important, since VM exit can be an asynchronous event. The IP will match that written into the VMCS.
	PIP(HostCR3, NR=0)		The PIP packet provides the new host CR3 value, as well as indication that the logical processor is entering VMX root operation. This allows the decoder to identify the change of executing context from guest to host and load the appropriate set of binaries to continue decode.
	TIP(HostIP)	BranchEn	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX root operation. Note, this packet could be preceded by a MODE.Exec packet (Section 33.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.
VM entry	CFEVMENTRY, FUP(CLIP)	EventEn	The CFE identifies the transfer as a VM entry, while the FUP identifies the VMLAUNCH/VMRESUME IP.
	PIP(GuestCR3, NR=1)	BranchEn	The PIP packet provides the new guest CR3 value, as well as indication that the logical processor is entering VMX non-root operation. This allows the decoder to identify the change of executing context from host to guest and load the appropriate set of binaries to continue decode.
	TIP(GuestIP)	BranchEn	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX non-root operation. This should match the RIP loaded from the VMCS. Note, this packet could be preceded by a MODE.Exec packet (Section 33.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.

Since the VMX controls that suppress packet generation are cleared, a VMCS packet will be included in all PSB+ for this usage scenario. Additionally, VMPTRLD will generate such a packet. Thus the decoder can distinguish the execution context of different VMs.

When the host VMM configures a system to collect trace packets in this scenario, it should emulate CPUID to report CPUID.(EAX=07H, ECX=0):EBX[bit 26] as 0 to guests, indicating to guests that Intel PT is not available.

VMX TSC Manipulation

The TSC packets generated while in VMX non-root operation will include any changes resulting from the use of a VMM's use of the TSC offsetting or TSC scaling VMX controls (see Chapter 26, "VMX Non-Root Operation"). In this system-wide usage model, the decoder may need to account for the effect of per-VM adjustments in the TSC packets generated in VMX non-root operation and the absence of TSC adjustments in TSC packets generated in VMX root operation. The VMM can supply this information to the decoder.

33.5.2.2 Guest-Only Tracing

A VMM can configure trace-packet generation while in VMX non-root operation for guests executing normally. This is accomplished by utilizing VMCS controls to manipulate the guest IA32_RTIT_CTL value on VMX transitions. For

older processors that do not support these VMCS controls, a VMM can use the VMX MSR-load areas on VM exits (see Section 25.7.2, "VM-Exit Controls for MSRs") and VM entries (see Section 25.8.2, "VM-Entry Controls for MSRs") to limit trace-packet generation to the guest environment.

For this usage, VM entry is programmed to enable trace packet generation, while VM exit is programmed to clear IA32_RTIT_CTL.TraceEn so as to disable trace-packet generation in the host. Further, if it is preferred that the guest packet stream contain no indication that execution was in VMX non-root operation, the VMM should set to 1 all the VMX controls enumerated in Table 33-52.

33.5.2.3 Emulation of Intel PT Traced State

If a VMM emulates an element of processor state by taking a VM exit on reads and/or writes to that piece of state, and the state element impacts Intel PT packet generation or values, it may be incumbent upon the VMM to insert or modify the output trace data.

If a VM exit is taken on a guest write to CR3 (including "MOV CR3" as well as task switches), the PIP packet normally generated on the CR3 write will be missing.

To avoid decoder confusion when the guest trace is decoded, the VMM should emulate the missing PIP by writing it into the guest output buffer. If the guest CR3 value is manipulated, the VMM may also need to manipulate the IA32_RTIT_CR3_MATCH value, in order to ensure the trace behavior matches the guest's expectation.

Similarly, if a VMM emulates the TSC value by taking a VM exit on RDTSC, the TSC packets generated in the trace may mismatch the TSC values returned by the VMM on RDTSC. To ensure that the trace can be properly aligned with software logs based on RDTSC, the VMM should either make corresponding modifications to the TSC packet values in the guest trace, or use mechanisms such as TSC offsetting or TSC scaling in place of exiting.

33.5.2.4 TSC Scaling

When TSC scaling is enabled for a guest using Intel PT, the VMM should ensure that the value of Maximum Non-Turbo Ratio[15:8] in MSR_PLATFORM_INFO (MSR 0CEH) and the TSC/"core crystal clock" ratio (EBX/EAX) in CPUID leaf 15H are set in a manner consistent with the resulting TSC rate that will be visible to the VM. This will allow the decoder to properly apply TSC packets, MTC packets (based on the core crystal clock or ART, whose frequency is indicated by CPUID leaf 15H), and CBR packets (which indicate the ratio of the processor frequency to the Max Non-Turbo frequency). Absent this, or separate indication of the scaling factor, the decoder will be unable to properly track time in the trace. See Section 33.8.3 for details on tracking time within an Intel PT trace.

33.5.2.5 Failed VM Entry

The packets generated by a failed VM entry depend both on the VMCS configuration, as well as on the type of failure. The results to expect are summarized in the table below. Note that packets in *italics* may or may not be generated, depending on implementation choice, and the point of failure.

Table 33-54. Packets on a Failed VM Entry

Usage Model	Entry Configuration	Early Failure (fall through to next IP)	Late Failure (VM exit like)
System-Wide	No use of "Load IA32_RTIT_CTL" entry control or VM-entry MSR-load area	TIP (NextIP)	*CFE.VMENTRY, FUP(CLIP) if EventEn=1* *PIP(Guest CR3, NR=1), TraceEn 0⟶1 Packets (See Section 33.2.8.3),* PIP(HostCR3, NR=0), TIP(HostIP)
VMM Only	"Load IA32_RTIT_CTL" entry control or VM-entry MSR-load area used to clear TraceEn	TIP (NextIP)	*TraceEn 0⟶1 Packets (See Section 33.2.8.3),* TIP(HostIP)
VM Only	"Load IA32_RTIT_CTL" entry control or VM-entry MSR-load area used to set TraceEn	None	None

33.5.2.6 VMX Abort

VMX abort conditions take the processor into a shutdown state. On a VM exit that leads to VMX abort, some packets (FUP, PIP) may be generated, but any expected TIP, TIP.PGE, or TIP.PGD may be dropped.

33.6 TRACING AND SMM TRANSFER MONITOR (STM)

The SMM-transfer monitor (STM) is a VMM that operates inside SMM while in VMX root operation. An STM operates in conjunction with an executive monitor. The latter operates outside SMM and in VMX root operation. Transitions from the executive monitor or its VMs to the STM are called SMM VM exits. The STM returns from SMM via a VM entry to the VM in VMX non-root operation or the executive monitor in VMX root operation.

Intel PT supports tracing in an STM similar to tracing support for VMX operation as described above in Section 33.5. As a result, on a SMM VM exit resulting from #SMI, TraceEn is neither saved nor cleared by default. Software can save the state of the trace configuration MSRs and clear TraceEn using the MSR load/save lists.

Within Event Trace, SMM VM exits generate packets indicating both an #SMI and a VM exit. Similarly, VM entries that return from SMM generate packets that indicate both an RSM and a VM entry. SMM VM exits initiated by the VMCALL instruction do not generate any CFE packet, though the subsequent VM entry returning from SMM will generate a CFE.RSM.

33.7 PACKET GENERATION SCENARIOS

The following tables provides examples of packet generation for various operations. The following acronyms are used in the packet examples below:

- CLIP - Current LIP
- NLIP - Next Sequential LIP
- BLIP - Branch Target LIP

Table 33-55 illustrates the packets generated by a series of example operations, assuming that PacketEn (TriggerEn && ContextEn && FilterEn && BranchEn) is set before and after the operation.

Table 33-55. Packet Generation under Different Example Operations

Case	Operation	Details	Packets
1	Normal non-jump operation		None
2	Conditional branch	6th branch in internal TNT buffer	TNT
3	Conditional branch	1st..5th branch in internal TNT buffer	None
4	Near indirect JMP or CALL		TIP(BLIP)
5	Direct near JMP or CALL		None
6	Near RET	Uncompressed	TIP(BLIP)
7	Near RET	Compressed, 6th branch in internal TNT buffer	TNT
8	Far Branch	Assumes no update to CR3, CS.L, or CS.D	TIP(BLIP)
9	Far Branch	Assumes update to CR3	PIP(NewCR3), TIP(BLIP)
10	Far Branch	Assumes update to CR3 and CS.D/CS.L	PIP(NewCR3), MODE.Exec, TIP(BLIP)
11	External Interrupt or NMI	Assumes no update to CR3, CS.D, or CS.L	FUP(NLIP), TIP(BLIP)
12	External Interrupt or NMI	Assumes update to CR3 and CS.D/CS.L	FUP(NLIP), PIP(NewCR3), MODE.Exec, TIP(BLIP)
13	Exception/Fault or Software Interrupt	Assumes no update to CR3, CS.D, or CS.L	FUP(CLIP), TIP(BLIP)
14	MOV to CR3		PIP(NewCR3, NR)

Table 33-55. Packet Generation under Different Example Operations

Case	Operation	Details	Packets
15	VM exit	Assumes system-wide tracing, see Section 33.5.2.1	See Table 33-53
16	VM entry	Assumes system-wide tracing, see Section 33.5.2.1	See Table 33-53
17	ENCLU[EENTER] / ENCLU[ERESUME] / ENCLU[EEXIT] / AEX/EEE	Only debug enclaves allow PacketEn to be set during enclave execution. Assumes no change to CS.L or CS.D.	FUP(CLIP), TIP(BLIP)
18	XBEGIN/XACQUIRE/XEND/XRELEASE	Does not begin/end transactional execution	None
19	XBEGIN/XACQUIRE	Assumes beginning of transactional execution	MODE.TSX(InTX=1, TXAbort=0), FUP(CLIP)
20	XEND/XRELEASE	Completes transaction	MODE.TSX(InTX=0, TXAbort=0), FUP(CLIP)
21	XABORT or Asynchronous Abort	Aborts transactional execution	MODE.TSX(InTX=0, TXAbort=1), FUP(CLIP), TIP(BLIP)
22	INIT	On BSP. Assumes no CR3, CS.D, or CS.L update.	FUP(NLIP), TIP(ResetLIP)
23	INIT	On AP, goes to wait-for-SIPI. Assumes no CR3 update.	FUP(NLIP)
24	SIPI	Assumes no CS.D or CS.L update	TIP.PGE(SIPI.LIP)
25	Wake from state deeper than C0.1, P-state change, or other scenario where timing packets (MTC, CYC) may have ceased.	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR
26	UINTR	User interrupt handler entry.	FUP(NLIP)
27	UIRET	Exiting from user interrupt handler.	FUP(NLIP)

Table 33-56 illustrates the packets generated in example scenarios where the operation alters the value of PacketEn. Note that insertion of PSB+ is not included here, though it can be coincident with initial enabling of Intel PT. See Section 33.3.7 for details.

Table 33-56. Packet Generation with Operations That Alter the Value of PacketEn

Case	Operation	PktEn Before	PktEn After	CntxEn After	Details	Packets
1	WRMSR/XRSTORS that changes TraceEn 0 ⟶ 1	0	1	0	TSC if TSCEn=1; TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, MODE.Exec
2	WRMSR/XRSTORS that changes TraceEn 0 ⟶ 1	0	1	1	TSC if TSCEn=1; TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, MODE.Exec, TIP.PGE(NLIP)
3	WRMSR that changes TraceEn 1 ⟶ 0	1	0	D.C.		FUP(CLIP), TIP.PGD()
4	Taken Branch	1	0	1	Source is in IP filter region. Target is outside IP filter region.	TIP.PGD(BLIP)
5	Taken Branch, Interrupt, EEXIT, etc.	0	1	1	Source is outside IP filter region. Target is in IP filter region.	TIP.PGE(BLIP)
6	Far Branch, Interrupt, EENTER, etc.	1	0	0	Requires change to CPL or CR3, or entry to opt-out enclave.	TIP.PGD()

Table 33-56. Packet Generation with Operations That Alter the Value of PacketEn (Contd.)

Case	Operation	PktEn Before	PktEn After	CntxEn After	Details	Packets
7	Trap-like event (external interrupt, NMI, VM exit/entry, etc.)	1	0	0	Requires change to CPL or CR3.	FUP(NLIP), TIP.PGD()
8	Fault-like event (exception/fault, software interrupt, VM exit/entry, etc.)	1	0	0	Requires change to CPL or CR3.	FUP(CLIP), TIP.PGD()
9	SMI, VM exit/entry	1	0	0	TraceEn is cleared.	FUP(NLIP), TIP.PGD()
10	RSM, VM exit/entry	0	1	1	TraceEn is set.	See Case 2 for packets on enable. FUP/TIP.PGE IP is the BLIP.
11	VM Exit	1	0	0	Assumes guest-only tracing, see Section 33.5.2.2. TraceEn is cleared.	FUP(VMCSg.RIP), TIP.PGD()
12	VM entry	0	1	1	Assumes guest-only tracing, see Section 33.5.2.2. TraceEn is set.	TIP.PGE(VMCSg.RIP)

Table 33-57 illustrates examples of PTWRITE, assuming TriggerEn && PTWEn is true.

Table 33-57. Examples of PTWRITE when TriggerEn && PTWEn is True

Case	Operation	ContextEn	Details	Packets
1	MWAIT/UMWAIT gets fault or VM exit.	D.C.		None. Other trace sources may generate packets on fault or VM exit.
2	MWAIT/UMWAIT requests C0, or monitor not armed, or VMX virtual-interrupt delivery.	D.C.		None.
3	MWAIT/UMWAIT enters C-state deeper than C0.1.	0		PWRE(Cx), EXSTOP
4	MWAIT/UMWAIT enters C-state deeper than C0.1.	1		MWAIT(Cy), PWRE(Cx), EXSTOP(IP), FUP(CLIP)
5	HLT, Triple-fault shutdown, other operation that enters C1.	1		PWRE(C1), EXSTOP(IP), FUP(CLIP)
6	Hardware Duty Cycling (HDC).	1	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	PWRE(HW, C6), EXSTOP(IP), FUP(NLIP), TSC?, TMA?, CBR, PWRX(CC6, CC6, 0x8)
7	Wake event during Cx (x > 0).	D.C.	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, PWRX(LCC, DCC, 0x1) Other trace sources may generate packets for the wake operation (e.g., interrupt).

Table 33-58 illustrates examples of Power Event Trace, assuming TriggerEn && PwrEvtEn is true.

Table 33-58. Examples of Power Event Trace when TriggerEn && PwrEvtEn is True

Case	Operation	ContextEn && FilterEn	Details	Packets
1	PTWRITE rm32/64	0		None

Table 33-58. Examples of Power Event Trace when (Contd.)TriggerEn && PwrEvtEn is True

Case	Operation	ContextEn && FilterEn	Details	Packets
2	PTWRITE rm32	1	FUP, PTW.IP=1 if FUPonPTW=1	PTW(IP=1?, 4B, rm32_value), FUP(CLIP)?
3	PTWRITE rm64	1	FUP, PTW.IP=1 if FUPonPTW=1	PTW(IP=1?, 8B, rm64_value), FUP(CLIP)?

Table 33-59 illustrates examples of Event Trace, assuming TriggerEn && ContextEn && EventEn is true. In all cases, other trace sources (e.g., BranchEn), if enabled, may generate additional packets. For details, see the other tables in this section.

Table 33-59. Event Trace Examples when TriggerEn && ContextEn && EventEn is True

Case	Operation	ContextEn Before	ContextEn After	Details	Packets
1	IRET	1	D.C.		CFE.IRET(IP=1), FUP(CLIP)
2	IRET	0	1		CFE(IRET)
3	External interrupt, including NMI	1	D.C.		CFE.INTR(IP=1, Vector), FUP(NLIP)
4	External interrupt, including NMI	1	1	Assumes BranchEn=1, illustrates the shared FUP.	CFE.INTR(IP=0, Vector), FUP(NLIP), TIP(BLIP)
5	SW Interrupt, Exception/Fault other than #PF	1	D.C.		CFE.INTR(IP=1, Vector), FUP(CLIP)
6	Page Fault (#PF)	1	D.C.		EVD.PFA, CFE.INTR(IP=1,14), FUP(CLIP)
7	Page Fault (#PF)	0	D.C.		None
10	SMI	1	D.C.		CFE.SMI(IP=1), FUP(NLIP)
11	RSM, TraceEn restored to 1	D.C.	1		CFE.RSM(IP=0)
12	Entry to Shutdown	1	D.C.		CFE.SHUTDOWN(IP=1), FUP(CLIP)
13	VM exit caused by interrupt, fault, or SMI	1	D.C.	Assumes "Conceal VMX in PT" exit control is 0.	EVD.VMXQ, EVD.VMXR, CFE.VMEXIT_INTR(IP=1, Vector), FUP(VMCSg.LIP)
14	VM exit caused by other than interrupt, fault, or SMI	1	D.C.	Assumes "Conceal VMX in PT" exit control is 0.	EVD.VMXQ, EVD.VMXR, CFE.VMEXIT(IP=1), FUP(VMCSg.LIP)
15	VM exit caused by other than interrupt, fault, or SMI	0	1	Assumes "Conceal VMX in PT" exit control is 0.	CFE.VMEXIT(IP=0)
16	VM entry	1	D.C.	Assumes "Conceal VMX in PT" entry control is 0.	CFE.VMENTRY(IP=1), FUP(VMCSh.LIP)
17	AEX/EEE, from opt-out (non-debug) enclave	0	0		None
18	AEX/EEE, from opt-out (non-debug) enclave	0	1		CFE.INTR(IP=0)
19	AEX, from opt-in (debug) enclave	1	D.C.		CFE.INTR(IP=1, Vec), FUP(AEP LIP)
20	INIT	1	D.C.		CFE.INIT(IP=1), FUP(NLIP)
21	SIPI	1	D.C.		CFE.SIPI(IP=0)

Table 33-59. Event Trace Examples when TriggerEn && ContextEn && EventEn is True

Case	Operation	ContextEn Before	ContextEn After	Details	Packets
22	STI/CLI/POPF	1	1	Assumes a change to RFLAGS.IF.	MODE.Exec, FUP(CLIP)

33.8 SOFTWARE CONSIDERATIONS

33.8.1 Tracing SMM Code

Nothing prevents an SMM handler from configuring and enabling packet generation for its own use. As described in Section Section 33.2.9.3, SMI will always clear TraceEn, so the SMM handler would have to set TraceEn in order to enable tracing. There are some unique aspects and guidelines involved with tracing SMM code, which follow:

1. SMM should save away the existing values of any configuration MSRs that SMM intends to modify for tracing. This will allow the non-SMM tracing context to be restored before RSM.
2. It is recommended that SMM wait until it sets CSbase to 0 before enabling packet generation, to avoid possible LIP vs RIP confusion.
3. Packet output cannot be directed to SMRR memory, even while tracing in SMM.
4. Before performing RSM, SMM should take care to restore modified configuration MSRs to the values they had immediately after #SMI. This involves first disabling packet generation by clearing TraceEn, then restoring any other configuration MSRs that were modified.
5. RSM
   - — Software must ensure that TraceEn=0 at the time of RSM. Tracing RSM is not a supported usage model, and the packets generated by RSM are undefined.
   - — For processors on which Intel PT and LBR use are mutually exclusive (see Section 33.3.1.2), any RSM during which TraceEn is restored to 1 will suspend any LBR or BTS logging.

33.8.2 Cooperative Transition of Multiple Trace Collection Agents

A third-party trace-collection tool should take into consideration the fact that it may be deployed on a processor that supports Intel PT but may run under any operating system.

In such a deployment scenario, Intel recommends that tool agents follow similar principles of cooperative transition of single-use hardware resources, similar to how performance monitoring tools handle performance monitoring hardware:

- Respect the "in-use" ownership of an agent who already configured the trace configuration MSRs, see architectural MSRs with the prefix "IA32_RTIT_" in Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, where "in-use" can be determined by reading the "enable bits" in the configuration MSRs.
- Relinquish ownership of the trace configuration MSRs by clearing the "enabled bits" of those configuration MSRs.

33.8.3 Tracking Time

This section describes the relationships of several clock counters whose update frequencies reside in different domains that feed into the timing packets. To track time, the decoder also needs to know the regularity or irregularity of the occurrences of various timing packets that store those clock counters.

Intel PT provides time information for three different but related domains:

- Processor timestamp counter

  This counter increments at the max non-turbo or P1 frequency, and its value is returned on a RDTSC. Its frequency is fixed. The TSC packet holds the lower 7 bytes of the timestamp counter value. The TSC packet occurs occasionally and are much less frequent than the frequency of the time stamp counter. The timestamp counter will continue to increment when the processor is in deep C-States, with the exception of processors reporting CPUID.80000007H:EDX.InvariantTSC[bit 8] =0.
- Core crystal clock

  The ratio of the core crystal clock to timestamp counter frequency is known as P, and can be calculated as CPUID.15H:EBX[31:0] / CPUID.15H:EAX[31:0]. The frequency of the core crystal clock is fixed and lower than that of the timestamp counter. The periodic MTC packet is generated based on software-selected multiples of the crystal clock frequency. The MTC packet is expected to occur more frequently than the TSC packet.
- Processor core clock

  The processor core clock frequency can vary due to P-state and thermal conditions. The CYC packet provides elapsed time as measured in processor core clock cycles relative to the last CYC packet.

A decoder can use all or some combination of these packets to track time at different resolutions throughout the trace packets.

33.8.3.1 Time Domain Relationships

The three domains are related by the following formula:

```
TimeStampValue = (CoreCrystalClockValue * P) + AdjustedProcessorCycles + Software_Offset;
```

The CoreCrystalClockValue, also known as the Always Running Timer (ART) value, can provide the coarse-grained component of the TSC value. P, or the TSC/ART ratio, can be derived from CPUID leaf 15H, as described in Section 33.8.3.

The AdjustedProcessorCycles component provides the fine-grained distance from the rising edge of the last core crystal clock. Specifically, it is a cycle count in the same frequency as the timestamp counter from the last crystal clock rising edge. The value is adjusted based on the ratio of the processor core clock frequency to the Maximum Non-Turbo (or P1) frequency.

The Software_Offsets component includes software offsets that are factored into the timestamp value, such as IA32_TSC_ADJUST.

33.8.3.2 Estimating TSC within Intel PT

For many usages, it may be useful to have an estimated timestamp value for all points in the trace. The formula provided in Section 33.8.3.1 above provides the framework for how such an estimate can be calculated from the various timing packets present in the trace.

The TSC packet provides the precise timestamp value at the time it is generated; however, TSC packets are infrequent, and estimates of the current timestamp value based purely on TSC packets are likely to be very inaccurate for this reason. In order to get more precise timing information between TSC packets, CYC packets and/or MTC packets should be enabled.

MTC packets provide incremental updates of the CoreCrystalClockValue. On processors that support CPUID leaf 15H, the frequency of the timestamp counter and the core crystal clock is fixed, thus MTC packets provide a means to update the running timestamp estimate. Between two MTC packets A and B, the number of crystal clock cycles passed is calculated from the 8-bit payloads of respective MTC packets:

(CTC_B - CTC_A), where CTC_i = MTC_i[15:8] << IA32_RTIT_CTL.MTCFreq and i = A, B.

The time from a TSC packet to the subsequent MTC packet can be calculated using the TMA packet that follows the TSC packet. The TMA packet provides both the crystal clock value (lower 16 bits, in the CTC field) and the AdjustedProcessorCycles value (in the FastCounter field) that can be used in the calculation of the corresponding core crystal clock value of the TSC packet.

When the next MTC after a pair of TSC/TMA is seen, the number of crystal clocks passed since the TSC packet can be calculated by subtracting the TMA.CTC value from the time indicated by the MTC_{Next} packet by

CTC_{Delta}[15:0] = (CTC_{Next}[15:0] - TMA.CTC[15:0]), where CTC_{Next} = $MTC_{Payload}$ << IA32_RTIT_CTL.MTCFreq.

The TMA.FastCounter field provides the number of AdjustedProcessorCycles since the last crystal clock rising edge, from which it can be determined the percentage of the next crystal clock cycle that had passed at the time of the TSC packet.

CYC packets can provide further precision of an estimated timestamp value to many non-timing packets, by providing an indication of the time passed between other timing packets (MTCs or TSCs).

When enabled, CYC packets are sent preceding each CYC-eligible packet, and provide the number of processor core clock cycles that have passed since the last CYC packet. Thus between MTCs and TSCs, the accumulated CYC values can be used to estimate the AdjustedProcessorCycles component of the timestamp value. The accumulated CPU cycles will have to be adjusted to account for the difference in frequency between the processor core clock and the P1 frequency. The necessary adjustment can be estimated using the core:bus ratio value given in the CBR packet, by multiplying the accumulated cycle count value by $P1/CBR_{payload}$.

Note that stand-alone TSC packets (that is, TSC packets that are not a part of a PSB+) are typically generated only when generation of other timing packets (MTCs and CYCs) has ceased for a period of time. Example scenarios include when Intel PT is re-enabled, or on wake after a sleep state. Thus any calculated estimate of the timestamp value leading up to a TSC packet will likely result in a discrepancy, which the TSC packet serves to correct.

A greater level of precision may be achieved by calculating the CPU clock frequency, see Section 33.8.3.4 below for a method to do so using Intel PT packets.

CYCs can be used to estimate time between TSCs even without MTCs, though this will likely result in a reduction in estimated TSC precision.

33.8.3.3 VMX TSC Manipulation

When software executes in non-Root operation, additional offset and scaling factors may be applied to the TSC value. These are optional, but may be enabled via VMCS controls on a per-VM basis. See Chapter 26, "VMX Non-Root Operation," for details on VMX TSC offsetting and TSC scaling.

Like the value returned by RDTSC, TSC packets will include these adjustments, but other timing packets (such as MTC, CYC, and CBR) are not impacted. In order to use the algorithm above to estimate the TSC value when TSC scaling is in use, it will be necessary for software to account for the scaling factor. See Section 33.5.2.4 for details.

33.8.3.4 Calculating Frequency with Intel PT

Because Intel PT can provide both wall-clock time and processor clock cycle time, it can be used to measure the processor core clock frequency. Either TSC or MTC packets can be used to track the wall-clock time. By using CYC packets to count the number of processor core cycles that pass in between a pair of wall-clock time packets, the ratio between processor core clock frequency and TSC frequency can be derived. If the P1 frequency is known, it can be applied to determine the CPU frequency. See Section 33.8.3.1 above for details on the relationship between TSC, MTC, and CYC.

17. Updates to Chapter 35, Volume 3D

Change bars and violet 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:

- Updated Section 35.9.2.2, "Page Fault Error Code," to remove Table 35-14 and direct readers to Figure 4-12, "Page-Fault Error Code."

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 type[1] (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	The RESERVED field consists of the following: ▪ 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. The entire 3834 byte field must be cleared prior to executing ECREATE.

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 EDBGRD 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 0DH 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 0DH 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: • Unconditionally supported exceptions: #DE, #DB, #BP, #BR, #UD, #MF, #AC, #XM. • Conditionally supported exception: — #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

A page-fault error code may be reported in EXINFO.ERRCD. The format of the error code is given in Figure 4-12, "Page-Fault Error Code."

Page faults that would report an error code clearing the U/S bit (indicating a supervisor-mode access) 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 35-14. 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 35-15. 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 35-16. 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 35-17. 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 35-18. 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 35-19. 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 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

SIGSTRUCT must be page aligned

In column 5 of Table 35-20, 'Y' indicates that this field should be included in the signature generated by the developer.

Table 35-20. 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 yyyymmdd 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
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[1]	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

Table 35-20. Layout of Enclave Signature Structure (SIGSTRUCT) (Contd.)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
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

NOTES:

1. If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISCSELECT must be 0.

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 35-21. 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_ATTTRIBUTES_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 35-22. 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 35-22. Layout of REPORT (Contd.)

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 35-23. 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 35-24. 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 35-25. Supported KEYName Values

Key Name	Value	Description
EINITTOKEN_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 35-26. 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 35-27. 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 35-28. 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 35-29. 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 35-30. 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 35-31. 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 37, Volume 3D

Change bars and violet text show changes to Chapter 37 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

Changes to this chapter:

- Updated Section 37.4.1, "AEX Operational Detail."

CHAPTER 37
ENCLAVE EXITING EVENTS

Certain events, such as exceptions and interrupts, incident to (but asynchronous with) enclave execution may cause control to transition outside of enclave mode. (Most of these also cause a change of privilege level.) To protect the integrity and security of the enclave, the processor will exit the enclave (and enclave mode) before invoking the handler for such an event. For that reason, such events are called **enclave-exiting events** (EEE); EEEs include external interrupts, non-maskable interrupts, system-management interrupts, exceptions, and VM exits.

The process of leaving an enclave in response to an EEE is called an **asynchronous enclave exit** (AEX). To protect the secrecy of the enclave, an AEX saves the state of certain registers within enclave memory and then loads those registers with fixed values called **synthetic state**.

37.1 COMPATIBLE SWITCH TO THE EXITING STACK OF AEX

AEXs load registers with a pre-determined synthetic state. These registers may be later pushed onto the appropriate stack in a form as defined by the enclave-exiting event. To allow enclave execution to resume after the invoking handler has processed the enclave exiting event, the asynchronous enclave exit loads the address of trampoline code outside of the enclave into RIP. This trampoline code eventually returns to the enclave by means of an ENCLU(ERESUME) leaf function. Prior to exiting the enclave the RSP and RBP registers are restored to their values prior to enclave entry.

The stack to be used is chosen using the same rules as for non-SGX mode:

- If there is a privilege level change, the stack will be the one associated with the new ring.
- If there is no privilege level change, the current application stack is used.
- If the IA-32e IST mechanism is used, the exit stack is chosen using that method.

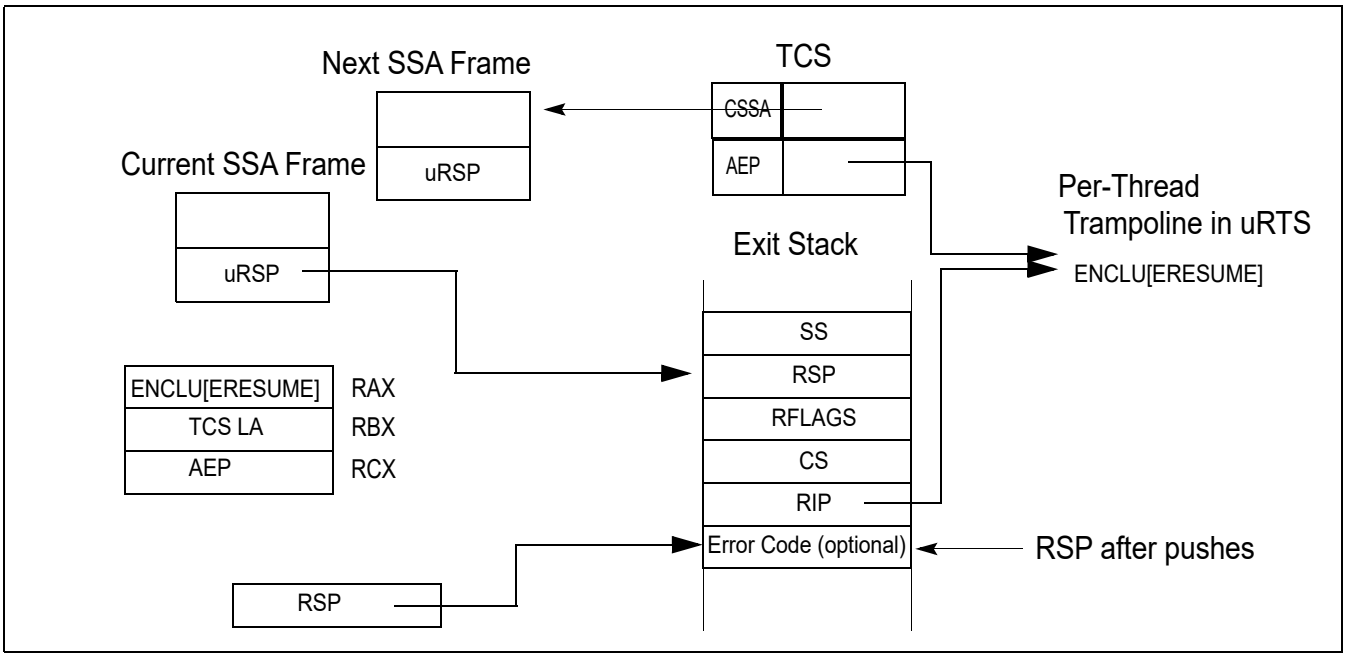


Figure 37-1. Exit Stack Just After Interrupt with Stack Switch

In all cases, the choice of exit stack and the information pushed onto it is consistent with non-SGX operation. Figure 37-1 shows the Application and Exiting Stacks after an exit with a stack switch. An exit without a stack switch uses the Application Stack. The ERESUME leaf index value is placed into RAX, the TCS pointer is placed in RBX and the AEP (see below) is placed into RCX to facilitate resuming the enclave after the exit.

Upon an AEX, the AEP (Asynchronous Exit Pointer) is loaded into the RIP. The AEP points to a trampoline code sequence which includes the ERESUME instruction that is later used to reenter the enclave.

The following bits of RFLAGS are cleared before RFLAGS is pushed onto the exit stack: CF, PF, AF, ZF, SF, OF, RF. The remaining bits are left unchanged.

37.2 STATE SAVING BY AEX

The State Save Area holds the processor state at the time of an AEX. To allow handling events within the enclave and re-entering it after an AEX, the SSA can be a stack of multiple SSA frames as illustrated in Figure 37-2.

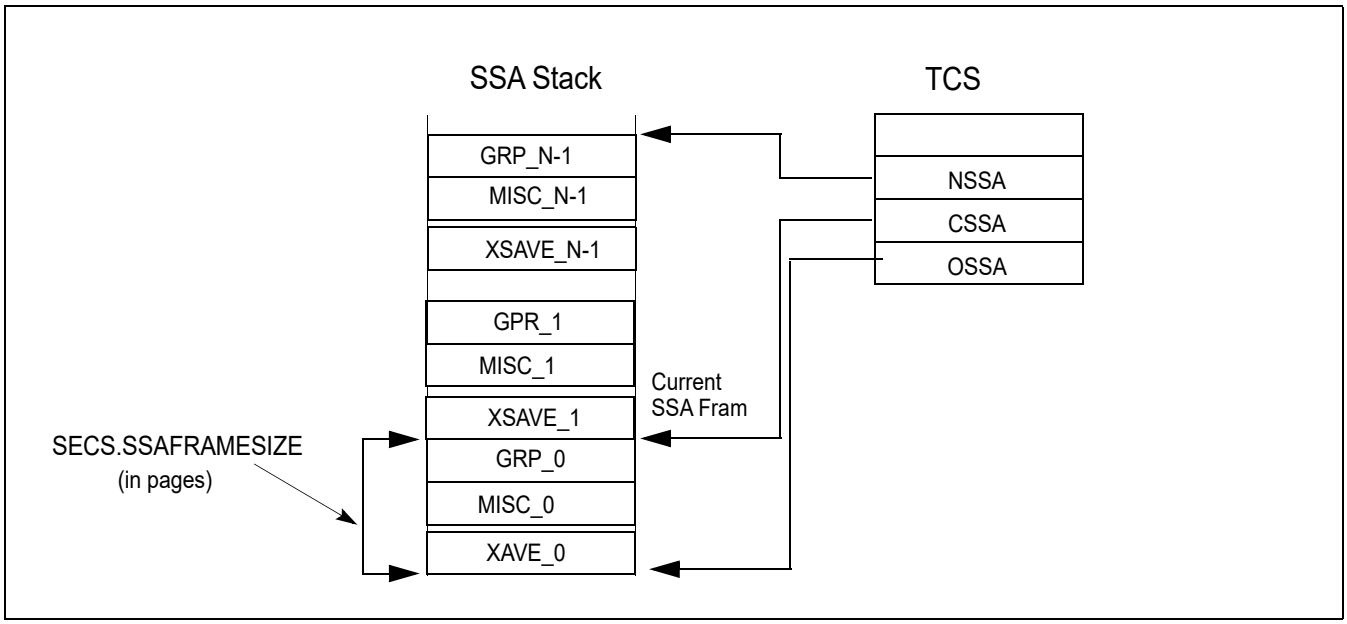


Figure 37-2. The SSA Stack

The location of the SSA frames to be used is controlled by the following variables in the TCS and the SECS:

- Size of a frame in the State Save Area (SECS.SSAFRAMESIZE): This defines the number of 4-KByte pages in a single frame in the State Save Area. The SSA frame size must be large enough to hold the GPR state, the XSAVE state, and the MISC state.
- Base address of the enclave (SECS.BASEADDR): This defines the enclave's base linear address from which the offset to the base of the SSA stack is calculated.
- Number of State Save Area Slots (TCS.NSSA): This defines the total number of slots (frames) in the State Save Area stack.
- Current State Save Area Slot (TCS.CSSA): This defines the slot to use on the next exit.
- State Save Area Offset (TCS.OSSA): This defines the offset of the base address of a set of State Save Area slots from the enclave’s base address.

When an AEX occurs, hardware selects the SSA frame to use by examining TCS.CSSA. Processor state is saved into the SSA frame (see Section 37.4) and loaded with a synthetic state (as described in Section 37.3.1) to avoid leaking secrets, RSP and RBP are restored to their values prior to enclave entry, and TCS.CSSA is incremented. As will be described later, if an exception takes the last slot, it will not be possible to reenter the enclave to handle the

exception from within the enclave. A subsequent ERESUME restores the processor state from the current SSA frame and frees the SSA frame.

The format of the XSAVE section of SSA is identical to the format used by the XSAVE/XRSTOR instructions. On EENTER, CSSA must be less than NSSA, ensuring that there is at least one State Save Area slot available for exits. If there is no free SSA frame when executing EENTER, the entry will fail.

37.3 SYNTHETIC STATE ON ASYNCHRONOUS ENCLAVE EXIT

37.3.1 Processor Synthetic State on Asynchronous Enclave Exit

Table 37-1 shows the synthetic state loaded on AEX. The values shown are the lower 32 bits when the processor is in 32 bit mode and 64 bits when the processor is in 64 bit mode.

Table 37-1. GPR, x87 Synthetic States on Asynchronous Enclave Exit

Register	Value
RAX	3 (ENCLU[3] is ERESUME).
RBX	Pointer to TCS of interrupted enclave thread.
RCX	AEP of interrupted enclave thread.
RDX, RSI, RDI	0.
RSP	Restored from SSA.uRSP.
RBP	Restored from SSA.uRBP.
R8-R15	0 in 64-bit mode; unchanged in 32-bit mode.
RIP	AEP of interrupted enclave thread.
RFLAGS	CF, PF, AF, ZF, SF, OF, RF bits are cleared. All other bits are left unchanged.
x87/SSE State	Unless otherwise listed here, all x87 and SSE state are set to the INIT state. The INIT state is the state that would be loaded by the XRSTOR instruction with bits 1:0 both set in the requested feature bitmask (RFBM), and both clear in XSTATE_BV the XSAVE header.
FCW	On #MF exception: set to 037EH. On all other exits: set to 037FH.
FSW	On #MF exception: set to 8081H. On all other exits: set to 0H.
MXCSR	On #XM exception: set to 1F01H. On all other exits: set to 1FB0H.
CR2	If the event that caused the AEX is a #PF, and the #PF does not directly cause a VM exit, then the low 12 bits are cleared. If the #PF leads directly to a VM exit, CR2 is not updated (usual IA behavior). Note: The low 12 bits are not cleared if a #PF is encountered during the delivery of the EEE that caused the AEX. This is because the #PF was not the EEE.
FS, GS	Restored to values as of most recent EENTER/ERESUME.

37.3.2 Synthetic State for Extended Features

When CR4.OSXSAVE = 1, extended features (those controlled by XCR0[63:2]) are set to their respective INIT states when this corresponding bit of SECS.XFRM is set. The INIT state is the state that would be loaded by the XRSTOR instruction had the instruction mask and the XSTATE_BV field of the XSAVE header each contained the value XFRM. (When the AEX occurs in 32-bit mode, those features that do not exist in 32-bit mode are unchanged.)

37.3.3 Synthetic State for MISC Features

State represented by SECS.MISCSELECT might also be overridden by synthetic state after it has been saved into the SSA. State represented by MISCSELECT[0] is not overridden but if the exiting event is a page fault then lower 12 bits of CR2 are cleared.

37.4 AEX FLOW

On Enclave Exiting Events (interrupts, exceptions, VM exits or SMIs), the processor state is securely saved inside the enclave, a synthetic state is loaded and the enclave is exited. The EEE then proceeds in the usual exit-defined fashion. The following sections describes the details of an AEX:

1. The exact processor state saved into the current SSA frame depends on whether the enclave is a 32-bit or a 64-bit enclave. 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 stored. The upper 32 bits of the legacy registers and the 64-bit registers (R8 … R15) are not stored.

   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 stored.

   The state of those extended features specified by SECS.ATTRIBUTES.XFRM are stored into the XSAVE area of the current SSA frame. The layout of the x87 and XMM portions (the 1st 512 bytes) depends on the current values of IA32_EFER.LMA and CS.L:

   If IA32_EFER.LMA = 0 || CS.L = 0, the same format (32-bit) that XSAVE/FXSAVE uses with these values.

   If IA32_EFER.LMA = 1 && CS.L = 1, the same format (64-bit) that XSAVE/FXSAVE uses with these values when REX.W = 1.

   The cause of the AEX is saved in the EXITINFO field. See Table 35-10 for details and values of the various fields.

   The state of those miscellaneous features (see Section 35.7.2) specified by SECS.MISCSELECT are stored into the MISC area of the current SSA frame.

   If CET was enabled in the enclave, then the CET state of the enclave is saved in the CET state save area. If shadow stacks were enabled in the enclave, then the SSP is also saved into the TCS.PREVSSP field.
2. Synthetic state is created for a number of processor registers to present an opaque view of the enclave state. Table 37-1 shows the values for GPRs, x87, SSE, FS, GS, Debug, and performance monitoring on AEX. The synthetic state for other extended features (those controlled by XCR0[62:2]) is set to their respective INIT states when their corresponding bit of SECS.ATTRIBUTES.XFRM is set. The INIT state is that state as defined by the behavior of the XRSTOR instruction when HEADER.XSTATE_BV[n] is 0. Synthetic state of those miscellaneous features specified by SECS.MISCSELECT depends on the miscellaneous feature. There is no synthetic state required for the miscellaneous state controlled by SECS.MISCSELECT[0].
3. Any code and data breakpoints that were suppressed at the time of enclave entry are unsuppressed when exiting the enclave.
4. RFLAGS.TF is set to the value that it had at the time of the most recent enclave entry (except for the situation that the entry was opt-in for debug; see Section 40.2). In the SSA, RFLAGS.TF is set to 0.
5. RFLAGS.RF is set to 0 in the synthetic state. In the SSA, the value saved is the same as what would have been saved on stack in the non-SGX case (architectural value of RF). Thus, AEXs due to interrupts, traps, and code breakpoints save RF unmodified into SSA, while AEXs due to other faults save RF as 1 in the SSA.

   If the event causing AEX happened on intermediate iteration of a REP-prefixed instruction, then RF=1 is saved on SSA, irrespective of its priority.
6. Any performance monitoring activity (including PEBS) or profiling activity (LBR, Tracing using Intel PT) on the exiting thread that was suppressed due to the enclave entry on that thread is unsuppressed. Any counting that had been demoted from AnyThread counting to MyThread counting (on one logical processor) is promoted back to AnyThread counting.
7. The CET state of the enclosing application is restored to the state at the time of the most recent enclave entry, and if CET indirect branch tracking was enabled then the indirect branch tracker is unsuppressed and moved to the WAIT_FOR_ENDBRANCH state.

37.4.1 AEX Operational Detail

Temp Variables in AEX Operational Flow

Name	Type	Size (bits)	Description
TMP_RIP	Effective Address	32/64	Address of instruction at which to resume execution on ERESUME.
TMP_MODE64	binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1)).
TMP_BRANCH_RECORD	LBR Record	2x64	From/To address to be pushed onto LBR stack.

The pseudo code in this section describes the internal operations that are executed when an AEX occurs in enclave mode. These operations occur just before the normal interrupt or exception processing occurs.

```
(* Save RIP for later use *)
TMP_RIP = Linear Address of Resume RIP
(* Is the processor in 64-bit mode? *)
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Save all registers, When saving EFLAGS, the TF bit is set to 0 and
   the RF bit is set to what would have been saved on stack in the non-SGX case *)

 IF (TMP_MODE64 = 0)
   THEN
       Save EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EFLAGS, EIP into the current SSA frame using
CR_GPR_PA; (* see Table 38-5 for list of CREGs used to describe internal operation within Intel SGX *)
       SSA.RFLAGS.TF := 0;
   ELSE    (* TMP_MODE64 = 1 *)
       Save RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8-R15, RFLAGS, RIP into the current SSA frame using
CR_GPR_PA;
       SSA.RFLAGS.TF := 0;
FI;
Save FS and GS BASE into SSA using CR_GPR_PA;

(* store XSAVE state into the current SSA frame's XSAVE area using the physical addresses
   that were determined and cached at enclave entry time with CR_XSAVE_PAGE_i. *)
For each XSAVE state i defined by (SECS.ATTRIBUTES.XFRM[i] = 1, destination address cached in
CR_XSAVE_PAGE_i)
  SSA.XSAVE.i := XSAVE_STATE_i;

(* Clear bytes 8 to 23 of XSAVE_HEADER, i.e., the next 16 bytes after XHEADER_BV *)

CR_XSAVE_PAGE_0.XHEADER_BV[191:64] := 0;

(* Clear bits in XHEADER_BV[63:0] that are not enabled in ATTRIBUTES.XFRM *)

CR_XSAVE_PAGE_0.XHEADER_BV[63:0] :=
   CR_XSAVE_PAGE_0.XHEADER_BV[63:0] & SECS(CR_ACTIVE_SECS).ATTRIBUTES.XFRM;
   Apply synthetic state to GPRs, RFLAGS, extended features, etc.

(* Restore the RSP and RBP from the current SSA frame's GPR area using the physical address
   that was determined and cached at enclave entry time with CR_GPR_PA. *)
RSP := CR_GPR_PA.URSP;
RBP := CR_GPR_PA.URBP;
```

```
(* Restore the FS and GS *)
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;

(* Examine exception code and update enclave internal states*)
exception_code := Exception or interrupt vector;

(* Indicate the exit reason in SSA *)
IF (exception_code = (#DE OR #DB OR #BP OR #BR OR #UD OR #MF OR #AC OR #XM ))
   THEN
       CR_GPR_PA.EXITINFO.VECTOR := exception_code;
       IF (exception_code = #BP)
           THEN CR_GPR_PA.EXITINFO.EXIT_TYPE := 6;
           ELSE CR_GPR_PA.EXITINFO.EXIT_TYPE := 3;
       FI;
       CR_GPR_PA.EXITINFO.VALID := 1;
ELSE IF (SECS.MISCSELECT[0] is set (* Check SECS.MISCSELECT using CR_ACTIVE_SECS *)
            AND (exception_code is #GP OR (exception_code is #PF AND PFEC.U/S = 1)))
   THEN
       CR_GPR_PA.EXITINFO.VECTOR := exception_code;
       CR_GPR_PA.EXITINFO.EXIT_TYPE := 3;
       IF (exception_code is #PF)
           THEN
               SSA.MISC.EXINFO. MADDR := CR2;
               SSA.MISC.EXINFO.ERRCD := PFEC; (* Page-Fault Exception Error Code *)
               SSA.MISC.EXINFO.RESERVED := 0;
       ELSE
           SSA.MISC.EXINFO. MADDR := 0;
           SSA.MISC.EXINFO.ERRCD := GPEC; (* General Protection Exception Error Code *)
           SSA.MISC.EXINFO.RESERVED := 0;
       FI;
       CR_GPR_PA.EXITINFO.VALID := 1;
ELSE IF (SECS.MISCSELECT[1] is set AND exception_code is #CP) (* Check SECS.MISCSELECT using
            CR_ACTIVE_SECS *)
   THEN
       CR_GPR_PA.EXITINFO.VECTOR := exception_code;
       CR_GPR_PA.EXITINFO.EXIT_TYPE := 3;
       CR_GPR_PA.EXITINFO.VALID := 1;
       SSA.MISC.EXINFO. MADDR := 0;
       SSA.MISC.EXINFO.ERRCD := CPEC; (* Control Protection Exception Error Code *)
       SSA.MISC.EXINFO.RESERVED := 0;
ELSE
   SSA.MISC.EXINFO.MADDR := 0;
   SSA.MISC.EXINFO.ERRCD := 0;
   SSA.MISC.EXINFO.RESERVED := 0;
   CR_GPR_PA.EXITINFO.VECTOR := 0;
   CR_GPR_PA.EXITINFO.EXIT_TYPE := 0;
```

```
    CR_GPR_PA.EXITINFO.VALID := 0;
FI;

(* Execution will resume at the AEP *)
RIP := CR_TCS_PA.AEP;

(* Set EAX to the ERESUME leaf index *)
EAX := 3;

(* Put the TCS LA into RBX for later use by ERESUME *)
RBX := CR_TCS_LA;

(* Put the AEP into RCX for later use by ERESUME *)
RCX := CR_TCS_PA.AEP;

(* Increment the SSA frame # *)
CR_TCS_PA.CSSA := CR_TCS_PA.CSSA + 1;

(* Restore XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
    THEN XCR0 := CR_SAVE_XCR0; FI;

Un-suppress all code breakpoints that are outside ELRANGE

IF (CPUID.(EAX=12H, ECX=1):EAX[6]= 1)
    THEN
        IF (CR4.CET == 1 AND IA32_U_CET.SH_STK_EN == 1)
            THEN
                CR_CET_SAVE_AREA_PA.SSP := SSP;
                CR_TCS_PA.PREVSSP := SSP;
        FI;
        IF (CR4.CET == 1 AND IA32_U_CET.ENDBR_EN == 1)
            THEN
                CR_CET_SAVE_AREA_PA.TRACKER := IA32_U_CET.TRACKER;
                CR_CET_SAVE_AREA_PA.SUPPRESS := IA32_U_CET.SUPPRESS;
        FI;
FI;
IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7H, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* restore enclosing applications CET state *)
        IA32_U_CET := CR_SAVE_IA32_U_CET;

        IF (CPUID.(EAX=7, ECX=0):ECX[CET_SS])
            SSP := CR_SAVE_SSP; FI;

        (* If indirect branch tracking enabled for enclosing application *)
        (* then move the tracker to wait_for_endbranch *)
        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;
```

```
(* Update the thread context to show not in enclave mode *)
CR_ENCLAVE_MODE := 0;

(* Assure consistent translations. *)
Flush linear context including TLBs and paging-structure caches

IF (CR_DBGOPTIN = 0)
    THEN
        Un-suppress all breakpoints that overlap ELRANGE
        (* Clear suppressed breakpoint matches *)
        Restore suppressed breakpoint matches
        (* Restore TF *)
        RFLAGS.TF := CR_SAVE_TF;
        Un-suppress monitor trap flag;
        Un-suppress branch recording facilities;
        Un-suppress all suppressed performance monitoring activity;
        Promote any sibling-thread counters that were demoted from AnyThread to MyThread during enclave
entry back to AnyThread;
FI;

IF the “monitor trap flag” VM-execution control is 1
    THEN Pend MTF VM Exit at the end of exit; FI;

(* Clear low 12 bits of CR2 on #PF *)
IF (Exception code is #PF)
    THEN CR2 := CR2 & ~0xFFF; FI;

(* end_of_flow *)
(* Execution continues with normal event processing. *)
```

19. Updates to Chapter 38, Volume 3D

Change bars and violet 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:

- Updated the "Operation" section for the EENTER and ERESUME instructions.

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)	
EDBGRD	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	RDX
EREPORT	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	RDX
EDECVIRTCHILD	00H (In)	EPCPAGE (In, EA)	SECS (In, EA)	
EINCVIRTCHILD	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_XCR0	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 leafs 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
EDBGRD	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
EACCEPTCOPY	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	Source	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EADD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Exclusive	#GP	Concurrent	
EAUG	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EBLOCK	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ECREATE	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDBGRD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDBGWR	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDECVIRTCHILD	Target	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EENTERTCS	SECS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EEXIT			Concurrent		Concurrent		Concurrent	
EEXTEND	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	
EGETKEY	KEYREQUEST	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EINCVIRTCHILD	Target	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EINIT	SECS	[DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	
ELDB/ELDU	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EDLBC/ELDUC	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EMODPE	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EMODPR	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	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	ZO	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
ZO	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	ZO	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
ZO	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	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 operating in 16-bit mode.
	If data segment is in 16-bit mode.
	If CR0.PG = 0 or CR0.NE= 0.
#NM	If CR0.TS = 1.

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 0F 01 C0 ENCLV	ZO	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
ZO	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_SRCPGE[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;
        ELSE
            (* 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 PAGEINFO (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 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;
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_SRCPGE[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)	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)	If a page fault occurs in accessing memory operands.
	If the SECS destination is outside the EPC.

64-Bit Mode Exceptions

#GP(0)	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 EDBGRD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDBGRD	Target [DS:RCX]	Shared	#GP	

Table 38-19. Additional Concurrency Restrictions of EDBGRD

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
EDBGRD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGRD 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;

(* 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 EDBGRD 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:RCS).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
```

```
        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 & 0FF8H) )
    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 compromising 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_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.

```
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)	If the address in RBX is outside the DS segment limit.
	If RBX points to an SECS page which is not the SECS of the data chunk.
	If the address in RCX is outside the DS segment limit.
	If RCX points to a memory location not 256Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	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 non-EPC page.
	If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RBX is non-canonical form.
	If RBX points to an SECS page which is not the SECS of the data chunk.
	If RCX is non-canonical form.
	If RCX points to a memory location not 256 Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	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 non-EPC page.
	If the address in RCX points to an invalid EPC page.

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 EINITTOKEN (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 EINITTOKEN. 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 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

The EINITTOKEN contains the MRENCLAVE, MRSIGNER, and ATTRIBUTES. These values must match the corresponding values in the SECS. If the EINITTOKEN was created with a debug launch key, the enclave must be in debug mode as well.

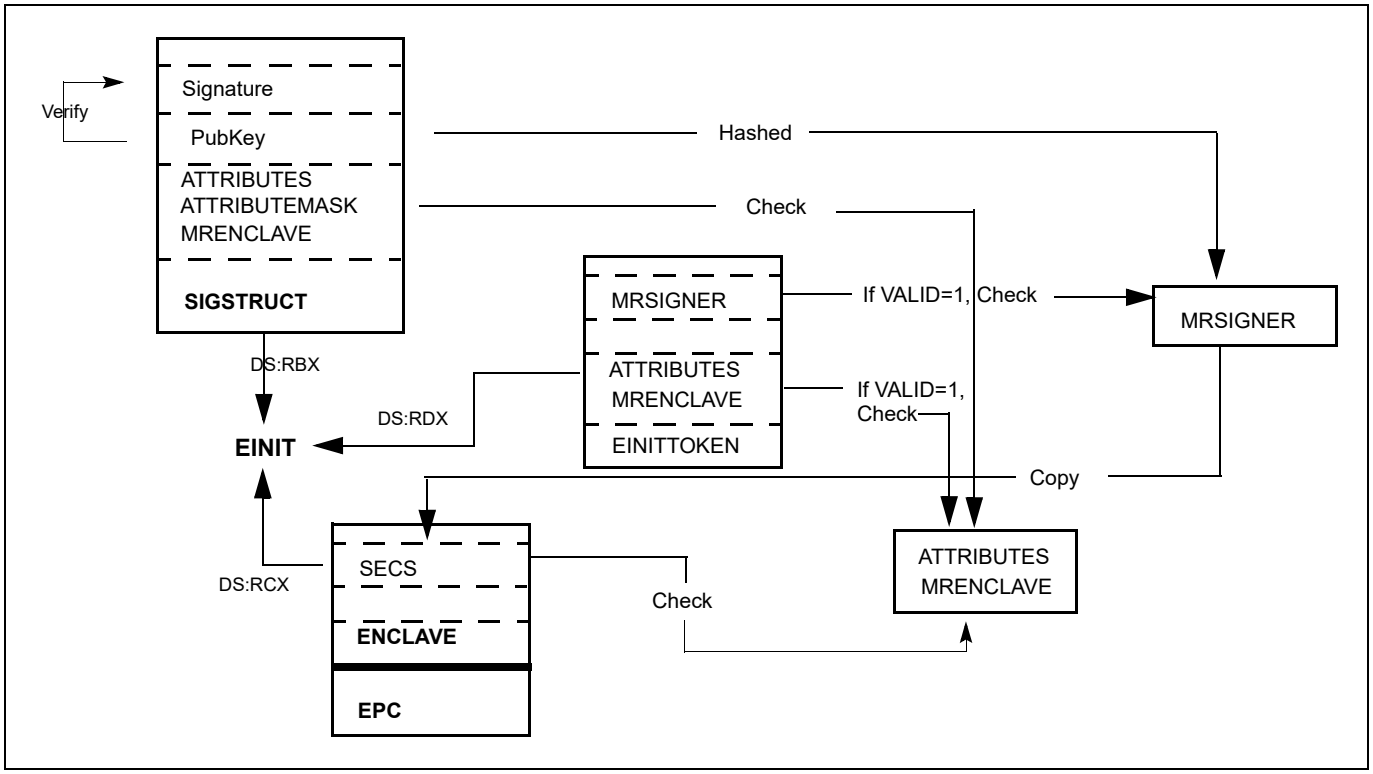


Figure 38-1. Relationships Between SECS, SIGSTRUCT, and EINITTOKEN

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_LEPUB-KEYHASH.
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 ENIT

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_ATTRIB-UTES_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.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
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 EINITTOKEN (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 EINITTOKEN 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)	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)	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)	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)	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 lie 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 permitted 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.SECS;
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 & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~0FFFH).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)	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)	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)	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)	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 := 0;
```

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.

ETRACK—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0CH ENCLS[ETRACK]	IR	V/V	SGX1	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	ETRACK (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 ETRACK leaf function.

ETRACK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 38-45. ETRACK Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	ETRACK successful.
SGX_PREV_TRK_INCMPL	All processors did not complete the previous shoot-down sequence.

Concurrency Restrictions

Table 38-46. Base Concurrency Restrictions of ETRACK

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ETRACK	SECS [DS:RCX]	Shared	#GP	

Table 38-47. Additional Concurrency Restrictions of ETRACK

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
ETRACK	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. If the 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.

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 & ~0FFFH).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)	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)	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)	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)	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 4KBbytes 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:TMPSSA_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 XCR0.
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, XCR0 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 FS 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[bits 11:9] != 001B))
    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 & FFFFFFFFFFFFFFFCH) ≠ 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 & XCR0) ≠ TMP_SECS.ATTRIBUES.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 XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 := XCR0;
    XCR0 := 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[bit 9];
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS[bit 21];
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS[bit 9];
GS.S := 1;
```

```
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS[bit 21];
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)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
	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 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 = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	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)	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 or not properly aligned.
	If the target address is not canonical.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	If a page fault occurs in accessing memory operands.
	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.

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 XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
    XCR0 := CR_SAVE__XCR0;
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.

EEGETKEY 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 PRODID	ISVEXT PRODID	ISVFAM ILYID	MRENCLAVE	MRSIGNER	CONFIG ID	CONFIGS VN	RAND
Source	Key Dependent Constant	Y := SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES; R := AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;	CR_SGX OWNER EPOCH	Y := CPUSVN Register; R := Req.CPU SVN;	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
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 hardcode 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)
        TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
            FI;

    (* Include enclave Config ID? *)
    TMP_CONFIGID := 0;
    TMP_CONFIGSVN := 0;
    IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
        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 := 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.ATTRIBUTEMASK;
            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;
```

```
        (* 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.ATTRIBUTEMASK;
        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)
        TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
            FI;

    (* Include enclave Config ID? *)
    TMP_CONFIGID := 0;
    TMP_CONFIGSVN := 0;
    IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
        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)	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.

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.

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) )
```

```
    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:RCX).PENDING = 1) or
    (EPCM(DS:RCX).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 XCR0.
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, XCR0 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 40.2.3).
- 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 FS 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[bits 11:9] != 001B))
```

```
    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).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 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 ( (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 & FFFFFFFFFFFFFFFCH) ≠ 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 & XCR0) ≠ 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:TMPSSA_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;

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;
```

```
        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;
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 sate 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 XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
        IF (CR4.OSXSAVE = 1)
            THEN
                CR_SAVE_XCR0 := XCR0;
                XCR0 := 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 XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
        IF (CR4.OSXSAVE = 1)
            THEN
                CR_SAVE_XCR0 := XCR0;
                XCR0 := 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[bit 9];
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS[bit 21];
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS[bit 9];
GS.S := 1;
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS[bit 21];
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)	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 = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	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)	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 canonical.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	If a page fault occurs in accessing memory operands.
	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.

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 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
    (* EDECVIRTCHILD 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 EDECVIRTCHILD fails due to concurrent operation with another SGX instruction, or if there is a VIRTCHILDCNT underflow. Otherwise 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.
	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.

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 EINCVIRTCHILD

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
EINCVIRTCHILD	Target [DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EINCVIRTCHILD Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.

EINCVIRTCHILD Return Value in RAX

Error	Value	Description
No Error	0	EINCVIRTCHILD 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 virtchild 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)	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.
	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.

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.

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 2, Volume 4

Change bars and violet text 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:

- Updated Table 2-2, "IA-32 Architectural MSRs," to correct four instances of "MAXPHYSADDR." The correct term is "MAXPHYADDR."
- Updated the IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS_RESET, and IA32_PERF_GLOBAL_STATUS_SET MSRs in Table 2-2, "IA-32 Architectural MSRs."
- Section 2.17.8, "MSRs Specific to the 4th and 5th Generation Intel® Xeon® Scalable Processor Families," was updated to add MSR_DRAM_ENERGY_STATUS (MSR address 619H) to Table 2-52, "Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH."
- Updated all instances of "COS" to "CLOS" in this chapter. These changes are not marked with change bars or violet text.

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_AAH	Intel® Core™ Ultra 7 processors supporting Meteor Lake performance hybrid architecture
06_CFH	5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids 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	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

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
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
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

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 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: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 0H, 0	IA32_P5_MC_ADDR (P5_MC_ADDR)	
See Section 2.23, "MSRs in Pentium Processors."		Pentium Processor (05_01H)
Register Address: 1H, 1	IA32_P5_MC_TYPE (P5_MC_TYPE)	
See Section 2.23, "MSRs in Pentium Processors."		DF_DM = 05_01H

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination."		0F_03H
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER (TSC)	
See Section 18.17, "Time-Stamp Counter."		05_01H
Register Address: 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.	
Register Address: 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.	
Register Address: 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: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
20	LMCE On (R/WL) 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.	
Register Address: 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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 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
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.	
Register Address: 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.	
Register Address: 4EH, 78	IA32_PPIN_CTL	
Protected Processor Inventory Number Enable Control (R/W)		If CPUID.(EAX=07H, ECX=01H):EBX[0]=1[1]

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
0	LockOut (R/WO) 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.	
Register Address: 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'.	
Register Address: 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
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread.		
0	Reserved.	
1	KL Keylocker feature activation.	
63:2	Reserved.	
Register Address: 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.	
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.
Register Address: 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.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 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.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 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.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Configuration (R/W)		If CPUID.01H: ECX[5]=1 || CPUID.01H: ECX[6] = 1
0	Valid (R/W)	
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.	
Register Address: 9EH, 158	IA32_SMBASE	
Base address of the logical processor's SMRAM image (R/O, SMM only).		If IA32_VMX_MISC[15]
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTLS	
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.	
Register Address: BDH, 189	IA32_XAPIC_DISABLE_STATUS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: C1H, 193	IA32_PMC0 (PERFCTR0)	
General Performance Counter 0 (R/W)		If CPUID.0AH: EAX[15:8] > 0
Register Address: C2H, 194	IA32_PMC1 (PERFCTR1)	
General Performance Counter 1 (R/W)		If CPUID.0AH: EAX[15:8] > 1
Register Address: C3H, 195	IA32_PMC2	
General Performance Counter 2 (R/W)		If CPUID.0AH: EAX[15:8] > 2
Register Address: C4H, 196	IA32_PMC3	
General Performance Counter 3 (R/W)		If CPUID.0AH: EAX[15:8] > 3
Register Address: C5H, 197	IA32_PMC4	
General Performance Counter 4 (R/W)		If CPUID.0AH: EAX[15:8] > 4
Register Address: C6H, 198	IA32_PMC5	
General Performance Counter 5 (R/W)		If CPUID.0AH: EAX[15:8] > 5
Register Address: C7H, 199	IA32_PMC6	
General Performance Counter 6 (R/W)		If CPUID.0AH: EAX[15:8] > 6
Register Address: C8H, 200	IA32_PMC7	
General Performance Counter 7 (R/W)		If CPUID.0AH: EAX[15:8] > 7
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register		If CPUID.(EAX=07H, ECX=0):EDX[30] = 1
63:0	Reserved.	No architecturally defined bits.
Register Address: 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.	
Register Address: 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: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
63:0	C0_MCNT: C0 TSC Frequency Clock Count Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_APERF.	
Register Address: E8H, 232	IA32_APERF	
Actual Performance Clock Counter (R/Write to clear)		If CPUID.06H: ECX[0] = 1
63:0	C0_ACNT: C0 Actual Frequency Clock Count Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_MPERF.	
Register Address: 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.	
Register Address: 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).	
6	IF_PSCHANGE_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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
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	PBRSB_NO: If 1, the processor is not affected by issues related to Post-Barrier Return Stack Buffer Predictions.	
63:25	Reserved.	
Register Address: 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.	
Register Address: 10FH, 271	IA32_TSX_FORCE_ABORT	
TSX Force Abort		If CPUID.(EAX=07H, ECX=0):EDX[13]=1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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.	
Register Address: 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.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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
4	GDS_MITG_DIS If 0, the Gather Data Sampling mitigation is enabled (patch load time default). If 1 on all threads for a given core, the Gather Data Sampling mitigation is disabled.	
5	GDS_MITG_LOCK If 0, not locked, and GDS_MITG_DIS is under OS control. If 1, locked and GDS_MITG_DIS is forced to 0 (writes are ignored).	
63:6	Reserved.	
Register Address: 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.
63:32	Not used.	Writes ignored; reads return zero.
Register Address: 175H, 373	IA32_SYSENTER_ESP	
SYSENTER_ESP_MSR (R/W)		06_01H
Register Address: 176H, 374	IA32_SYSENTER_EIP	
SYSENTER_EIP_MSR (R/W)		06_01H
Register Address: 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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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
63:28	Reserved.	
Register Address: 17AH, 378	IA32_MCG_STATUS (MCG_STATUS)	
Global Machine Check Status (R/W)		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.	
Register Address: 17BH, 379	IA32_MCG_CTL (MCG_CTL)	
Global Machine Check Control (R/W)		If IA32_MCG_CAP.CTL_P[8] =1
Register Address: 180H–185H, 384–389	N/A	
Reserved		06_0EH[2]
Register Address: 186H, 390	IA32_PERFEVTSEL0 (PERFEVTSEL0)	
Performance Event Select Register 0 (R/W)		If CPUID.0AH: 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.	
Register Address: 187H, 391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Performance Event Select Register 1 (R/W)		If CPUID.0AH: EAX[15:8] > 1
Register Address: 188H, 392	IA32_PERFEVTSEL2	
Performance Event Select Register 2 (R/W)		If CPUID.0AH: EAX[15:8] > 2
Register Address: 189H, 393	IA32_PERFEVTSEL3	
Performance Event Select Register 3 (R/W)		If CPUID.0AH: EAX[15:8] > 3
Register Address: 18AH, 394	IA32_PERFEVTSEL4	
Performance Event Select Register 4 (R/W)		If CPUID.0AH: EAX[15:8] > 4
Register Address: 18BH, 395	IA32_PERFEVTSEL5	
Performance Event Select Register 5 (R/W)		If CPUID.0AH: EAX[15:8] > 5
Register Address: 18CH, 396	IA32_PERFEVTSEL6	
Performance Event Select Register 6 (R/W)		If CPUID.0AH: EAX[15:8] > 6
Register Address: 18DH, 397	IA32_PERFEVTSEL7	
Performance Event Select Register 7 (R/W)		If CPUID.0AH: EAX[15:8] > 7
Register Address: 18AH—194H, 394—404	N/A	
Reserved.		06_0EH[3]
Register Address: 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, 1indicates 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.	
Register Address: 196H—197H, 406—407	N/A	
Reserved.		06_0EH[3]
Register Address: 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.	
Register Address: 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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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.	
Register Address: 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.	
Register Address: 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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
3	PROCHOT # or FORCEPR# log (R/WC0)	If CPUID.01H:EDX[22] = 1
4	Critical Temperature Status (R/O)	If CPUID.01H:EDX[22] = 1
5	Critical Temperature Status log (R/WC0)	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/WC0)	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/WC0)	If CPUID.01H:ECX[8] = 1
10	Power Limitation Status (R/O)	If CPUID.06H:EAX[4] = 1
11	Power Limitation log (R/WC0)	If CPUID.06H:EAX[4] = 1
12	Current Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
13	Current Limit log (R/WC0)	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/WC0)	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.	
Register Address: 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.	0F_0H
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.	0F_0H
6:4	Reserved.	
7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.	0F_0H
10:8	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
22	Limit CPUID Maxval (R/W) When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 2. 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. Otherwise, this bit is not supported. Setting this bit when the maximum value is not greater than 2 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 2.	0F_03H
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.	If CPUID.01H:ECX[14] = 1
63:24	Reserved. Note: Some older processors defined one of these bits as a disable for the execute-disable feature of paging. If a processor supports this bit, this information is provided in the model-specific tables. See Table 2-3 for the definition of this bit.	
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Performance Energy Bias Hint (R/W)		If CPUID.6H:ECX[3] = 1
3:0	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
63:4	Reserved.	
Register Address: 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/WC0)	
4	Pkg Critical Temperature Status. (R/O)	
5	Pkg Critical Temperature Status Log. (R/WC0)	
6	Pkg Thermal Threshold #1 Status. (R/O)	
7	Pkg Thermal Threshold #1 Log. (R/WC0)	
8	Pkg Thermal Threshold #2 Status. (R/O)	
9	Pkg Thermal Threshold #1 Log. (R/WC0)	
10	Pkg Power Limitation Status. (R/O)	
11	Pkg Power Limitation Log. (R/WC0)	
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.	
Register Address: 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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
22:16	Pkg Threshold #2 Value.	
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.	
Register Address: 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
Register Address: 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
Register Address: 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
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)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
63:16	Reserved.	
Register Address: 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
Register Address: 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
Register Address: 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
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
Register Address: 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.	
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
SMRR Range Mask (Writeable only in SMM) Range Mask of SMM memory range.		If IA32_MTRRCAP[SMRR] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
10:0	Reserved.	
11	Valid Enable range mask.	
31:12	PhysMask SMRR address range mask.	
63:32	Reserved.	
Register Address: 1F8H, 504	IA32_PLATFORM_DCA_CAP	
DCA Capability (R)		If CPUID.01H: ECX[18] = 1
Register Address: 1F9H, 505	IA32_CPU_DCA_CAP	
If set, CPU supports Prefetch-Hint type.		If CPUID.01H: ECX[18] = 1
Register Address: 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.	
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.	
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	
See Section 12.11.2.3, "Variable Range MTRRs."		If IA32_MTRRCAP[7:0] > 0
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
MTRRphysMask0		If IA32_MTRRCAP[7:0] > 0
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
MTRRphysBase1		If IA32_MTRRCAP[7:0] > 1
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
MTRRphysMask1		If IA32_MTRRCAP[7:0] > 1
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
MTRRphysBase2		If IA32_MTRRCAP[7:0] > 2
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
MTRRphysMask2		If IA32_MTRRCAP[7:0] > 2
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
MTRRphysBase3		If IA32_MTRRCAP[7:0] > 3

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
MTRRphysMask3		If IA32_MTRRCAP[7:0] > 3
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
MTRRphysBase4		If IA32_MTRRCAP[7:0] > 4
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
MTRRphysMask4		If IA32_MTRRCAP[7:0] > 4
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
MTRRphysBase5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
MTRRphysMask5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
MTRRphysBase6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
MTRRphysMask6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
MTRRphysBase7		If IA32_MTRRCAP[7:0] > 7
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
MTRRphysMask7		If IA32_MTRRCAP[7:0] > 7
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
MTRRphysBase8		If IA32_MTRRCAP[7:0] > 8
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
MTRRphysMask8		If IA32_MTRRCAP[7:0] > 8
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
MTRRphysBase9		If IA32_MTRRCAP[7:0] > 9
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
MTRRphysMask9		If IA32_MTRRCAP[7:0] > 9
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
MTRRfix64K_00000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
MTRRfix16K_80000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
MTRRfix16K_A0000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	
See Section 12.11.2.2, “Fixed Range MTRRs.”		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
MTRRfix4K_C8000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
MTRRfix4K_D0000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
MTRRfix4K_D8000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
MTRRfix4K_E0000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
MTRRfix4K_E8000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
MTRRfix4K_F0000		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
MTRRfix4K_F8000.		If CPUID.01H: EDX.MTRR[12] =1
Register Address: 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.	
Register Address: 280H, 640	IA32_MC0_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.	
Register Address: 281H, 641	IA32_MC1_CTL2	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
Register Address: 282H, 642	IA32_MC2_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
Register Address: 283H, 643	IA32_MC3_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
Register Address: 284H, 644	IA32_MC4_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
Register Address: 285H, 645	IA32_MC5_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
Register Address: 286H, 646	IA32_MC6_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
Register Address: 287H, 647	IA32_MC7_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
Register Address: 288H, 648	IA32_MC8_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
Register Address: 289H, 649	IA32_MC9_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
Register Address: 28AH, 650	IA32_MC10_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
Register Address: 28BH, 651	IA32_MC11_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
Register Address: 28CH, 652	IA32_MC12_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
Register Address: 28DH, 653	IA32_MC13_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
Register Address: 28EH, 654	IA32_MC14_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
Register Address: 28FH, 655	IA32_MC15_CTL2	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
Register Address: 290H, 656	IA32_MC16_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
Register Address: 291H, 657	IA32_MC17_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
Register Address: 292H, 658	IA32_MC18_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
Register Address: 293H, 659	IA32_MC19_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
Register Address: 294H, 660	IA32_MC20_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
Register Address: 295H, 661	IA32_MC21_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
Register Address: 296H, 662	IA32_MC22_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
Register Address: 297H, 663	IA32_MC23_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
Register Address: 298H, 664	IA32_MC24_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
Register Address: 299H, 665	IA32_MC25_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
Register Address: 29AH, 666	IA32_MC26_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
Register Address: 29BH, 667	IA32_MC27_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
Register Address: 29CH, 668	IA32_MC28_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
Register Address: 29DH, 669	IA32_MC29_CTL2	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
Register Address: 29EH, 670	IA32_MC30_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
Register Address: 29FH, 671	IA32_MC31_CTL2	
Same fields as IA32_MC0_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31
Register Address: 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.	
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any.		If CPUID.0AH: EDX[4:0] > 0
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter 1 (R/W): Counts CPU_CLK_Unhalted.Core.		If CPUID.0AH: EDX[4:0] > 1
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter 2 (R/W): Counts CPU_CLK_Unhalted.Ref.		If CPUID.0AH: EDX[4:0] > 2
Register Address: 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.	
Register Address: 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	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0.	
1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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 1to count while CPL = 0.	
5	EN1_Usr: Enable Fixed Counter 1to 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.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
Global Performance Counter Status (R/O)		If CPUID.0AH: EAX[7:0] > 0 II (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
32+m	Ovf_FixedCtrm: Overflow status of IA32_FIXED_CTRm.	If CPUID.0AH:ECX[m] == 1 || CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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: ▪ 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: ▪ IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1. ▪ One or more core PMU counters overflowed.	If CPUID.0AH: EAX[7:0] > 3
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
Register Address: 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
32+m	EN_FIXED_CTRm	If CPUID.0AH:ECX[m] == 1 || CPUID.0AH:EDX[4:0] > m
47:33+m	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.	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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
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
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
Global Performance Counter Overflow Reset Control (R/W)		If CPUID.0AH: EAX[7:0] > 3 II (CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=14H, 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
32+m	Set 1 to Clear Ovf_FIXED_CTRm bit.	If CPUID.0AH:ECX[m] == 1 || CPUID.0AH:EDX[4:0] > m
47:33+m	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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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
58	Set 1 to Clear ASCI 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
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
Global Performance Counter Overflow Set Control (R/W)		If CPUID.0AH: EAX[7:0] > 3 II (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
32+m	Set 1 to cause Ovf_FIXED_CTRm = 1.	If CPUID.0AH:ECX[m] == 1 || CPUID.0AH:EDX[4:0] > m
47:33+m	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 ASCI = 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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
63	Reserved.	
Register Address: 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.	
1	IA32_PERFEVTSEL1 in use.	If CPUID.0AH: EAX[15:8] > 1
2	IA32_PERFEVTSEL2 in use.	If CPUID.0AH: EAX[15:8] > 2
n	IA32_PERFEVTSELn in use.	If CPUID.0AH: 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.	
Register Address: 3F1H, 1009	IA32_PEBS_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.	
Register Address: 400H, 1024	IA32_MC0_CTL	
MC0_CTL		If IA32_MCG_CAP.CNT >0
Register Address: 401H, 1025	IA32_MC0_STATUS	
MC0_STATUS		If IA32_MCG_CAP.CNT >0
Register Address: 402H, 1026	IA32_MC0_ADDR[1]	
MC0_ADDR		If IA32_MCG_CAP.CNT >0
Register Address: 403H, 1027	IA32_MC0_MISC	
MC0_MISC		If IA32_MCG_CAP.CNT >0
Register Address: 404H, 1028	IA32_MC1_CTL	
MC1_CTL		If IA32_MCG_CAP.CNT >1
Register Address: 405H, 1029	IA32_MC1_STATUS	
MC1_STATUS		If IA32_MCG_CAP.CNT >1
Register Address: 406H, 1030	IA32_MC1_ADDR[2]	
MC1_ADDR		If IA32_MCG_CAP.CNT >1
Register Address: 407H, 1031	IA32_MC1_MISC	
MC1_MISC		If IA32_MCG_CAP.CNT >1
Register Address: 408H, 1032	IA32_MC2_CTL	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
MC2_CTL		If IA32_MCG_CAP.CNT >2
Register Address: 409H, 1033	IA32_MC2_STATUS	
MC2_STATUS		If IA32_MCG_CAP.CNT >2
Register Address: 40AH, 1034	IA32_MC2_ADDR[1]	
MC2_ADDR		If IA32_MCG_CAP.CNT >2
Register Address: 40BH, 1035	IA32_MC2_MISC	
MC2_MISC		If IA32_MCG_CAP.CNT >2
Register Address: 40CH, 1036	IA32_MC3_CTL	
MC3_CTL		If IA32_MCG_CAP.CNT >3
Register Address: 40DH, 1037	IA32_MC3_STATUS	
MC3_STATUS		If IA32_MCG_CAP.CNT >3
Register Address: 40EH, 1038	IA32_MC3_ADDR[1]	
MC3_ADDR		If IA32_MCG_CAP.CNT >3
Register Address: 40FH, 1039	IA32_MC3_MISC	
MC3_MISC		If IA32_MCG_CAP.CNT >3
Register Address: 410H, 1040	IA32_MC4_CTL	
MC4_CTL		If IA32_MCG_CAP.CNT >4
Register Address: 411H, 1041	IA32_MC4_STATUS	
MC4_STATUS		If IA32_MCG_CAP.CNT >4
Register Address: 412H, 1042	IA32_MC4_ADDR[1]	
MC4_ADDR		If IA32_MCG_CAP.CNT >4
Register Address: 413H, 1043	IA32_MC4_MISC	
MC4_MISC		If IA32_MCG_CAP.CNT >4
Register Address: 414H, 1044	IA32_MC5_CTL	
MC5_CTL		If IA32_MCG_CAP.CNT >5
Register Address: 415H, 1045	IA32_MC5_STATUS	
MC5_STATUS		If IA32_MCG_CAP.CNT >5
Register Address: 416H, 1046	IA32_MC5_ADDR[1]	
MC5_ADDR		If IA32_MCG_CAP.CNT >5
Register Address: 417H, 1047	IA32_MC5_MISC	
MC5_MISC		If IA32_MCG_CAP.CNT >5
Register Address: 418H, 1048	IA32_MC6_CTL	
MC6_CTL		If IA32_MCG_CAP.CNT >6
Register Address: 419H, 1049	IA32_MC6_STATUS	
MC6_STATUS		If IA32_MCG_CAP.CNT >6
Register Address: 41AH, 1050	IA32_MC6_ADDR[1]	
MC6_ADDR		If IA32_MCG_CAP.CNT >6

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 41BH, 1051	IA32_MC6_MISC	
MC6_MISC		If IA32_MCG_CAP.CNT >6
Register Address: 41CH, 1052	IA32_MC7_CTL	
MC7_CTL		If IA32_MCG_CAP.CNT >7
Register Address: 41DH, 1053	IA32_MC7_STATUS	
MC7_STATUS		If IA32_MCG_CAP.CNT >7
Register Address: 41EH, 1054	IA32_MC7_ADDR[1]	
MC7_ADDR		If IA32_MCG_CAP.CNT >7
Register Address: 41FH, 1055	IA32_MC7_MISC	
MC7_MISC		If IA32_MCG_CAP.CNT >7
Register Address: 420H, 1056	IA32_MC8_CTL	
MC8_CTL		If IA32_MCG_CAP.CNT >8
Register Address: 421H, 1057	IA32_MC8_STATUS	
MC8_STATUS		If IA32_MCG_CAP.CNT >8
Register Address: 422H, 1058	IA32_MC8_ADDR[1]	
MC8_ADDR		If IA32_MCG_CAP.CNT >8
Register Address: 423H, 1059	IA32_MC8_MISC	
MC8_MISC		If IA32_MCG_CAP.CNT >8
Register Address: 424H, 1060	IA32_MC9_CTL	
MC9_CTL		If IA32_MCG_CAP.CNT >9
Register Address: 425H, 1061	IA32_MC9_STATUS	
MC9_STATUS		If IA32_MCG_CAP.CNT >9
Register Address: 426H, 1062	IA32_MC9_ADDR[1]	
MC9_ADDR		If IA32_MCG_CAP.CNT >9
Register Address: 427H, 1063	IA32_MC9_MISC	
MC9_MISC		If IA32_MCG_CAP.CNT >9
Register Address: 428H, 1064	IA32_MC10_CTL	
MC10_CTL		If IA32_MCG_CAP.CNT >10
Register Address: 429H, 1065	IA32_MC10_STATUS	
MC10_STATUS		If IA32_MCG_CAP.CNT >10
Register Address: 42AH, 1066	IA32_MC10_ADDR[1]	
MC10_ADDR		If IA32_MCG_CAP.CNT >10
Register Address: 42BH, 1067	IA32_MC10_MISC	
MC10_MISC		If IA32_MCG_CAP.CNT >10
Register Address: 42CH, 1068	IA32_MC11_CTL	
MC11_CTL		If IA32_MCG_CAP.CNT >11
Register Address: 42DH, 1069	IA32_MC11_STATUS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
MC11_STATUS		If IA32_MCG_CAP.CNT >11
Register Address: 42EH, 1070	IA32_MC11_ADDR[1]	
MC11_ADDR		If IA32_MCG_CAP.CNT >11
Register Address: 42FH, 1071	IA32_MC11_MISC	
MC11_MISC		If IA32_MCG_CAP.CNT >11
Register Address: 430H, 1072	IA32_MC12_CTL	
MC12_CTL		If IA32_MCG_CAP.CNT >12
Register Address: 431H, 1073	IA32_MC12_STATUS	
MC12_STATUS		If IA32_MCG_CAP.CNT >12
Register Address: 432H, 1074	IA32_MC12_ADDR[1]	
MC12_ADDR		If IA32_MCG_CAP.CNT >12
Register Address: 433H, 1075	IA32_MC12_MISC	
MC12_MISC		If IA32_MCG_CAP.CNT >12
Register Address: 434H, 1076	IA32_MC13_CTL	
MC13_CTL		If IA32_MCG_CAP.CNT >13
Register Address: 435H, 1077	IA32_MC13_STATUS	
MC13_STATUS		If IA32_MCG_CAP.CNT >13
Register Address: 436H, 1078	IA32_MC13_ADDR[1]	
MC13_ADDR		If IA32_MCG_CAP.CNT >13
Register Address: 437H, 1079	IA32_MC13_MISC	
MC13_MISC		If IA32_MCG_CAP.CNT >13
Register Address: 438H, 1080	IA32_MC14_CTL	
MC14_CTL		If IA32_MCG_CAP.CNT >14
Register Address: 439H, 1081	IA32_MC14_STATUS	
MC14_STATUS		If IA32_MCG_CAP.CNT >14
Register Address: 43AH, 1082	IA32_MC14_ADDR[1]	
MC14_ADDR		If IA32_MCG_CAP.CNT >14
Register Address: 43BH, 1083	IA32_MC14_MISC	
MC14_MISC		If IA32_MCG_CAP.CNT >14
Register Address: 43CH, 1084	IA32_MC15_CTL	
MC15_CTL		If IA32_MCG_CAP.CNT >15
Register Address: 43DH, 1085	IA32_MC15_STATUS	
MC15_STATUS		If IA32_MCG_CAP.CNT >15
Register Address: 43EH, 1086	IA32_MC15_ADDR[1]	
MC15_ADDR		If IA32_MCG_CAP.CNT >15
Register Address: 43FH, 1087	IA32_MC15_MISC	
MC15_MISC		If IA32_MCG_CAP.CNT >15

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 440H, 1088	IA32_MC16_CTL	
MC16_CTL		If IA32_MCG_CAP.CNT >16
Register Address: 441H, 1089	IA32_MC16_STATUS	
MC16_STATUS		If IA32_MCG_CAP.CNT >16
Register Address: 442H, 1090	IA32_MC16_ADDR[1]	
MC16_ADDR		If IA32_MCG_CAP.CNT >16
Register Address: 443H, 1091	IA32_MC16_MISC	
MC16_MISC		If IA32_MCG_CAP.CNT >16
Register Address: 444H, 1092	IA32_MC17_CTL	
MC17_CTL		If IA32_MCG_CAP.CNT >17
Register Address: 445H, 1093	IA32_MC17_STATUS	
MC17_STATUS		If IA32_MCG_CAP.CNT >17
Register Address: 446H, 1094	IA32_MC17_ADDR[1]	
MC17_ADDR		If IA32_MCG_CAP.CNT >17
Register Address: 447H, 1095	IA32_MC17_MISC	
MC17_MISC		If IA32_MCG_CAP.CNT >17
Register Address: 448H, 1096	IA32_MC18_CTL	
MC18_CTL		If IA32_MCG_CAP.CNT >18
Register Address: 449H, 1097	IA32_MC18_STATUS	
MC18_STATUS		If IA32_MCG_CAP.CNT >18
Register Address: 44AH, 1098	IA32_MC18_ADDR[1]	
MC18_ADDR		If IA32_MCG_CAP.CNT >18
Register Address: 44BH, 1099	IA32_MC18_MISC	
MC18_MISC		If IA32_MCG_CAP.CNT >18
Register Address: 44CH, 1100	IA32_MC19_CTL	
MC19_CTL		If IA32_MCG_CAP.CNT >19
Register Address: 44DH, 1101	IA32_MC19_STATUS	
MC19_STATUS		If IA32_MCG_CAP.CNT >19
Register Address: 44EH, 1102	IA32_MC19_ADDR[1]	
MC19_ADDR		If IA32_MCG_CAP.CNT >19
Register Address: 44FH, 1103	IA32_MC19_MISC	
MC19_MISC		If IA32_MCG_CAP.CNT >19
Register Address: 450H, 1104	IA32_MC20_CTL	
MC20_CTL		If IA32_MCG_CAP.CNT >20
Register Address: 451H, 1105	IA32_MC20_STATUS	
MC20_STATUS		If IA32_MCG_CAP.CNT >20
Register Address: 452H, 11061106	IA32_MC20_ADDR[1]	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
MC20_ADDR		If IA32_MCG_CAP.CNT >20
Register Address: 453H, 1107	IA32_MC20_MISC	
MC20_MISC		If IA32_MCG_CAP.CNT >20
Register Address: 454H, 1108	IA32_MC21_CTL	
MC21_CTL		If IA32_MCG_CAP.CNT >21
Register Address: 455H, 1109	IA32_MC21_STATUS	
MC21_STATUS		If IA32_MCG_CAP.CNT >21
Register Address: 456H, 1110	IA32_MC21_ADDR[1]	
MC21_ADDR		If IA32_MCG_CAP.CNT >21
Register Address: 457H, 1111	IA32_MC21_MISC	
MC21_MISC		If IA32_MCG_CAP.CNT >21
Register Address: 458H, 1112	IA32_MC22_CTL	
MC22_CTL		If IA32_MCG_CAP.CNT >22
Register Address: 459H, 1113	IA32_MC22_STATUS	
MC22_STATUS		If IA32_MCG_CAP.CNT >22
Register Address: 45AH, 1114	IA32_MC22_ADDR[1]	
MC22_ADDR		If IA32_MCG_CAP.CNT >22
Register Address: 45BH, 1115	IA32_MC22_MISC	
MC22_MISC		If IA32_MCG_CAP.CNT >22
Register Address: 45CH, 1116	IA32_MC23_CTL	
MC23_CTL		If IA32_MCG_CAP.CNT >23
Register Address: 45DH, 1117	IA32_MC23_STATUS	
MC23_STATUS		If IA32_MCG_CAP.CNT >23
Register Address: 45EH, 1118	IA32_MC23_ADDR[1]	
MC23_ADDR		If IA32_MCG_CAP.CNT >23
Register Address: 45FH, 1119	IA32_MC23_MISC	
MC23_MISC		If IA32_MCG_CAP.CNT >23
Register Address: 460H, 1120	IA32_MC24_CTL	
MC24_CTL		If IA32_MCG_CAP.CNT >24
Register Address: 461H, 1121	IA32_MC24_STATUS	
MC24_STATUS		If IA32_MCG_CAP.CNT >24
Register Address: 462H, 1122	IA32_MC24_ADDR[1]	
MC24_ADDR		If IA32_MCG_CAP.CNT >24
Register Address: 463H, 1123	IA32_MC24_MISC	
MC24_MISC		If IA32_MCG_CAP.CNT >24
Register Address: 464H, 1124	IA32_MC25_CTL	
MC25_CTL		If IA32_MCG_CAP.CNT >25

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 465H, 1125	IA32_MC25_STATUS	
MC25_STATUS		If IA32_MCG_CAP.CNT >25
Register Address: 466H, 1126	IA32_MC25_ADDR[1]	
MC25_ADDR		If IA32_MCG_CAP.CNT >25
Register Address: 467H, 1127	IA32_MC25_MISC	
MC25_MISC		If IA32_MCG_CAP.CNT >25
Register Address: 468H, 1128	IA32_MC26_CTL	
MC26_CTL		If IA32_MCG_CAP.CNT >26
Register Address: 469H, 1129	IA32_MC26_STATUS	
MC26_STATUS		If IA32_MCG_CAP.CNT >26
Register Address: 46AH, 1130	IA32_MC26_ADDR[1]	
MC26_ADDR		If IA32_MCG_CAP.CNT >26
Register Address: 46BH, 1131	IA32_MC26_MISC	
MC26_MISC		If IA32_MCG_CAP.CNT >26
Register Address: 46CH, 1132	IA32_MC27_CTL	
MC27_CTL		If IA32_MCG_CAP.CNT >27
Register Address: 46DH, 1133	IA32_MC27_STATUS	
MC27_STATUS		If IA32_MCG_CAP.CNT >27
Register Address: 46EH, 1134	IA32_MC27_ADDR[1]	
MC27_ADDR		If IA32_MCG_CAP.CNT >27
Register Address: 46FH, 1135	IA32_MC27_MISC	
MC27_MISC		If IA32_MCG_CAP.CNT >27
Register Address: 470H, 1136	IA32_MC28_CTL	
MC28_CTL		If IA32_MCG_CAP.CNT >28
Register Address: 471H, 1137	IA32_MC28_STATUS	
MC28_STATUS		If IA32_MCG_CAP.CNT >28
Register Address: 472H, 1138	IA32_MC28_ADDR[1]	
MC28_ADDR		If IA32_MCG_CAP.CNT >28
Register Address: 473H, 1139	IA32_MC28_MISC	
MC28_MISC		If IA32_MCG_CAP.CNT >28
Register Address: 474H, 1140	IA32_MC29_CTL	
MC29_CTL		If IA32_MCG_CAP.CNT >29
Register Address: 475H, 1141	IA32_MC29_STATUS	
MC29_STATUS		If IA32_MCG_CAP.CNT >29
Register Address: 476H, 1142	IA32_MC29_ADDR	
MC29_ADDR		If IA32_MCG_CAP.CNT >29
Register Address: 477H, 1143	IA32_MC29_MISC	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
MC29_MISC		If IA32_MCG_CAP.CNT >29
Register Address: 478H, 1144	IA32_MC30_CTL	
MC30_CTL		If IA32_MCG_CAP.CNT >30
Register Address: 479H, 1145	IA32_MC30_STATUS	
MC30_STATUS		If IA32_MCG_CAP.CNT >30
Register Address: 47AH, 1146	IA32_MC30_ADDR	
MC30_ADDR		If IA32_MCG_CAP.CNT >30
Register Address: 47BH, 1147	IA32_MC30_MISC	
MC30_MISC		If IA32_MCG_CAP.CNT >30
Register Address: 47CH, 1148	IA32_MC31_CTL	
MC31_CTL		If IA32_MCG_CAP.CNT >31
Register Address: 47DH, 1149	IA32_MC31_STATUS	
MC31_STATUS		If IA32_MCG_CAP.CNT >31
Register Address: 47EH, 1150	IA32_MC31_ADDR	
MC31_ADDR		If IA32_MCG_CAP.CNT >31
Register Address: 47FH, 1151	IA32_MC31_MISC	
MC31_MISC		If IA32_MCG_CAP.CNT >31
Register Address: 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
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
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
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
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
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
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
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls."		If CPUID.01H:ECX.[5] = 1
Register Address: 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
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX.[5] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX.[5] = 1
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
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_CTLS[63])
Register Address: 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_CTLS[63] && (IA32_VMX_PROCBASED_CTLS2[33] || IA32_VMX_PROCBASED_CTLS2[37]))
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	
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])
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLS	
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])
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLS	
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])
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLS	
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])
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O)		If(CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTLS[63] && IA32_VMX_PROCBASED_CTLS2[45])

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTLS3	
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_CTLS[49])
Register Address: 493H, 1171	IA32_VMX_EXIT_CTLS2	
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_CTLS[63])
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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.	
Register Address: 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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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.	
Register Address: 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.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
0	TraceEn	
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)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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)
56	InjectPsbPmiOnEnable	If (CPUID.(EAX=07H, ECX=1):EBX[6] = 1)
63:57	Reserved, must be zero.	
Register Address: 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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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.	
Register Address: 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	
Register Address: 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	
Register Address: 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	
Register Address: 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	
Register Address: 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	
Register Address: 585H, 1413	IA32_RTIT_ADDR2_B	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Region 2 End Address (R/W)		If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 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	
Register Address: 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	
Register Address: 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.	
Register Address: 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.	
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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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 used.	
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W)		See IA32_U_CET (6A0H) for reference; similar format.
Register Address: 6A4H, 1700	IA32_PL0_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. Bits 1:0 of the MSR must be 0. Transitions to privilege level 0 will check that bit 2 is also 0.		If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
Register Address: 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. Bits 1:0 of the MSR must be 0. Transitions to privilege level 1 from a higher privilege level will check that bit 2 is also 0.		If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
Register Address: 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. Bits 1:0 of the MSR must be 0. Transitions to privilege level 2 from a higher privilege level will check that bit 2 is also 0.		If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
Register Address: 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. Bits 1:0 of the MSR must be 0.		If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
Register Address: 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
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
TSC Target of Local APIC's TSC Deadline Mode (R/W)		If CPUID.01H:ECX.[24] = 1
Register Address: 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.	
Register Address: 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.	
Register Address: 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.	
Register Address: 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.	
Register Address: 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: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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.	
Register Address: 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.	
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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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.	
Register Address: 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/WC0) See Section 15.4.5, "HWP Feedback."	If CPUID.06H:EAX.[7] = 1
1	Reserved.	
2	Excursion_To_Minimum (R/WC0) See Section 15.4.5, "HWP Feedback."	If CPUID.06H:EAX.[7] = 1
63:3	Reserved.	
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits 31:0 (R/O)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 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)
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 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
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 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
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 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
Register Address: 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
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 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.	
2	Support for AES-XTS 256-bit encryption algorithm.	
29:3	Reserved.	
30	SUPPORT_IA32_TME_CLEAR_SAVED_KEY Support for the IA32_TME_CLEAR_SAVED_KEY MSR.	
31	TME encryption bypass supported.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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 TME-MK 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.	
Register Address: 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. ▪ 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 TME-MK; TME-MK 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).	
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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
31	TME Encryption Bypass Enable When encryption hardware is enabled: ▪ 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 TME-MK is not enumerated, otherwise: The number of key identifier bits to allocate to TME-MK 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 TME-MK. Example: To support 255 keys, this field would be set to a value of 8.	
47:36	Reserved.	
63:48	MK_TME_CRYPTO_ALGS Reserved if TME-MK 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 TME-MK, would be later enforced by the key loading ISA ('1 = allowed).	
Register Address: 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/TME-MK (for KeyID=0).	
MAXPHYADDR-1:12	TMEEMASK: This field indicates the bits that must match TMEEBASE in order to qualify as a TME/TME-MK (for KeyID=0) exclusion memory range access.	
63:MAXPHYADDR	Reserved; must be zero.	
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
Memory Encryption Exclude Base		IF CPUID.07H:ECX.[13] = 1
11:0	Reserved.	
MAXPHYADDR-1:12	TMEEBASE: Base physical address to be excluded for TME/TME-MK (for KeyID=0) encryption.	
63:MAXPHYADDR	Reserved; must be zero.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 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.	
Register Address: 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.	
Register 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.	
63:3	STACK_ADJUST Stack adjust value.	
Register Address: 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.	
Register Address: 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.	
Register 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.	
Register 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))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS[5]	
Information about IWKeyBackup Register (R/O)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] =1))
0	Backup/Restore Valid Cleared when a write to IWKeyBackup is initiated, and then set when the latest write of IWKeyBackup has been written to storage that persists across S3/S4 sleep state. If S3/S4 is entered between when an IWKeyBackup write occurs and when this bit is set, then IWKeyBackup 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 IWKeyBackup is restored from persistent storage and thus available to be copied to IWKey using IA32_COPY_PLATFORM_TO_LOCAL MSR. Another write to IWKeyBackup (via IA32_COPY_LOCAL_TO_PLATFORM 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	IWKeyBackup 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 IWKeyBackup write as the previous IWKeyBackup write may still need to set Backup/restore valid.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] =1))
63:4	Reserved.	
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (W/O)		
0	TME_CLEAR_SAVED_KEY Clear saved TME keys.	
63:1	Reserved.	
Register Address: 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.	
Register Address: C81H, 3201	IA32_L3_QOS_CFG	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
L3 QOS Configuration (R/W)		If (CPUID.(EAX=10H, ECX=1):ECX.[2] = 1)
0	Enable (R/W) Set 1 to enable L3 CAT masks and CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: 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 CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: C83H, 3203	IA32_L3_IO_QOS_CFG	
L3 I/O QOS Configuration (R/W) This MSR is used to enable the I/O RDT features.		If (CPUID.(EAX=0FH, ECX=1):EAX.[10:9] = 1)
0	L3 I/O RDT Allocation Enable.	
1	L3 I/O RDT Monitoring Enable.	
63:2	Reserved.	
Register Address: 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.	
Register Address: 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.	
Register Address: 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))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
31:N	Reserved.	
63:32	CLOS (R/W): The class of service (CLOS) to enforce (on writes); returns the current CLOS when read.	If (CPUID.(EAX=07H, ECX=0):EBX.[15] = 1)
Register Address: 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."		
Register Address: 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.	
Register Address: 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.	
Register Address: 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."		
Register Address: 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.	
Register Address: 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.	
Register Address: 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.	
Register Address: 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: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
0	IWKeyBackup Copy IWKey to IWKeyBackup.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
63:1	Reserved.	
Register Address: 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	IWKeyBackup Copy IWKeyBackup to IWKey.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
63:1	Reserved.	
Register Address: 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.	
Register Address: 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.	
Register Address: 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.	
Register Address: 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: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: DB2H, 3506	IA32_THREAD_STALL	
Per-Logical_Processor_ID HDC Idle Residency (R/0)		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
Register Address: 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
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
Register Address: 1406H, 5126	IA32_MCU_CONTROL	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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
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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 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
Register Address: 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 ≥ 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
Register Address: 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 ≥ 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
Register Address: 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.	
Register Address: 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.	
Register Address: 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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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.	
Register Address: 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.	
Register Address: 17DAH, 6106	IA32_HRESET_ENABLE	
History Reset Enable (R/W)		
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.	
Register Address: 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.	
Register Address: 4000_0000H—4000_00FFH	Reserved MSR Address Space	
All existing and future processors will not implement MSRs in this range.		
Register Address: 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.	
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W)		If CPUID.80000001:EDX.[29] = 1
Register Address: C000_0082H	IA32_LSTAR	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal	Architectural MSR Name (Former MSR Name)	
Bit Fields	**MSR/Bit Description**	**Comment**
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
Register Address: 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
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W)		If CPUID.80000001:EDX.[29] = 1
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W)		If CPUID.80000001:EDX.[29] = 1
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W)		If CPUID.80000001:EDX.[29] = 1
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W)		If CPUID.80000001:EDX.[29] = 1
Register Address: 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_MC*i*_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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Unique
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Unique
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 18.17, "Time-Stamp Counter," and Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
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.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
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.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
4	Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
5	Reserved.	
6	Reserved.	
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)	
Register Address: 3AH, 58	MSR_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Unique
3	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.	Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ 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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.5.		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
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.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: A0H, 160	MSR_SMRR_PHYSBASE	
System Management Mode Base Address register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
11:0	Reserved.	
31:12	PhysBase: SMRR physical Base Address.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
63:32	Reserved.	
Register Address: A1H, 161	MSR_SMRR_PHYSMASK	
System Management Mode Physical Address Mask register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
10:0	Reserved.	
11	Valid: Physical address base and range mask are valid.	
31:12	PhysMask: SMRR physical address range mask.	
63:32	Reserved.	
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Core microarchitecture.		Shared
2:0	▪ 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)	
	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. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.	
63:3	Reserved.	
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Enhanced Intel Core microarchitecture.		Shared

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
2:0	▪ 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) 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 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Unique
11	SMRR Capability Using MSR 0A0H and 0A1H (R)	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
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.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
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.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Current performance status. See Section 15.1.1, "Software Interface For Initiating Performance State Transitions."		Shared
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.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
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.	
Register Address: 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	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
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.	
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.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared
13	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.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	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.	Shared
20	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: ▪ 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.	Shared
21	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Shared
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
34	XD Bit Disable (R/W) 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). 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. 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.	Unique
36:35	Reserved.	
37	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.	Unique
38	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.	Shared
39	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.	Unique
63:40	Reserved.	
Register Address: 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 MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ 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.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
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.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Unique
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Unique
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Unique
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Unique
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Unique
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Unique
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Unique
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Unique
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Unique
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Unique
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Unique
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Unique
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Unique
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Unique
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Unique
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Unique
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Unique
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Unique
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Unique
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Unique
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Unique
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Unique
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Unique
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Unique
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Unique
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Unique
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Unique
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."		Unique
Register Address: 345H, 837	MSR_PERF_CAPABILITIES	
R/O. This applies to processors that do not support architectural perfmon version 2.		Unique
5:0	LBR Format. See Table 2-2.	
6	PEBS Record Format.	
7	PEBSSaveArchRegs. See Table 2-2.	
63:8	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	MSR_PERF_GLOBAL_CTRL	
See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0. (R/W)	
Register Address: 400H, 1024	IA32_MC0_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 402H, 1026	IA32_MC0_ADDR	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 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 ADDRV 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.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 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 ADDRV 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.		Unique
Register Address: 40CH, 1036	IA32_MC4_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 40DH, 1037	IA32_MC4_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 40EH, 1038	IA32_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 ADDRV 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.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		
Register Address: 411H, 1041	IA32_MC3_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		
Register Address: 412H, 1042	IA32_MC3_ADDR	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ 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 ADDRV 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.		Unique
Register Address: 413H, 1043	IA32_MC3_MISC	
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.		Unique
Register Address: 414H, 1044	IA32_MC5_CTL	
Machine Check Error Reporting Register: Controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	IA32_MC5_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.		Unique
Register Address: 416H, 1046	IA32_MC5_ADDR	
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.		Unique
Register Address: 417H, 1047	IA32_MC5_MISC	
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.		Unique
Register Address: 419H, 1045	IA32_MC6_STATUS	
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.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: 107CCH, 67532	MSR_EMON_L3_CTR_CTL0	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: 107CDH, 67533	MSR_EMON_L3_CTR_CTL1	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: 107CEH, 67534	MSR_EMON_L3_CTR_CTL2	
GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: 107CFH, 67535	MSR_EMON_L3_CTR_CTL3	
GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: 107D0H, 67536	MSR_EMON_L3_CTR_CTL4	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ 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.		Unique
Register Address: 107D1H, 67537	MSR_EMON_L3_CTR_CTL5	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: 107D2H, 67538	MSR_EMON_L3_CTR_CTL6	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: 107D3H, 67539	MSR_EMON_L3_CTR_CTL7	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: 107D8H, 67544	MSR_EMON_L3_GL_CTL	
L3/FSB Common Control Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.2.2.		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 18.17, "Time-Stamp Counter," and see Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
7:0	Reserved.	
12:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
63:13	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
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.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
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.	
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)	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.5.		Unique

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
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.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: C1H, 193	IA32_PMC0	
Performance counter register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Atom microarchitecture.		Shared
2:0	▪ 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.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Memory Type Range Register (R) See Table 2-2.		Shared
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Shared
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.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
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.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Shared
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.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Shared
15:0	Reserved.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
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.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Unique
0	Fast-Strings Enable See Table 2-2.	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
8	Reserved.	
9	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.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared
13	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.	Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
20	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: ▪ 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.	Shared
21	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Unique
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Unique
63:35	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
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).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Unique
Register Address: 1DDH, 477	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.		Unique
Register Address: 1DEH, 478	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.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
See Table 2-2.		Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Shared
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Shared
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Shared
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."		Shared
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0 (R/W)	
Register Address: 400H, 1024	IA32_MC0_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Shared
Register Address: 402H, 1026	IA32_MC0_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Shared
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Shared
Register Address: 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 ADDRV 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.		Shared
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Shared
Register Address: 40EH, 1038	IA32_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 ADDRV 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.		Shared
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Shared
Register Address: 412H, 1042	IA32_MC4_ADDR	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ 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 ADDRV 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.		Shared
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 20.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3F8H, 1016	MSR_PKG_C2_RESIDENCY	
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.		Package

Table 2-5. MSRs Supported by Intel Atom® Processors (Contd.)with a CPUID Signature DisplayFamily_DisplayModel Value of 06_27H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:0	Package C2 Residency Counter (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3F9H, 1017	MSR_PKG_C4_RESIDENCY	
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.		Package
63:0	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
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.		Package
63:0	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.	Package

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 Intel Atom® Processors (Silvermont and Newer Microarchitectures)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 1H, 1	IA32_P5_MC_TYPE	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Core
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 18.17, "Time-Stamp Counter," and Table 2-2.		Core
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.		Core
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Writes ignored.		Module
63:0	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Core
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Core
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Core
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Core
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org.		Module
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 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
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.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Core
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Core
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Core
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Core
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Core
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.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Module
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Core
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Core
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
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.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Core
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Core
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
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.		Core
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
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.		Core
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Core
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Core
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Core
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."		Core
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Core
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 400H, 1024	IA32_MC0_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 402H, 1026	IA32_MC0_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 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 ADDRV 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.		Module
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40EH, 1038	IA32_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 ADDRV 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.		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 412H, 1042	IA32_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 ADDRV 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.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Package
Register Address: 416H, 1046	IA32_MC5_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 ADDRV 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.		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Core
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Core
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Core
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Core
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Core
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Core
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.		Core
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLS	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.		Core
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.		Core
Register Address: 491H, 1169	IA32_VMX_FMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Core
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 20.6.3.4, "Debug Store (DS) Mechanism."		Core
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Core
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Core
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Core
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Core
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Core
Register Address: C000_0103H	IA32_TSC_AUX	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
AUXILIARY TSC Signature (R/W) See Table 2-2.		Core

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
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.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation (R/WL)	
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.5 and record format in Section 18.4.8.1.		Core
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
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.		Core
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information: Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.	Package
63:16	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Module
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-sate 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.	
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Module
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.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 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.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Module
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Module
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Module
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
38	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 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 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.	Module
63:39	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."		Core
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.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
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.		Core
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS for precise event on IA32_PMC0 (R/W)	
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
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.		Module
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Silvermont microarchitecture.		Module
2:0	▪ 100B: 080.0 MHz ▪ 000B: 083.3 MHz ▪ 001B: 100.0 MHz ▪ 010B: 133.3 MHz ▪ 011B: 116.7 MHz	
63:3	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
	Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."	Package
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.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W)		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 668H, 1640	MSR_CC6_DEMOTION_POLICY_CONFIG	
Core C6 Demotion Policy Config MSR		Package
63:0	Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.	
Register Address: 669H, 1641	MSR_MC6_DEMOTION_POLICY_CONFIG	
Module C6 Demotion Policy Config MSR		Package
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.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
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.		Module
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	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.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode (R/W)		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."		Package
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.	

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 66EH, 1646	MSR_PKG_POWER_INFO	
PKG RAPL Parameter (R/0)		Package
14:0	Thermal Spec Power (R/0) 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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Airmont microarchitecture.		Module
3:0	▪ 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.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Module
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.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org.		Module
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 - Deep Power Down Technology is the max C-State. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: 638H, 1592	MSR_PP0_POWER_LIMIT	
PP0 RAPL Power Limit Control (R/W)		Package

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
14:0	PP0 Power Limit #1 (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.	
15	Enable Power Limit #1 (R/W) See Section 15.10.4, "PP0/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 PP0_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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
49:0	Reserved.	
52:50	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:33	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Core
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.	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Core TSC ADJUST (R/W) See Table 2-2.		Core
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Core
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	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.	Package
29	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.	Package
30	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.	Package
39:31	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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.	
63:16	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability enhancement. Accessible only while in SMM.		Core
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.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Core
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Core
Register Address: 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.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Package
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Package
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
38	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.	Package
63:39	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	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.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org.		Package
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.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
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.		Package
7:0	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.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
15:8	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.	Package
23:16	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.	Package
31:24	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.	Package
39:32	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.	Package
47:40	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.	Package
55:48	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.	Package
63:56	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.	Package
Register Address: 1AEH, 430	MSR_TURBO_GROUP_CORECNT	
Group Size of Active Cores for Turbo Mode Operation (R/W) Writes of 0 threshold is ignored.		Package
7:0	Group 0 Core Count Threshold Maximum number of active cores to operate under the Group 0 Max Turbo Ratio limit.	Package
15:8	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.	Package
23:16	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.	Package
31:24	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.	Package
39:32	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.	Package
47:40	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.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
55:48	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.	Package
63:56	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.	Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."		Core
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	
63:10	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
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.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org.		Core
0	Reserved.	
1	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).	Package
63:2	Reserved.	
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Core
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Core
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address:	IA32_MTRR_PHYSMASK9	
213H, 531	See Table 2-2.	Core
Register Address:	IA32_MC0_CTL2	
280H, 640	See Table 2-2.	Module
Register Address:	IA32_MC1_CTL2	
281H, 641	See Table 2-2.	Module
Register Address:	IA32_MC2_CTL2	
282H, 642	See Table 2-2.	Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Module
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 300H, 768	MSR_SGXOWNEREPOCH0	
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.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNEREPOCH1	
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.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 20.2.4, "Architectural Performance Monitoring Version 4."		Core
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	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
57:56	Reserved.	
58	LBR_Frz	
59	CTR_Frz	
60	ASCI	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 20.2.4, "Architectural Performance Monitoring Version 4."		Core
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.	
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 20.2.4, "Architectural Performance Monitoring Version 4."		Core
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.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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 ASCI = 1.	
61	Set 1 to cause Ovf_Uncore.	
62	Set 1 to cause Ovf_BufDSSAVE.	
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2 and Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 406H, 1030	IA32_MC1_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 ADDRV 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.		Module
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Core
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Core
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package
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.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
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.		Package
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.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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 0FFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Core
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.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Core
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Core
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
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	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
18	Reserved, must be zero.	
22:19	CycThresh	
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Core
0	FilterEn Writes ignored.	
1	ContextEn 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.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Core
4:0	Reserved	
63:5	CR3[63:5] value to match.	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Core
63:0	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."		Package
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.	
Register Address: 60AH, 1546	MSR_PKGC3_IRTL	
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.		Package
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-sate management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKGC_IRTL1	
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.		Package
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.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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-sate management.	
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKGC_IRTL2	
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.		Package
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-sate management.	
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
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.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W)		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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) * 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.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
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.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
ConfigTDP Control (R/W)		Package
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.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
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.	
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.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.6 and record format in Section 18.4.8.1.		Core
0:47	From Linear Address (R/W)	
62:48	Signed extension of bits 47:0.	
63	Mispred	
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
Last Branch Record 16 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	
Last Branch Record 19From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
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.		Core
0:47	Target Linear Address (R/W)	
63:48	Elapsed cycles from last update to the LBR.	
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D0H, 1744	MSR_LASTBRANCH_16_TO_IP	
Last Branch Record 16 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID register (R/O)		Core
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version register (R/O)		Core
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority register (R/W)		Core
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority register (R/O)		Core
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI register (W/O)		Core
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination register (R/O)		Core
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector register (R/W)		Core
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service register bits [31:0] (R/O)		Core
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service register bits [63:32] (R/O)		Core
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service register bits [95:64] (R/O)		Core
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service register bits [127:96] (R/O)		Core
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service register bits [159:128] (R/O)		Core
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service register bits [191:160] (R/O)		Core
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service register bits [223:192] (R/O)		Core
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service register bits [255:224] (R/O)		Core
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode register bits [31:0] (R/O)		Core
Register Address: 819H, 2073	IA32_X2APIC_TMR1	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
x2APIC Trigger Mode register bits [63:32] (R/O)		Core
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode register bits [95:64] (R/O)		Core
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode register bits [127:96] (R/O)		Core
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode register bits [159:128] (R/O)		Core
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode register bits [191:160] (R/O)		Core
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode register bits [223:192] (R/O)		Core
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode register bits [255:224] (R/O)		Core
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request register bits [31:0] (R/O)		Core
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request register bits [63:32] (R/O)		Core
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request register bits [95:64] (R/O)		Core
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request register bits [127:96] (R/O)		Core
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request register bits [159:128] (R/O)		Core
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request register bits [191:160] (R/O)		Core
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request register bits [223:192] (R/O)		Core
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request register bits [255:224] (R/O)		Core
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status register (R/W)		Core
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt register (R/W)		Core
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command register (R/W)		Core
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt register (R/W)		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt register (R/W)		Core
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor register (R/W)		Core
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 register (R/W)		Core
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 register (R/W)		Core
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error register (R/W)		Core
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count register (R/W)		Core
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count register (R/O)		Core
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration register (R/W)		Core
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI register (W/O)		Core
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Core
31:0	Reserved.	
33:32	CLOS (R/W)	
63: 34	Reserved.	
Register Address: D10H, 3344	IA32_L2_QOS_MASK_0	
L2 Class Of Service Mask - CLOS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D11H, 3345	IA32_L2_QOS_MASK_1	
L2 Class Of Service Mask - CLOS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D12H, 3346	IA32_L2_QOS_MASK_2	
L2 Class Of Service Mask - CLOS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: D13H, 3347	IA32_L2_QOS_MASK_3	
L2 Class Of Service Mask - CLOS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.		Package
0:19	CBM: Bit vector of available L2 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Core
Register Address: DA0H, 3488	IA32_XSS	
See Table 2-2.		Core
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 supersede 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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
18	SGX global functions enable (R/WL)	
63:19	Reserved.	
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Core
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Core
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Core
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
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.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
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	
10	TSCEn	

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
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: ▪ 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."		Core
Register Address: 681H–69FH, 1665–1695	MSR_LASTBRANCH_*i*_FROM_IP	
Last Branch Record *i* From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP; *i* = 1-31.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
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: ▪ Section 18.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."		Core
Register Address: 6C1H–6DFH, 1729–1759	MSR_LASTBRANCH_*i*_TO_IP	
Last Branch Record *i* To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP; *i* = 1-31.		Core
Register Address: DC0H, 3520	MSR_LASTBRANCH_INFO_0	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.9.1, "LBR Stack."		Core
Register Address: DC1H, 3521	MSR_LASTBRANCH_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: DC2H, 3522	MSR_LASTBRANCH_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC3H, 3523	MSR_LASTBRANCH_INFO_3	
Last Branch Record 3 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC4H, 3524	MSR_LASTBRANCH_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC5H, 3525	MSR_LASTBRANCH_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC6H, 3526	MSR_LASTBRANCH_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC7H, 3527	MSR_LASTBRANCH_INFO_7	
Last Branch Record 7 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC8H, 3528	MSR_LASTBRANCH_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC9H, 3529	MSR_LASTBRANCH_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCAH, 3530	MSR_LASTBRANCH_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCBH, 3531	MSR_LASTBRANCH_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCCH, 3532	MSR_LASTBRANCH_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCDH, 3533	MSR_LASTBRANCH_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCEH, 3534	MSR_LASTBRANCH_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: DCFH, 3535	MSR_LASTBRANCH_INFO_15	
Last Branch Record 15 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD0H, 3536	MSR_LASTBRANCH_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD1H, 3537	MSR_LASTBRANCH_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD2H, 3538	MSR_LASTBRANCH_INFO_18	
Last Branch Record 18 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD3H, 3539	MSR_LASTBRANCH_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD4H, 3520	MSR_LASTBRANCH_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD5H, 3521	MSR_LASTBRANCH_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD6H, 3522	MSR_LASTBRANCH_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD7H, 3523	MSR_LASTBRANCH_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD8H, 3524	MSR_LASTBRANCH_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD9H, 3525	MSR_LASTBRANCH_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDAH, 3526	MSR_LASTBRANCH_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDBH, 3527	MSR_LASTBRANCH_INFO_27	
Last Branch Record 27 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: DDCH, 3528	MSR_LASTBRANCH_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDDH, 3529	MSR_LASTBRANCH_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDEH, 3530	MSR_LASTBRANCH_INFO_30	
Last Branch Record 30 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDFH, 3531	MSR_LASTBRANCH_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
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 supersede 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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
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.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
IA32 Core Capabilities Register If CPUID.(EAX=07H, ECX=0):EDX[30] = 1.		Core
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.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 20.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
n:0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMCx. The maximum value n can be determined from CPUID.0AH:EAX[15:8].	
31:*n*+1	Reserved.	
32+*m*:32	Enable PEBS trigger and recording for IA32_FIXED_CTRx. The maximum value m can be determined from CPUID.0AH:EDX[4:0].	
59:33+*m*	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.	
63	Reserved.	
Register Address: 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.	
Register Address: 14C1H—14C4H, 5313—5316	MSR_RELOAD_PMCx	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Reload value for IA32_PMCx (R/W)		Core
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 18.17, "Time-Stamp Counter," and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Package
49:0	Reserved.	
52:50	See Table 2-2.	
63:53	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	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.	Package
27:16	Reserved.	
28	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.	Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
29	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.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 133.33MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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-sate 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.	
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.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org.		Core
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.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
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.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Thread
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.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Core
15:0	Current Performance State Value.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:16	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
0	Reserved.	
3:1	On demand Clock Modulation Duty Cycle (R/W)	
4	On demand Clock Modulation Enable (R/W)	
63:5	Reserved.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 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.	Thread
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Thread
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM. (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	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.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Thread
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.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	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.	Core
1	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).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	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.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
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.	Package
1	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].	Thread
63:2	Reserved.	
Register Address: 1ACH, 428	MSR_TURBO_POWER_CURRENT_LIMIT	
See http://biosbits.org.		
14:0	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.	Package
15	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
30:16	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.	Package
31	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
63:32	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."		Core
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.	
Register Address: 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 MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
Register Address: 1DDH, 477	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.		Thread
Register Address: 1DEH, 478	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.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org.		Core
0	Reserved.	
1	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).	Package
63:2	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MC0_CTL2	
See Table 2-2.		Package
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Package
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Core
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."		Thread
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.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
Provides single-bit status used by software to query the overflow condition of each performance counter. (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
61	UNC_Ovf Uncore overflowed if 1.	
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 20.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
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.		Thread
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
(R/W)		Thread
61	CLR_UNC_Ovf Set 1 to clear UNC_Ovf.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 20.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
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)	
63:36	Reserved.	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	
See Section 20.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 400H, 1024	IA32_MC0_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Package
Register Address: 402H, 1026	IA32_MC0_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package
Register Address: 403H, 1027	IA32_MC0_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Package
Register Address: 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 ADDRV 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.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 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 ADDRV 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.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40EH, 1038	IA32_MC3_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 ADDRV 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.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 412H, 1042	IA32_MC4_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 ADDRV 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.		Core
Register Address: 413H, 1043	IA32_MC4_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 20.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ See Section 18.9.1 and record format in Section 18.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
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.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 18.17.2, "IA32_TSC_AUX Register and RDTSCP Support."		Thread

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Actual maximum turbo frequency is multiplied by 133.33MHz. (Not available in model 06_2EH.)		Package
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.	
Register Address: 301H, 769	MSR_GQ_SNOOP_MESF	
MSR_GQ_SNOOP_MESF		Package
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)	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
5	From E to I (R/W)	
6	From S to I (R/W)	
7	From F to I (R/W)	
63:8	Reserved.	
Register Address: 391H, 913	MSR_UNCORE_PERF_GLOBAL_CTRL	
See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 392H, 914	MSR_UNCORE_PERF_GLOBAL_STATUS	
See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 393H, 915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	
See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 394H, 916	MSR_UNCORE_FIXED_CTR0	
See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 395H, 917	MSR_UNCORE_FIXED_CTR_CTRL	
See Section 20.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 396H, 918	MSR_UNCORE_ADDR_OPCODE_MATCH	
See Section 20.3.1.2.3, "Uncore Address/Opcode Match MSR."		Package
Register Address: 3B0H, 960	MSR_UNCORE_PMC0	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B1H, 961	MSR_UNCORE_PMC1	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B2H, 962	MSR_UNCORE_PMC2	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B3H, 963	MSR_UNCORE_PMC3	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B4H, 964	MSR_UNCORE_PMC4	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B6H, 966	MSR_UNCORE_PMC6	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B7H, 967	MSR_UNCORE_PMC7	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C0H, 944	MSR_UNCORE_PERFEVTSEL0	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C1H, 945	MSR_UNCORE_PERFEVTSEL1	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C2H, 946	MSR_UNCORE_PERFEVTSEL2	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C3H, 947	MSR_UNCORE_PERFEVTSEL3	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C4H, 948	MSR_UNCORE_PERFEVTSEL4	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C5H, 949	MSR_UNCORE_PERFEVTSEL5	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C6H, 950	MSR_UNCORE_PERFEVTSEL6	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C7H, 951	MSR_UNCORE_PERFEVTSEL7	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 394H, 816	MSR_W_PMON_FIXED_CTR	
Uncore W-box perfmon fixed counter.		Package
Register Address: 395H, 817	MSR_W_PMON_FIXED_CTR_CTL	
Uncore U-box perfmon fixed counter control MSR.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: C00H, 3072	MSR_U_PMON_GLOBAL_CTRL	
Uncore U-box perfmon global control MSR.		Package
Register Address: C01H, 3073	MSR_U_PMON_GLOBAL_STATUS	
Uncore U-box perfmon global status MSR.		Package
Register Address: C02H, 3074	MSR_U_PMON_GLOBAL_OVF_CTRL	
Uncore U-box perfmon global overflow control MSR.		Package
Register Address: C10H, 3088	MSR_U_PMON_EVNT_SEL	
Uncore U-box perfmon event select MSR.		Package
Register Address: C11H, 3089	MSR_U_PMON_CTR	
Uncore U-box perfmon counter MSR.		Package
Register Address: C20H, 3104	MSR_B0_PMON_BOX_CTRL	
Uncore B-box 0 perfmon local box control MSR.		Package
Register Address: C21H, 3105	MSR_B0_PMON_BOX_STATUS	
Uncore B-box 0 perfmon local box status MSR.		Package
Register Address: C22H, 3106	MSR_B0_PMON_BOX_OVF_CTRL	
Uncore B-box 0 perfmon local box overflow control MSR.		Package
Register Address: C30H, 3120	MSR_B0_PMON_EVNT_SEL0	
Uncore B-box 0 perfmon event select MSR.		Package
Register Address: C31H, 3121	MSR_B0_PMON_CTR0	
Uncore B-box 0 perfmon counter MSR.		Package
Register Address: C32H, 3122	MSR_B0_PMON_EVNT_SEL1	
Uncore B-box 0 perfmon event select MSR.		Package
Register Address: C33H, 3123	MSR_B0_PMON_CTR1	
Uncore B-box 0 perfmon counter MSR.		Package
Register Address: C34H, 3124	MSR_B0_PMON_EVNT_SEL2	
Uncore B-box 0 perfmon event select MSR.		Package
Register Address: C35H, 3125	MSR_B0_PMON_CTR2	
Uncore B-box 0 perfmon counter MSR.		Package
Register Address: C36H, 3126	MSR_B0_PMON_EVNT_SEL3	
Uncore B-box 0 perfmon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: C37H, 3127	MSR_B0_PMON_CTR3	
Uncore B-box 0 perfmon counter MSR.		Package
Register Address: C40H, 3136	MSR_S0_PMON_BOX_CTRL	
Uncore S-box 0 perfmon local box control MSR.		Package
Register Address: C41H, 3137	MSR_S0_PMON_BOX_STATUS	
Uncore S-box 0 perfmon local box status MSR.		Package
Register Address: C42H, 3138	MSR_S0_PMON_BOX_OVF_CTRL	
Uncore S-box 0 perfmon local box overflow control MSR.		Package
Register Address: C50H, 3152	MSR_S0_PMON_EVNT_SEL0	
Uncore S-box 0 perfmon event select MSR.		Package
Register Address: C51H, 3153	MSR_S0_PMON_CTR0	
Uncore S-box 0 perfmon counter MSR.		Package
Register Address: C52H, 3154	MSR_S0_PMON_EVNT_SEL1	
Uncore S-box 0 perfmon event select MSR.		Package
Register Address: C53H, 3155	MSR_S0_PMON_CTR1	
Uncore S-box 0 perfmon counter MSR.		Package
Register Address: C54H, 3156	MSR_S0_PMON_EVNT_SEL2	
Uncore S-box 0 perfmon event select MSR.		Package
Register Address: C55H, 3157	MSR_S0_PMON_CTR2	
Uncore S-box 0 perfmon counter MSR.		Package
Register Address: C56H, 3158	MSR_S0_PMON_EVNT_SEL3	
Uncore S-box 0 perfmon event select MSR.		Package
Register Address: C57H, 3159	MSR_S0_PMON_CTR3	
Uncore S-box 0 perfmon counter MSR.		Package
Register Address: C60H, 3168	MSR_B1_PMON_BOX_CTRL	
Uncore B-box 1 perfmon local box control MSR.		Package
Register Address: C61H, 3169	MSR_B1_PMON_BOX_STATUS	
Uncore B-box 1 perfmon local box status MSR.		Package
Register Address: C62H, 3170	MSR_B1_PMON_BOX_OVF_CTRL	
Uncore B-box 1 perfmon local box overflow control MSR.		Package
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SEL0	
Uncore B-box 1 perfmon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 perfmon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 perfmon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTR2	
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vperfmon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTR3	
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C80H, 3120	MSR_W_PMON_BOX_CTRL	
Uncore W-box perfmon local box control MSR.		Package
Register Address: C81H, 3121	MSR_W_PMON_BOX_STATUS	
Uncore W-box perfmon local box status MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box perfmon local box overflow control MSR.		Package
Register Address: C90H, 3136	MSR_W_PMON_EVNT_SEL0	
Uncore W-box perfmon event select MSR.		Package
Register Address: C91H, 3137	MSR_W_PMON_CTR0	
Uncore W-box perfmon counter MSR.		Package
Register Address: C92H, 3138	MSR_W_PMON_EVNT_SEL1	
Uncore W-box perfmon event select MSR.		Package
Register Address: C93H, 3139	MSR_W_PMON_CTR1	
Uncore W-box perfmon counter MSR.		Package
Register Address: C94H, 3140	MSR_W_PMON_EVNT_SEL2	
Uncore W-box perfmon event select MSR.		Package
Register Address: C95H, 3141	MSR_W_PMON_CTR2	
Uncore W-box perfmon counter MSR.		Package
Register Address: C96H, 3142	MSR_W_PMON_EVNT_SEL3	
Uncore W-box perfmon event select MSR.		Package
Register Address: C97H, 3143	MSR_W_PMON_CTR3	
Uncore W-box perfmon counter MSR.		Package
Register Address: CA0H, 3232	MSR_M0_PMON_BOX_CTRL	
Uncore M-box 0 perfmon local box control MSR.		Package
Register Address: CA1H, 3233	MSR_M0_PMON_BOX_STATUS	
Uncore M-box 0 perfmon local box status MSR.		Package
Register Address: CA2H, 3234	MSR_M0_PMON_BOX_OVF_CTRL	
Uncore M-box 0 perfmon local box overflow control MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: CA4H, 3236	MSR_M0_PMON_TIMESTAMP	
Uncore M-box 0 perfmon time stamp unit select MSR.		Package
Register Address: CA5H, 3237	MSR_M0_PMON_DSP	
Uncore M-box 0 perfmon DSP unit select MSR.		Package
Register Address: CA6H, 3238	MSR_M0_PMON_ISS	
Uncore M-box 0 perfmon ISS unit select MSR.		Package
Register Address: CA7H, 3239	MSR_M0_PMON_MAP	
Uncore M-box 0 perfmon MAP unit select MSR.		Package
Register Address: CA8H, 3240	MSR_M0_PMON_MSC_THR	
Uncore M-box 0 perfmon MIC THR select MSR.		Package
Register Address: CA9H, 3241	MSR_M0_PMON_PGT	
Uncore M-box 0 perfmon PGT unit select MSR.		Package
Register Address: CAAH, 3242	MSR_M0_PMON_PLD	
Uncore M-box 0 perfmon PLD unit select MSR.		Package
Register Address: CABH, 3243	MSR_M0_PMON_ZDP	
Uncore M-box 0 perfmon ZDP unit select MSR.		Package
Register Address: CB0H, 3248	MSR_M0_PMON_EVNT_SEL0	
Uncore M-box 0 perfmon event select MSR.		Package
Register Address: CB1H, 3249	MSR_M0_PMON_CTR0	
Uncore M-box 0 perfmon counter MSR.		Package
Register Address: CB2H, 3250	MSR_M0_PMON_EVNT_SEL1	
Uncore M-box 0 perfmon event select MSR.		Package
Register Address: CB3H, 3251	MSR_M0_PMON_CTR1	
Uncore M-box 0 perfmon counter MSR.		Package
Register Address: CB4H, 3252	MSR_M0_PMON_EVNT_SEL2	
Uncore M-box 0 perfmon event select MSR.		Package
Register Address: CB5H, 3253	MSR_M0_PMON_CTR2	
Uncore M-box 0 perfmon counter MSR.		Package
Register Address: CB6H, 3254	MSR_M0_PMON_EVNT_SEL3	
Uncore M-box 0 perfmon event select MSR.		Package
Register Address: CB7H, 3255	MSR_M0_PMON_CTR3	
Uncore M-box 0 perfmon counter MSR.		Package
Register Address: CB8H, 3256	MSR_M0_PMON_EVNT_SEL4	
Uncore M-box 0 perfmon event select MSR.		Package
Register Address: CB9H, 3257	MSR_M0_PMON_CTR4	
Uncore M-box 0 perfmon counter MSR.		Package
Register Address: CBAH, 3258	MSR_M0_PMON_EVNT_SEL5	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore M-box 0 perfmon event select MSR.		Package
Register Address: CBBH, 3259	MSR_M0_PMON_CTR5	
Uncore M-box 0 perfmon counter MSR.		Package
Register Address: CC0H, 3264	MSR_S1_PMON_BOX_CTRL	
Uncore S-box 1 perfmon local box control MSR.		Package
Register Address: CC1H, 3265	MSR_S1_PMON_BOX_STATUS	
Uncore S-box 1 perfmon local box status MSR.		Package
Register Address: CC2H, 3266	MSR_S1_PMON_BOX_OVF_CTRL	
Uncore S-box 1 perfmon local box overflow control MSR.		Package
Register Address: CD0H, 3280	MSR_S1_PMON_EVNT_SEL0	
Uncore S-box 1 perfmon event select MSR.		Package
Register Address: CD1H, 3281	MSR_S1_PMON_CTR0	
Uncore S-box 1 perfmon counter MSR.		Package
Register Address: CD2H, 3282	MSR_S1_PMON_EVNT_SEL1	
Uncore S-box 1 perfmon event select MSR.		Package
Register Address: CD3H, 3283	MSR_S1_PMON_CTR1	
Uncore S-box 1 perfmon counter MSR.		Package
Register Address: CD4H, 3284	MSR_S1_PMON_EVNT_SEL2	
Uncore S-box 1 perfmon event select MSR.		Package
Register Address: CD5H, 3285	MSR_S1_PMON_CTR2	
Uncore S-box 1 perfmon counter MSR.		Package
Register Address: CD6H, 3286	MSR_S1_PMON_EVNT_SEL3	
Uncore S-box 1 perfmon event select MSR.		Package
Register Address: CD7H, 3287	MSR_S1_PMON_CTR3	
Uncore S-box 1 perfmon counter MSR.		Package
Register Address: CE0H, 3296	MSR_M1_PMON_BOX_CTRL	
Uncore M-box 1 perfmon local box control MSR.		Package
Register Address: CE1H, 3297	MSR_M1_PMON_BOX_STATUS	
Uncore M-box 1 perfmon local box status MSR.		Package
Register Address: CE2H, 3298	MSR_M1_PMON_BOX_OVF_CTRL	
Uncore M-box 1 perfmon local box overflow control MSR.		Package
Register Address: CE4H, 3300	MSR_M1_PMON_TIMESTAMP	
Uncore M-box 1 perfmon time stamp unit select MSR.		Package
Register Address: CE5H, 3301	MSR_M1_PMON_DSP	
Uncore M-box 1 perfmon DSP unit select MSR.		Package
Register Address: CE6H, 3302	MSR_M1_PMON_ISS	
Uncore M-box 1 perfmon ISS unit select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: CE7H, 3303	MSR_M1_PMON_MAP	
Uncore M-box 1 perfmon MAP unit select MSR.		Package
Register Address: CE8H, 3304	MSR_M1_PMON_MSC_THR	
Uncore M-box 1 perfmon MIC THR select MSR.		Package
Register Address: CE9H, 3305	MSR_M1_PMON_PGT	
Uncore M-box 1 perfmon PGT unit select MSR.		Package
Register Address: CEAH, 3306	MSR_M1_PMON_PLD	
Uncore M-box 1 perfmon PLD unit select MSR.		Package
Register Address: CEBH, 3307	MSR_M1_PMON_ZDP	
Uncore M-box 1 perfmon ZDP unit select MSR.		Package
Register Address: CF0H, 3312	MSR_M1_PMON_EVNT_SEL0	
Uncore M-box 1 perfmon event select MSR.		Package
Register Address: CF1H, 3313	MSR_M1_PMON_CTR0	
Uncore M-box 1 perfmon counter MSR.		Package
Register Address: CF2H, 3314	MSR_M1_PMON_EVNT_SEL1	
Uncore M-box 1 perfmon event select MSR.		Package
Register Address: CF3H, 3315	MSR_M1_PMON_CTR1	
Uncore M-box 1 perfmon counter MSR.		Package
Register Address: CF4H, 3316	MSR_M1_PMON_EVNT_SEL2	
Uncore M-box 1 perfmon event select MSR.		Package
Register Address: CF5H, 3317	MSR_M1_PMON_CTR2	
Uncore M-box 1 perfmon counter MSR.		Package
Register Address: CF6H, 3318	MSR_M1_PMON_EVNT_SEL3	
Uncore M-box 1 perfmon event select MSR.		Package
Register Address: CF7H, 3319	MSR_M1_PMON_CTR3	
Uncore M-box 1 perfmon counter MSR.		Package
Register Address: CF8H, 3320	MSR_M1_PMON_EVNT_SEL4	
Uncore M-box 1 perfmon event select MSR.		Package
Register Address: CF9H, 3321	MSR_M1_PMON_CTR4	
Uncore M-box 1 perfmon counter MSR.		Package
Register Address: CFAH, 3322	MSR_M1_PMON_EVNT_SEL5	
Uncore M-box 1 perfmon event select MSR.		Package
Register Address: CFBH, 3323	MSR_M1_PMON_CTR5	
Uncore M-box 1 perfmon counter MSR.		Package
Register Address: D00H, 3328	MSR_C0_PMON_BOX_CTRL	
Uncore C-box 0 perfmon local box control MSR.		Package
Register Address: D01H, 3329	MSR_C0_PMON_BOX_STATUS	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-box 0 perfmon local box status MSR.		Package
Register Address: D02H, 3330	MSR_C0_PMON_BOX_OVF_CTRL	
Uncore C-box 0 perfmon local box overflow control MSR.		Package
Register Address: D10H, 3344	MSR_C0_PMON_EVNT_SEL0	
Uncore C-box 0 perfmon event select MSR.		Package
Register Address: D11H, 3345	MSR_C0_PMON_CTR0	
Uncore C-box 0 perfmon counter MSR.		Package
Register Address: D12H, 3346	MSR_C0_PMON_EVNT_SEL1	
Uncore C-box 0 perfmon event select MSR.		Package
Register Address: D13H, 3347	MSR_C0_PMON_CTR1	
Uncore C-box 0 perfmon counter MSR.		Package
Register Address: D14H, 3348	MSR_C0_PMON_EVNT_SEL2	
Uncore C-box 0 perfmon event select MSR.		Package
Register Address: D15H, 3349	MSR_C0_PMON_CTR2	
Uncore C-box 0 perfmon counter MSR.		Package
Register Address: D16H, 3350	MSR_C0_PMON_EVNT_SEL3	
Uncore C-box 0 perfmon event select MSR.		Package
Register Address: D17H, 3351	MSR_C0_PMON_CTR3	
Uncore C-box 0 perfmon counter MSR.		Package
Register Address: D18H, 3352	MSR_C0_PMON_EVNT_SEL4	
Uncore C-box 0 perfmon event select MSR.		Package
Register Address: D19H, 3353	MSR_C0_PMON_CTR4	
Uncore C-box 0 perfmon counter MSR.		Package
Register Address: D1AH, 3354	MSR_C0_PMON_EVNT_SEL5	
Uncore C-box 0 perfmon event select MSR.		Package
Register Address: D1BH, 3355	MSR_C0_PMON_CTR5	
Uncore C-box 0 perfmon counter MSR.		Package
Register Address: D20H, 3360	MSR_C4_PMON_BOX_CTRL	
Uncore C-box 4 perfmon local box control MSR.		Package
Register Address: D21H, 3361	MSR_C4_PMON_BOX_STATUS	
Uncore C-box 4 perfmon local box status MSR.		Package
Register Address: D22H, 3362	MSR_C4_PMON_BOX_OVF_CTRL	
Uncore C-box 4 perfmon local box overflow control MSR.		Package
Register Address: D30H, 3376	MSR_C4_PMON_EVNT_SEL0	
Uncore C-box 4 perfmon event select MSR.		Package
Register Address: D31H, 3377	MSR_C4_PMON_CTR0	
Uncore C-box 4 perfmon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: D32H, 3378	MSR_C4_PMON_EVNT_SEL1	
Uncore C-box 4 perfmon event select MSR.		Package
Register Address: D33H, 3379	MSR_C4_PMON_CTR1	
Uncore C-box 4 perfmon counter MSR.		Package
Register Address: D34H, 3380	MSR_C4_PMON_EVNT_SEL2	
Uncore C-box 4 perfmon event select MSR.		Package
Register Address: D35H, 3381	MSR_C4_PMON_CTR2	
Uncore C-box 4 perfmon counter MSR.		Package
Register Address: D36H, 3382	MSR_C4_PMON_EVNT_SEL3	
Uncore C-box 4 perfmon event select MSR.		Package
Register Address: D37H, 3383	MSR_C4_PMON_CTR3	
Uncore C-box 4 perfmon counter MSR.		Package
Register Address: D38H, 3384	MSR_C4_PMON_EVNT_SEL4	
Uncore C-box 4 perfmon event select MSR.		Package
Register Address: D39H, 3385	MSR_C4_PMON_CTR4	
Uncore C-box 4 perfmon counter MSR.		Package
Register Address: D3AH, 3386	MSR_C4_PMON_EVNT_SEL5	
Uncore C-box 4 perfmon event select MSR.		Package
Register Address: D3BH, 3387	MSR_C4_PMON_CTR5	
Uncore C-box 4 perfmon counter MSR.		Package
Register Address: D40H, 3392	MSR_C2_PMON_BOX_CTRL	
Uncore C-box 2 perfmon local box control MSR.		Package
Register Address: D41H, 3393	MSR_C2_PMON_BOX_STATUS	
Uncore C-box 2 perfmon local box status MSR.		Package
Register Address: D42H, 3394	MSR_C2_PMON_BOX_OVF_CTRL	
Uncore C-box 2 perfmon local box overflow control MSR.		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVNT_SEL0	
Uncore C-box 2 perfmon event select MSR.		Package
Register Address: D51H, 3409	MSR_C2_PMON_CTR0	
Uncore C-box 2 perfmon counter MSR.		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNT_SEL1	
Uncore C-box 2 perfmon event select MSR.		Package
Register Address: D53H, 3411	MSR_C2_PMON_CTR1	
Uncore C-box 2 perfmon counter MSR.		Package
Register Address: D54H, 3412	MSR_C2_PMON_EVNT_SEL2	
Uncore C-box 2 perfmon event select MSR.		Package
Register Address: D55H, 3413	MSR_C2_PMON_CTR2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-box 2 perfmon counter MSR.		Package
Register Address: D56H, 3414	MSR_C2_PMON_EVNT_SEL3	
Uncore C-box 2 perfmon event select MSR.		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR3	
Uncore C-box 2 perfmon counter MSR.		Package
Register Address: D58H, 3416	MSR_C2_PMON_EVNT_SEL4	
Uncore C-box 2 perfmon event select MSR.		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR4	
Uncore C-box 2 perfmon counter MSR.		Package
Register Address: D5AH, 3418	MSR_C2_PMON_EVNT_SEL5	
Uncore C-box 2 perfmon event select MSR.		Package
Register Address: D5BH, 3419	MSR_C2_PMON_CTR5	
Uncore C-box 2 perfmon counter MSR.		Package
Register Address: D60H, 3424	MSR_C6_PMON_BOX_CTRL	
Uncore C-box 6 perfmon local box control MSR.		Package
Register Address: D61H, 3425	MSR_C6_PMON_BOX_STATUS	
Uncore C-box 6 perfmon local box status MSR.		Package
Register Address: D62H, 3426	MSR_C6_PMON_BOX_OVF_CTRL	
Uncore C-box 6 perfmon local box overflow control MSR.		Package
Register Address: D70H, 3440	MSR_C6_PMON_EVNT_SEL0	
Uncore C-box 6 perfmon event select MSR.		Package
Register Address: D71H, 3441	MSR_C6_PMON_CTR0	
Uncore C-box 6 perfmon counter MSR.		Package
Register Address: D72H, 3442	MSR_C6_PMON_EVNT_SEL1	
Uncore C-box 6 perfmon event select MSR.		Package
Register Address: D73H, 3443	MSR_C6_PMON_CTR1	
Uncore C-box 6 perfmon counter MSR.		Package
Register Address: D74H, 3444	MSR_C6_PMON_EVNT_SEL2	
Uncore C-box 6 perfmon event select MSR.		Package
Register Address: D75H, 3445	MSR_C6_PMON_CTR2	
Uncore C-box 6 perfmon counter MSR.		Package
Register Address: D76H, 3446	MSR_C6_PMON_EVNT_SEL3	
Uncore C-box 6 perfmon event select MSR.		Package
Register Address: D77H, 3447	MSR_C6_PMON_CTR3	
Uncore C-box 6 perfmon counter MSR.		Package
Register Address: D78H, 3448	MSR_C6_PMON_EVNT_SEL4	
Uncore C-box 6 perfmon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: D79H, 3449	MSR_C6_PMON_CTR4	
Uncore C-box 6 perfmon counter MSR.		Package
Register Address: D7AH, 3450	MSR_C6_PMON_EVNT_SEL5	
Uncore C-box 6 perfmon event select MSR.		Package
Register Address: D7BH, 3451	MSR_C6_PMON_CTR5	
Uncore C-box 6 perfmon counter MSR.		Package
Register Address: D80H, 3456	MSR_C1_PMON_BOX_CTRL	
Uncore C-box 1 perfmon local box control MSR.		Package
Register Address: D81H, 3457	MSR_C1_PMON_BOX_STATUS	
Uncore C-box 1 perfmon local box status MSR.		Package
Register Address: D82H, 3458	MSR_C1_PMON_BOX_OVF_CTRL	
Uncore C-box 1 perfmon local box overflow control MSR.		Package
Register Address: D90H, 3472	MSR_C1_PMON_EVNT_SEL0	
Uncore C-box 1 perfmon event select MSR.		Package
Register Address: D91H, 3473	MSR_C1_PMON_CTR0	
Uncore C-box 1 perfmon counter MSR.		Package
Register Address: D92H, 3474	MSR_C1_PMON_EVNT_SEL1	
Uncore C-box 1 perfmon event select MSR.		Package
Register Address: D93H, 3475	MSR_C1_PMON_CTR1	
Uncore C-box 1 perfmon counter MSR.		Package
Register Address: D94H, 3476	MSR_C1_PMON_EVNT_SEL2	
Uncore C-box 1 perfmon event select MSR.		Package
Register Address: D95H, 3477	MSR_C1_PMON_CTR2	
Uncore C-box 1 perfmon counter MSR.		Package
Register Address: D96H, 3478	MSR_C1_PMON_EVNT_SEL3	
Uncore C-box 1 perfmon event select MSR.		Package
Register Address: D97H, 3479	MSR_C1_PMON_CTR3	
Uncore C-box 1 perfmon counter MSR.		Package
Register Address: D98H, 3480	MSR_C1_PMON_EVNT_SEL4	
Uncore C-box 1 perfmon event select MSR.		Package
Register Address: D99H, 3481	MSR_C1_PMON_CTR4	
Uncore C-box 1 perfmon counter MSR.		Package
Register Address: D9AH, 3482	MSR_C1_PMON_EVNT_SEL5	
Uncore C-box 1 perfmon event select MSR.		Package
Register Address: D9BH, 3483	MSR_C1_PMON_CTR5	
Uncore C-box 1 perfmon counter MSR.		Package
Register Address: DA0H, 3488	MSR_C5_PMON_BOX_CTRL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-box 5 perfmon local box control MSR.		Package
Register Address: DA1H, 3489	MSR_C5_PMON_BOX_STATUS	
Uncore C-box 5 perfmon local box status MSR.		Package
Register Address: DA2H, 3490	MSR_C5_PMON_BOX_OVF_CTRL	
Uncore C-box 5 perfmon local box overflow control MSR.		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNT_SEL0	
Uncore C-box 5 perfmon event select MSR.		Package
Register Address: DB1H, 3505	MSR_C5_PMON_CTR0	
Uncore C-box 5 perfmon counter MSR.		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNT_SEL1	
Uncore C-box 5 perfmon event select MSR.		Package
Register Address: DB3H, 3507	MSR_C5_PMON_CTR1	
Uncore C-box 5 perfmon counter MSR.		Package
Register Address: DB4H, 3508	MSR_C5_PMON_EVNT_SEL2	
Uncore C-box 5 perfmon event select MSR.		Package
Register Address: DB5H, 3509	MSR_C5_PMON_CTR2	
Uncore C-box 5 perfmon counter MSR.		Package
Register Address: DB6H, 3510	MSR_C5_PMON_EVNT_SEL3	
Uncore C-box 5 perfmon event select MSR.		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR3	
Uncore C-box 5 perfmon counter MSR.		Package
Register Address: DB8H, 3512	MSR_C5_PMON_EVNT_SEL4	
Uncore C-box 5 perfmon event select MSR.		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR4	
Uncore C-box 5 perfmon counter MSR.		Package
Register Address: DBAH, 3514	MSR_C5_PMON_EVNT_SEL5	
Uncore C-box 5 perfmon event select MSR.		Package
Register Address: DBBH, 3515	MSR_C5_PMON_CTR5	
Uncore C-box 5 perfmon counter MSR.		Package
Register Address: DC0H, 3520	MSR_C3_PMON_BOX_CTRL	
Uncore C-box 3 perfmon local box control MSR.		Package
Register Address: DC1H, 3521	MSR_C3_PMON_BOX_STATUS	
Uncore C-box 3 perfmon local box status MSR.		Package
Register Address: DC2H, 3522	MSR_C3_PMON_BOX_OVF_CTRL	
Uncore C-box 3 perfmon local box overflow control MSR.		Package
Register Address: DD0H, 3536	MSR_C3_PMON_EVNT_SEL0	
Uncore C-box 3 perfmon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: DD1H, 3537	MSR_C3_PMON_CTR0	
Uncore C-box 3 perfmon counter MSR.		Package
Register Address: DD2H, 3538	MSR_C3_PMON_EVNT_SEL1	
Uncore C-box 3 perfmon event select MSR.		Package
Register Address: DD3H, 3539	MSR_C3_PMON_CTR1	
Uncore C-box 3 perfmon counter MSR.		Package
Register Address: DD4H, 3540	MSR_C3_PMON_EVNT_SEL2	
Uncore C-box 3 perfmon event select MSR.		Package
Register Address: DD5H, 3541	MSR_C3_PMON_CTR2	
Uncore C-box 3 perfmon counter MSR.		Package
Register Address: DD6H, 3542	MSR_C3_PMON_EVNT_SEL3	
Uncore C-box 3 perfmon event select MSR.		Package
Register Address: DD7H, 3543	MSR_C3_PMON_CTR3	
Uncore C-box 3 perfmon counter MSR.		Package
Register Address: DD8H, 3544	MSR_C3_PMON_EVNT_SEL4	
Uncore C-box 3 perfmon event select MSR.		Package
Register Address: DD9H, 3545	MSR_C3_PMON_CTR4	
Uncore C-box 3 perfmon counter MSR.		Package
Register Address: DDAH, 3546	MSR_C3_PMON_EVNT_SEL5	
Uncore C-box 3 perfmon event select MSR.		Package
Register Address: DDBH, 3547	MSR_C3_PMON_CTR5	
Uncore C-box 3 perfmon counter MSR.		Package
Register Address: DE0H, 3552	MSR_C7_PMON_BOX_CTRL	
Uncore C-box 7 perfmon local box control MSR.		Package
Register Address: DE1H, 3553	MSR_C7_PMON_BOX_STATUS	
Uncore C-box 7 perfmon local box status MSR.		Package
Register Address: DE2H, 3554	MSR_C7_PMON_BOX_OVF_CTRL	
Uncore C-box 7 perfmon local box overflow control MSR.		Package
Register Address: DF0H, 3568	MSR_C7_PMON_EVNT_SEL0	
Uncore C-box 7 perfmon event select MSR.		Package
Register Address: DF1H, 3569	MSR_C7_PMON_CTR0	
Uncore C-box 7 perfmon counter MSR.		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVNT_SEL1	
Uncore C-box 7 perfmon event select MSR.		Package
Register Address: DF3H, 3571	MSR_C7_PMON_CTR1	
Uncore C-box 7 perfmon counter MSR.		Package
Register Address: DF4H, 3572	MSR_C7_PMON_EVNT_SEL2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-box 7 perfmon event select MSR.		Package
Register Address: DF5H, 3573	MSR_C7_PMON_CTR2	
Uncore C-box 7 perfmon counter MSR.		Package
Register Address: DF6H, 3574	MSR_C7_PMON_EVNT_SEL3	
Uncore C-box 7 perfmon event select MSR.		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTR3	
Uncore C-box 7 perfmon counter MSR.		Package
Register Address: DF8H, 3576	MSR_C7_PMON_EVNT_SEL4	
Uncore C-box 7 perfmon event select MSR.		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR4	
Uncore C-box 7 perfmon counter MSR.		Package
Register Address: DFAH, 3578	MSR_C7_PMON_EVNT_SEL5	
Uncore C-box 7 perfmon event select MSR.		Package
Register Address: DFBH, 3579	MSR_C7_PMON_CTR5	
Uncore C-box 7 perfmon counter MSR.		Package
Register Address: E00H, 3584	MSR_R0_PMON_BOX_CTRL	
Uncore R-box 0 perfmon local box control MSR.		Package
Register Address: E01H, 3585	MSR_R0_PMON_BOX_STATUS	
Uncore R-box 0 perfmon local box status MSR.		Package
Register Address: E02H, 3586	MSR_R0_PMON_BOX_OVF_CTRL	
Uncore R-box 0 perfmon local box overflow control MSR.		Package
Register Address: E04H, 3588	MSR_R0_PMON_IPERF0_P0	
Uncore R-box 0 perfmon IPERF0 unit Port 0 select MSR.		Package
Register Address: E05H, 3589	MSR_R0_PMON_IPERF0_P1	
Uncore R-box 0 perfmon IPERF0 unit Port 1 select MSR.		Package
Register Address: E06H, 3590	MSR_R0_PMON_IPERF0_P2	
Uncore R-box 0 perfmon IPERF0 unit Port 2 select MSR.		Package
Register Address: E07H, 3591	MSR_R0_PMON_IPERF0_P3	
Uncore R-box 0 perfmon IPERF0 unit Port 3 select MSR.		Package
Register Address: E08H, 3592	MSR_R0_PMON_IPERF0_P4	
Uncore R-box 0 perfmon IPERF0 unit Port 4 select MSR.		Package
Register Address: E09H, 3593	MSR_R0_PMON_IPERF0_P5	
Uncore R-box 0 perfmon IPERF0 unit Port 5 select MSR.		Package
Register Address: E0AH, 3594	MSR_R0_PMON_IPERF0_P6	
Uncore R-box 0 perfmon IPERF0 unit Port 6 select MSR.		Package
Register Address: E0BH, 3595	MSR_R0_PMON_IPERF0_P7	
Uncore R-box 0 perfmon IPERF0 unit Port 7 select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E0CH, 3596	MSR_R0_PMON_QLX_P0	
Uncore R-box 0 perfmon QLX unit Port 0 select MSR.		Package
Register Address: E0DH, 3597	MSR_R0_PMON_QLX_P1	
Uncore R-box 0 perfmon QLX unit Port 1 select MSR.		Package
Register Address: E0EH, 3598	MSR_R0_PMON_QLX_P2	
Uncore R-box 0 perfmon QLX unit Port 2 select MSR.		Package
Register Address: E0FH, 3599	MSR_R0_PMON_QLX_P3	
Uncore R-box 0 perfmon QLX unit Port 3 select MSR.		Package
Register Address: E10H, 3600	MSR_R0_PMON_EVNT_SEL0	
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E11H, 3601	MSR_R0_PMON_CTR0	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E12H, 3602	MSR_R0_PMON_EVNT_SEL1	
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E13H, 3603	MSR_R0_PMON_CTR1	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E14H, 3604	MSR_R0_PMON_EVNT_SEL2	
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E15H, 3605	MSR_R0_PMON_CTR2	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E16H, 3606	MSR_R0_PMON_EVNT_SEL3	
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E17H, 3607	MSR_R0_PMON_CTR3	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E18H, 3608	MSR_R0_PMON_EVNT_SEL4	
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E19H, 3609	MSR_R0_PMON_CTR4	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E1AH, 3610	MSR_R0_PMON_EVNT_SEL5	
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E1BH, 3611	MSR_R0_PMON_CTR5	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E1CH, 3612	MSR_R0_PMON_EVNT_SEL6	
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E1DH, 3613	MSR_R0_PMON_CTR6	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E1EH, 3614	MSR_R0_PMON_EVNT_SEL7	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore R-box 0 perfmon event select MSR.		Package
Register Address: E1FH, 3615	MSR_R0_PMON_CTR7	
Uncore R-box 0 perfmon counter MSR.		Package
Register Address: E20H, 3616	MSR_R1_PMON_BOX_CTRL	
Uncore R-box 1 perfmon local box control MSR.		Package
Register Address: E21H, 3617	MSR_R1_PMON_BOX_STATUS	
Uncore R-box 1 perfmon local box status MSR.		Package
Register Address: E22H, 3618	MSR_R1_PMON_BOX_OVF_CTRL	
Uncore R-box 1 perfmon local box overflow control MSR.		Package
Register Address: E24H, 3620	MSR_R1_PMON_IPERF1_P8	
Uncore R-box 1 perfmon IPERF1 unit Port 8 select MSR.		Package
Register Address: E25H, 3621	MSR_R1_PMON_IPERF1_P9	
Uncore R-box 1 perfmon IPERF1 unit Port 9 select MSR.		Package
Register Address: E26H, 3622	MSR_R1_PMON_IPERF1_P10	
Uncore R-box 1 perfmon IPERF1 unit Port 10 select MSR.		Package
Register Address: E27H, 3623	MSR_R1_PMON_IPERF1_P11	
Uncore R-box 1 perfmon IPERF1 unit Port 11 select MSR.		Package
Register Address: E28H, 3624	MSR_R1_PMON_IPERF1_P12	
Uncore R-box 1 perfmon IPERF1 unit Port 12 select MSR.		Package
Register Address: E29H, 3625	MSR_R1_PMON_IPERF1_P13	
Uncore R-box 1 perfmon IPERF1 unit Port 13 select MSR.		Package
Register Address: E2AH, 3626	MSR_R1_PMON_IPERF1_P14	
Uncore R-box 1 perfmon IPERF1 unit Port 14 select MSR.		Package
Register Address: E2BH, 3627	MSR_R1_PMON_IPERF1_P15	
Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR.		Package
Register Address: E2CH, 3628	MSR_R1_PMON_QLX_P4	
Uncore R-box 1 perfmon QLX unit Port 4 select MSR.		Package
Register Address: E2DH, 3629	MSR_R1_PMON_QLX_P5	
Uncore R-box 1 perfmon QLX unit Port 5 select MSR.		Package
Register Address: E2EH, 3630	MSR_R1_PMON_QLX_P6	
Uncore R-box 1 perfmon QLX unit Port 6 select MSR.		Package
Register Address: E2FH, 3631	MSR_R1_PMON_QLX_P7	
Uncore R-box 1 perfmon QLX unit Port 7 select MSR.		Package
Register Address: E30H, 3632	MSR_R1_PMON_EVNT_SEL8	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E31H, 3633	MSR_R1_PMON_CTR8	
Uncore R-box 1 perfmon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E32H, 3634	MSR_R1_PMON_EVNT_SEL9	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E33H, 3635	MSR_R1_PMON_CTR9	
Uncore R-box 1 perfmon counter MSR.		Package
Register Address: E34H, 3636	MSR_R1_PMON_EVNT_SEL10	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E35H, 3637	MSR_R1_PMON_CTR10	
Uncore R-box 1 perfmon counter MSR.		Package
Register Address: E36H, 3638	MSR_R1_PMON_EVNT_SEL11	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E37H, 3639	MSR_R1_PMON_CTR11	
Uncore R-box 1 perfmon counter MSR.		Package
Register Address: E38H, 3640	MSR_R1_PMON_EVNT_SEL12	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E39H, 3641	MSR_R1_PMON_CTR12	
Uncore R-box 1 perfmon counter MSR.		Package
Register Address: E3AH, 3642	MSR_R1_PMON_EVNT_SEL13	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E3BH, 3643	MSR_R1_PMON_CTR13	
Uncore R-box 1perfmon counter MSR.		Package
Register Address: E3CH, 3644	MSR_R1_PMON_EVNT_SEL14	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E3DH, 3645	MSR_R1_PMON_CTR14	
Uncore R-box 1 perfmon counter MSR.		Package
Register Address: E3EH, 3646	MSR_R1_PMON_EVNT_SEL15	
Uncore R-box 1 perfmon event select MSR.		Package
Register Address: E3FH, 3647	MSR_R1_PMON_CTR15	
Uncore R-box 1 perfmon counter MSR.		Package
Register Address: E45H, 3653	MSR_B0_PMON_MATCH	
Uncore B-box 0 perfmon local box match MSR.		Package
Register Address: E46H, 3654	MSR_B0_PMON_MASK	
Uncore B-box 0 perfmon local box mask MSR.		Package
Register Address: E49H, 3657	MSR_S0_PMON_MATCH	
Uncore S-box 0 perfmon local box match MSR.		Package
Register Address: E4AH, 3658	MSR_S0_PMON_MASK	
Uncore S-box 0 perfmon local box mask MSR.		Package
Register Address: E4DH, 3661	MSR_B1_PMON_MATCH	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore B-box 1 perfmon local box match MSR.		Package
Register Address: E4EH, 3662	MSR_B1_PMON_MASK	
Uncore B-box 1 perfmon local box mask MSR.		Package
Register Address: E54H, 3668	MSR_M0_PMON_MM_CONFIG	
Uncore M-box 0 perfmon local box address match/mask config MSR.		Package
Register Address: E55H, 3669	MSR_M0_PMON_ADDR_MATCH	
Uncore M-box 0 perfmon local box address match MSR.		Package
Register Address: E56H, 3670	MSR_M0_PMON_ADDR_MASK	
Uncore M-box 0 perfmon local box address mask MSR.		Package
Register Address: E59H, 3673	MSR_S1_PMON_MATCH	
Uncore S-box 1 perfmon local box match MSR.		Package
Register Address: E5AH, 3674	MSR_S1_PMON_MASK	
Uncore S-box 1 perfmon local box mask MSR.		Package
Register Address: E5CH, 3676	MSR_M1_PMON_MM_CONFIG	
Uncore M-box 1 perfmon local box address match/mask config MSR.		Package
Register Address: E5DH, 3677	MSR_M1_PMON_ADDR_MATCH	
Uncore M-box 1 perfmon local box address match MSR.		Package
Register Address: E5EH, 3678	MSR_M1_PMON_ADDR_MASK	
Uncore M-box 1 perfmon local box address mask MSR.		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 20.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
63:48	Reserved.	
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
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.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: F40H, 3904	MSR_C8_PMON_BOX_CTRL	
Uncore C-box 8 perfmon local box control MSR.		Package
Register Address: F41H, 3905	MSR_C8_PMON_BOX_STATUS	
Uncore C-box 8 perfmon local box status MSR.		Package
Register Address: F42H, 3906	MSR_C8_PMON_BOX_OVF_CTRL	
Uncore C-box 8 perfmon local box overflow control MSR.		Package
Register Address: F50H, 3920	MSR_C8_PMON_EVNT_SEL0	
Uncore C-box 8 perfmon event select MSR.		Package
Register Address: F51H, 3921	MSR_C8_PMON_CTR0	
Uncore C-box 8 perfmon counter MSR.		Package
Register Address: F52H, 3922	MSR_C8_PMON_EVNT_SEL1	
Uncore C-box 8 perfmon event select MSR.		Package
Register Address: F53H, 3923	MSR_C8_PMON_CTR1	
Uncore C-box 8 perfmon counter MSR.		Package
Register Address: F54H, 3924	MSR_C8_PMON_EVNT_SEL2	
Uncore C-box 8 perfmon event select MSR.		Package
Register Address: F55H, 3925	MSR_C8_PMON_CTR2	
Uncore C-box 8 perfmon counter MSR.		Package
Register Address: F56H, 3926	MSR_C8_PMON_EVNT_SEL3	
Uncore C-box 8 perfmon event select MSR.		Package

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: F57H, 3927	MSR_C8_PMON_CTR3	
Uncore C-box 8 perfmon counter MSR.		Package
Register Address: F58H, 3928	MSR_C8_PMON_EVNT_SEL4	
Uncore C-box 8 perfmon event select MSR.		Package
Register Address: F59H, 3929	MSR_C8_PMON_CTR4	
Uncore C-box 8 perfmon counter MSR.		Package
Register Address: F5AH, 3930	MSR_C8_PMON_EVNT_SEL5	
Uncore C-box 8 perfmon event select MSR.		Package
Register Address: F5BH, 3931	MSR_C8_PMON_CTR5	
Uncore C-box 8 perfmon counter MSR.		Package
Register Address: FC0H, 4032	MSR_C9_PMON_BOX_CTRL	
Uncore C-box 9 perfmon local box control MSR.		Package
Register Address: FC1H, 4033	MSR_C9_PMON_BOX_STATUS	
Uncore C-box 9 perfmon local box status MSR.		Package
Register Address: FC2H, 4034	MSR_C9_PMON_BOX_OVF_CTRL	
Uncore C-box 9 perfmon local box overflow control MSR.		Package
Register Address: FD0H, 4048	MSR_C9_PMON_EVNT_SEL0	
Uncore C-box 9 perfmon event select MSR.		Package
Register Address: FD1H, 4049	MSR_C9_PMON_CTR0	
Uncore C-box 9 perfmon counter MSR.		Package
Register Address: FD2H, 4050	MSR_C9_PMON_EVNT_SEL1	
Uncore C-box 9 perfmon event select MSR.		Package
Register Address: FD3H, 4051	MSR_C9_PMON_CTR1	
Uncore C-box 9 perfmon counter MSR.		Package
Register Address: FD4H, 4052	MSR_C9_PMON_EVNT_SEL2	
Uncore C-box 9 perfmon event select MSR.		Package
Register Address: FD5H, 4053	MSR_C9_PMON_CTR2	
Uncore C-box 9 perfmon counter MSR.		Package
Register Address: FD6H, 4054	MSR_C9_PMON_EVNT_SEL3	
Uncore C-box 9 perfmon event select MSR.		Package
Register Address: FD7H, 4055	MSR_C9_PMON_CTR3	
Uncore C-box 9 perfmon counter MSR.		Package
Register Address: FD8H, 4056	MSR_C9_PMON_EVNT_SEL4	
Uncore C-box 9 perfmon event select MSR.		Package
Register Address: FD9H, 4057	MSR_C9_PMON_CTR4	
Uncore C-box 9 perfmon counter MSR.		Package
Register Address: FDAH, 4058	MSR_C9_PMON_EVNT_SEL5	

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-box 9 perfmon event select MSR.		Package
Register Address: FDBH, 4059	MSR_C9_PMON_CTR5	
Uncore C-box 9 perfmon counter MSR.		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 18.17, "Time-Stamp Counter," and see Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Count SMIs.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: C5H, 197	IA32_PMC4	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C6H, 198	IA32_PMC5	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C7H, 199	IA32_PMC6	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C8H, 200	IA32_PMC7	
Performance Counter Register (if core not shared by threads)		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	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.	Package

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
29	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.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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.	
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.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org.		Core
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.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
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.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
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.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Thread
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Thread
Register Address: 18AH, 394	IA32_PERFEVTSEL4	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 4.		Core
Register Address: 18BH, 395	IA32_PERFEVTSEL5	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 5.		Core
Register Address: 18CH, 396	IA32_PERFEVTSEL6	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 6.		Core
Register Address: 18DH, 397	IA32_PERFEVTSEL7	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 7.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Table 2-2.		Package
15:0	Current Performance State Value	
63:16	Reserved.	
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Package
47:32	Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2^13).	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
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.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) 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.	
Register Address: 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.	Thread
6:1	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	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.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Unique
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.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	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.	Core
1	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).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	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.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org.		
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."		Thread
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.	
Register Address: 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 MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
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	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
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.		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
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.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
See http://biosbits.org.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MC0_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
Always 0 (CMCI not supported).		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2 and Section 18.4.1, "IA32_DEBUGCTL MSR."		Thread
5:0	LBR Format See Table 2-2.	
6	PEBS Record Format.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 20.6.2.2, "Global Counter Control Facilities."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Core
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Core
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Core
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Core
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
60:35	Reserved.	
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2 and Section 20.6.2.2, "Global Counter Control Facilities."		Thread
0	Set 1 to enable PMC0 to count.	Thread
1	Set 1 to enable PMC1 to count.	Thread
2	Set 1 to enable PMC2 to count.	Thread
3	Set 1 to enable PMC3 to count.	Thread
4	Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7).	Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
31:8	Reserved.	
32	Set 1 to enable FixedCtr0 to count.	Thread
33	Set 1 to enable FixedCtr1 to count.	Thread
34	Set 1 to enable FixedCtr2 to count.	Thread
63:35	Reserved.	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 20.6.2.2, "Global Counter Control Facilities."		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Core
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
60:35	Reserved.	
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 20.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
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)	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 20.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 3FEH, 1022	MSR_CORE_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
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.	
Register Address: 400H, 1024	IA32_MC0_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Core
Register Address: 402H, 1026	IA32_MC0_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 403H, 1027	IA32_MC0_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
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.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:2	Reserved.	
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2		Thread
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLS	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2		Thread
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2		Thread
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Thread
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Thread
Register Address: 4C5H, 1221	IA32_A_PMC4	
See Table 2-2.		Core
Register Address: 4C6H, 1222	IA32_A_PMC5	
See Table 2-2.		Core
Register Address: 4C7H, 1223	IA32_A_PMC6	
See Table 2-2.		Core
Register Address: 4C8H, 1224	IA32_A_PMC7	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 20.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 15.10.1, "RAPL Interfaces."		Package
Register Address: 60AH, 1546	MSR_PKGC3_IRTL	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
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-sate management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKGC6_IRTL	
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.		Package
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-sate management.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PP0_POWER_LIMIT	
PP0 RAPL Power Limit Control (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.9.1 and record format in Section 18.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
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.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
See Table 2-2.		Thread
Register Address: 802H—83FH, 2050—2111	X2APIC MSRs	
See Table 2-2.		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 18.17.2, "IA32_TSC_AUX Register and RDTSCP Support."		Thread

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 60CH, 1548	MSR_PKGC7_IRTL	
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.		Package
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-sate management.	
63:16	Reserved.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 63AH, 1594	MSR_PP0_POLICY	

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
PP0 Balance Policy (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
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.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSEL0	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSEL0	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFEVTSEL2	
Uncore C-Box 0, Counter 2 Event Select MSR		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFEVTSEL3	
Uncore C-Box 0, Counter 3 Event Select MSR		Package
Register Address: 705H, 1797	MSR_UNC_CBO_0_UNIT_STATUS	
Uncore C-Box 0, Unit Status for Counter 0-3		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 708H, 1800	MSR_UNC_CBO_0_PERFCTR2	
Uncore C-Box 0, Performance Counter 2		Package
Register Address: 709H, 1801	MSR_UNC_CBO_0_PERFCTR3	
Uncore C-Box 0, Performance Counter 3		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSEL0	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_1_PERFEVTSEL2	
Uncore C-Box 1, Counter 2 Event Select MSR		Package
Register Address: 713H, 1811	MSR_UNC_CBO_1_PERFEVTSEL3	
Uncore C-Box 1, Counter 3 Event Select MSR		Package
Register Address: 715H, 1813	MSR_UNC_CBO_1_UNIT_STATUS	
Uncore C-Box 1, Unit Status for Counter 0-3		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_1_PERFCTR2	
Uncore C-Box 1, Performance Counter 2		Package
Register Address: 719H, 1817	MSR_UNC_CBO_1_PERFCTR3	
Uncore C-Box 1, Performance Counter 3		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSEL0	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 722H, 1826	MSR_UNC_CBO_2_PERFEVTSEL2	
Uncore C-Box 2, Counter 2 Event Select MSR		Package
Register Address: 723H, 1827	MSR_UNC_CBO_2_PERFEVTSEL3	
Uncore C-Box 2, Counter 3 Event Select MSR		Package
Register Address: 725H, 1829	MSR_UNC_CBO_2_UNIT_STATUS	
Uncore C-Box 2, Unit Status for Counter 0-3		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_3_PERFCTR2	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 729H, 1833	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSEL0	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_3_PERFEVTSEL2	
Uncore C-Box 3, Counter 2 Event Select MSR		Package
Register Address: 733H, 1843	MSR_UNC_CBO_3_PERFEVTSEL3	
Uncore C-Box 3, counter 3 Event Select MSR		Package
Register Address: 735H, 1845	MSR_UNC_CBO_3_UNIT_STATUS	
Uncore C-Box 3, Unit Status for Counter 0-3		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_3_PERFCTR2	
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 739H, 1849	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 740H, 1856	MSR_UNC_CBO_4_PERFEVTSEL0	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 741H, 1857	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 742H, 1858	MSR_UNC_CBO_4_PERFEVTSEL2	
Uncore C-Box 4, Counter 2 Event Select MSR		Package
Register Address: 743H, 1859	MSR_UNC_CBO_4_PERFEVTSEL3	
Uncore C-Box 4, Counter 3 Event Select MSR		Package
Register Address: 745H, 1861	MSR_UNC_CBO_4_UNIT_STATUS	
Uncore C-Box 4, Unit status for Counter 0-3		Package
Register Address: 746H, 1862	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 747H, 1863	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 748H, 1864	MSR_UNC_CBO_4_PERFCTR2	
Uncore C-Box 4, Performance Counter 2		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 749H, 1865	MSR_UNC_CBO_4_PERFCTR3	
Uncore C-Box 4, Performance Counter 3		Package

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. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
	MC Bank Error Configuration (R/W)	Package
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.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 cores active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 cores active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 cores active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 cores active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 cores active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 cores active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 cores active.	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 39CH, 924	MSR_PEBS_NUM_ALT	
ENABLE_PEBS_NUM_ALT (R/W)		Package
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.	
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 16.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: C08H, 3080	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-box UCLK Fixed Counter Control		Package
Register Address: C09H, 3081	MSR_U_PMON_UCLK_FIXED_CTR	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore U-box UCLK Fixed Counter		Package
Register Address: C10H, 3088	MSR_U_PMON_EVNTSEL0	
Uncore U-box Perfmon Event Select for U-box Counter 0		Package
Register Address: C11H, 3089	MSR_U_PMON_EVNTSEL1	
Uncore U-box Perfmon Event Select for U-box Counter 1		Package
Register Address: C16H, 3094	MSR_U_PMON_CTR0	
Uncore U-box Perfmon Counter 0		Package
Register Address: C17H, 3095	MSR_U_PMON_CTR1	
Uncore U-box Perfmon Counter 1		Package
Register Address: C24H, 3108	MSR_PCU_PMON_BOX_CTL	
Uncore PCU Perfmon for PCU-box-wide Control		Package
Register Address: C30H, 3120	MSR_PCU_PMON_EVNTSEL0	
Uncore PCU Perfmon Event Select for PCU Counter 0		Package
Register Address: C31H, 3121	MSR_PCU_PMON_EVNTSEL1	
Uncore PCU Perfmon Event Select for PCU Counter 1		Package
Register Address: C32H, 3122	MSR_PCU_PMON_EVNTSEL2	
Uncore PCU Perfmon Event Select for PCU Counter 2		Package
Register Address: C33H, 3123	MSR_PCU_PMON_EVNTSEL3	
Uncore PCU Perfmon Event Select for PCU Counter 3		Package
Register Address: C34H, 3124	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU Perfmon box-wide Filter		Package
Register Address: C36H, 3126	MSR_PCU_PMON_CTR0	
Uncore PCU Perfmon Counter 0		Package
Register Address: C37H, 3127	MSR_PCU_PMON_CTR1	
Uncore PCU Perfmon Counter 1		Package
Register Address: C38H, 3128	MSR_PCU_PMON_CTR2	
Uncore PCU Perfmon Counter 2		Package
Register Address: C39H, 3129	MSR_PCU_PMON_CTR3	
Uncore PCU Perfmon Counter 3		Package
Register Address: D04H, 3332	MSR_C0_PMON_BOX_CTL	
Uncore C-box 0 Perfmon Local Box Wide Control		Package
Register Address: D10H, 3344	MSR_C0_PMON_EVNTSEL0	
Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 0		Package
Register Address: D11H, 3345	MSR_C0_PMON_EVNTSEL1	
Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 1		Package
Register Address: D12H, 3346	MSR_C0_PMON_EVNTSEL2	
Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 2		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: D13H, 3347	MSR_C0_PMON_EVNTSEL3	
Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 3		Package
Register Address: D14H, 3348	MSR_C0_PMON_BOX_FILTER	
Uncore C-box 0 Perfmon Box Wide Filter		Package
Register Address: D16H, 3350	MSR_C0_PMON_CTR0	
Uncore C-box 0 Perfmon Counter 0		Package
Register Address: D17H, 3351	MSR_C0_PMON_CTR1	
Uncore C-box 0 Perfmon Counter 1		Package
Register Address: D18H, 3352	MSR_C0_PMON_CTR2	
Uncore C-box 0 Perfmon Counter 2		Package
Register Address: D19H, 3353	MSR_C0_PMON_CTR3	
Uncore C-box 0 Perfmon Counter 3		Package
Register Address: D24H, 3364	MSR_C1_PMON_BOX_CTL	
Uncore C-box 1 Perfmon Local Box Wide Control		Package
Register Address: D30H, 3376	MSR_C1_PMON_EVNTSEL0	
Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 0		Package
Register Address: D31H, 3377	MSR_C1_PMON_EVNTSEL1	
Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 1		Package
Register Address: D32H, 3378	MSR_C1_PMON_EVNTSEL2	
Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 2		Package
Register Address: D33H, 3379	MSR_C1_PMON_EVNTSEL3	
Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 3		Package
Register Address: D34H, 3380	MSR_C1_PMON_BOX_FILTER	
Uncore C-box 1 Perfmon Box Wide Filter		Package
Register Address: D36H, 3382	MSR_C1_PMON_CTR0	
Uncore C-box 1 Perfmon Counter 0		Package
Register Address: D37H, 3383	MSR_C1_PMON_CTR1	
Uncore C-box 1 Perfmon Counter 1		Package
Register Address: D38H, 3384	MSR_C1_PMON_CTR2	
Uncore C-box 1 Perfmon Counter 2		Package
Register Address: D39H, 3385	MSR_C1_PMON_CTR3	
Uncore C-box 1 Perfmon Counter 3		Package
Register Address: D44H, 3396	MSR_C2_PMON_BOX_CTL	
Uncore C-box 2 Perfmon Local Box Wide Control		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVNTSEL0	
Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 0		Package
Register Address: D51H, 3409	MSR_C2_PMON_EVNTSEL1	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 1		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNTSEL2	
Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 2		Package
Register Address: D53H, 3411	MSR_C2_PMON_EVNTSEL3	
Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 3		Package
Register Address: D54H, 3412	MSR_C2_PMON_BOX_FILTER	
Uncore C-box 2 Perfmon Box Wide Filter		Package
Register Address: D56H, 3414	MSR_C2_PMON_CTR0	
Uncore C-box 2 Perfmon Counter 0		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR1	
Uncore C-box 2 Perfmon Counter 1		Package
Register Address: D58H, 3416	MSR_C2_PMON_CTR2	
Uncore C-box 2 Perfmon Counter 2		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR3	
Uncore C-box 2 Perfmon Counter 3		Package
Register Address: D64H, 3428	MSR_C3_PMON_BOX_CTL	
Uncore C-box 3 Perfmon Local Box Wide Control		Package
Register Address: D70H, 3440	MSR_C3_PMON_EVNTSEL0	
Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 0		Package
Register Address: D71H, 3441	MSR_C3_PMON_EVNTSEL1	
Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 1		Package
Register Address: D72H, 3442	MSR_C3_PMON_EVNTSEL2	
Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 2		Package
Register Address: D73H, 3443	MSR_C3_PMON_EVNTSEL3	
Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 3		Package
Register Address: D74H, 3444	MSR_C3_PMON_BOX_FILTER	
Uncore C-box 3 Perfmon Box Wide Filter		Package
Register Address: D76H, 3446	MSR_C3_PMON_CTR0	
Uncore C-box 3 Perfmon Counter 0		Package
Register Address: D77H, 3447	MSR_C3_PMON_CTR1	
Uncore C-box 3 Perfmon Counter 1		Package
Register Address: D78H, 3448	MSR_C3_PMON_CTR2	
Uncore C-box 3 Perfmon Counter 2		Package
Register Address: D79H, 3449	MSR_C3_PMON_CTR3	
Uncore C-box 3 Perfmon Counter 3		Package
Register Address: D84H, 3460	MSR_C4_PMON_BOX_CTL	
Uncore C-box 4 Perfmon Local Box Wide Control		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: D90H, 3472	MSR_C4_PMON_EVNTSEL0	
Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 0		Package
Register Address: D91H, 3473	MSR_C4_PMON_EVNTSEL1	
Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 1		Package
Register Address: D92H, 3474	MSR_C4_PMON_EVNTSEL2	
Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 2		Package
Register Address: D93H, 3475	MSR_C4_PMON_EVNTSEL3	
Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 3		Package
Register Address: D94H, 3476	MSR_C4_PMON_BOX_FILTER	
Uncore C-box 4 Perfmon Box Wide Filter		Package
Register Address: D96H, 3478	MSR_C4_PMON_CTR0	
Uncore C-box 4 Perfmon Counter 0		Package
Register Address: D97H, 3479	MSR_C4_PMON_CTR1	
Uncore C-box 4 Perfmon Counter 1		Package
Register Address: D98H, 3480	MSR_C4_PMON_CTR2	
Uncore C-box 4 Perfmon Counter 2		Package
Register Address: D99H, 3481	MSR_C4_PMON_CTR3	
Uncore C-box 4 Perfmon Counter 3		Package
Register Address: DA4H, 3492	MSR_C5_PMON_BOX_CTL	
Uncore C-box 5 Perfmon Local Box Wide Control		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNTSEL0	
Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 0		Package
Register Address: DB1H, 3505	MSR_C5_PMON_EVNTSEL1	
Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 1		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNTSEL2	
Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 2		Package
Register Address: DB3H, 3507	MSR_C5_PMON_EVNTSEL3	
Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 3		Package
Register Address: DB4H, 3508	MSR_C5_PMON_BOX_FILTER	
Uncore C-box 5 Perfmon Box Wide Filter		Package
Register Address: DB6H, 3510	MSR_C5_PMON_CTR0	
Uncore C-box 5 Perfmon Counter 0		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR1	
Uncore C-box 5 Perfmon Counter 1		Package
Register Address: DB8H, 3512	MSR_C5_PMON_CTR2	
Uncore C-box 5 Perfmon Counter 2		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR3	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-box 5 Perfmon Counter 3		Package
Register Address: DC4H, 3524	MSR_C6_PMON_BOX_CTL	
Uncore C-box 6 Perfmon Local Box Wide Control		Package
Register Address: DD0H, 3536	MSR_C6_PMON_EVNTSEL0	
Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 0		Package
Register Address: DD1H, 3537	MSR_C6_PMON_EVNTSEL1	
Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 1		Package
Register Address: DD2H, 3538	MSR_C6_PMON_EVNTSEL2	
Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 2		Package
Register Address: DD3H, 3539	MSR_C6_PMON_EVNTSEL3	
Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 3		Package
Register Address: DD4H, 3540	MSR_C6_PMON_BOX_FILTER	
Uncore C-box 6 Perfmon Box Wide Filter		Package
Register Address: DD6H, 3542	MSR_C6_PMON_CTR0	
Uncore C-box 6 Perfmon Counter 0		Package
Register Address: DD7H, 3543	MSR_C6_PMON_CTR1	
Uncore C-box 6 Perfmon Counter 1		Package
Register Address: DD8H, 3544	MSR_C6_PMON_CTR2	
Uncore C-box 6 Perfmon Counter 2		Package
Register Address: DD9H, 3545	MSR_C6_PMON_CTR3	
Uncore C-box 6 Perfmon Counter 3		Package
Register Address: DE4H, 3556	MSR_C7_PMON_BOX_CTL	
Uncore C-box 7 Perfmon Local Box Wide Control		Package
Register Address: DF0H, 3568	MSR_C7_PMON_EVNTSEL0	
Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 0		Package
Register Address: DF1H, 3569	MSR_C7_PMON_EVNTSEL1	
Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 1		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVNTSEL2	
Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 2		Package
Register Address: DF3H, 3571	MSR_C7_PMON_EVNTSEL3	
Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 3		Package
Register Address: DF4H, 3572	MSR_C7_PMON_BOX_FILTER	
Uncore C-box 7 Perfmon Box Wide Filter		Package
Register Address: DF6H, 3574	MSR_C7_PMON_CTR0	
Uncore C-box 7 Perfmon Counter 0		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTR1	
Uncore C-box 7 Perfmon Counter 1		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: DF8H, 3576	MSR_C7_PMON_CTR2	
Uncore C-box 7 Perfmon Counter 2		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR3	
Uncore C-box 7 Perfmon Counter 3		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	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.	Package
29	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.	Package
31:30	Reserved.	
32	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.	Package
34:33	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	Package
39:35	Reserved.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
	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.	Core
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.	
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.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
	ConfigTDP Level 1 ratio and power level (R/O)	Package
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.	
47	Reserved.	
62:48	PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.	
63	Reserved.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O)		Package
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.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
62:48	PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.	
63	Reserved.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
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.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
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 (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
22:16	Reserved.	
23	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.	Package
27:24	Reserved.	
28	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.	Package
29	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.	Package
30	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.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
63:16	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
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.	
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
MC Bank Error Configuration (R/W)		Package
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.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
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.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
63:32	Reserved.	
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 296H, 662	IA32_MC22_CTL2	
See Table 2-2.		Package
Register Address: 297H, 663	IA32_MC23_CTL2IA32_MC23_CTL2	
See Table 2-2.		Package
Register Address: 298H, 664	IA32_MC24_CTL2	
See Table 2-2.		Package
Register Address: 299H, 665	IA32_MC25_CTL2	
See Table 2-2.		Package
Register Address: 29AH, 666	IA32_MC26_CTL2	
See Table 2-2.		Package
Register Address: 29BH, 667	IA32_MC27_CTL2	
See Table 2-2.		Package
Register Address: 29CH, 668	IA32_MC28_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
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.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
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.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 416H, 1046	IA32_MC5_ADDR	
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.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
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.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
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.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
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.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
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.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
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.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
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.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
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.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
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.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
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.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
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.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
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.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
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.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 423H, 1059	IA32_MC8_MISC	
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.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
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.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
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.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
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.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
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.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
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.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
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.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
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.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
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.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 430H, 1072	IA32_MC12_CTL	
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.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
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.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
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.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
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.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
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.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
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.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
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.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
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.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
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.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
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.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
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.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
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.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
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.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 43DH, 1085	IA32_MC15_STATUS	
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.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
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.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
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.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
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.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
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.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
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.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
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.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
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.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
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.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
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.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
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.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
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.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
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.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 44AH, 1098	IA32_MC18_ADDR	
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.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
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.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
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.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
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.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
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.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
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.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
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.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
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.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
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.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
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.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	
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.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
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.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
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.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 457H, 1111	IA32_MC21_MISC	
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.		Package
Register Address: 458H, 1112	IA32_MC22_CTL	
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.		Package
Register Address: 459H, 1113	IA32_MC22_STATUS	
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.		Package
Register Address: 45AH, 1114	IA32_MC22_ADDR	
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.		Package
Register Address: 45BH, 1115	IA32_MC22_MISC	
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.		Package
Register Address: 45CH, 1116	IA32_MC23_CTL	
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.		Package
Register Address: 45DH, 1117	IA32_MC23_STATUS	
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.		Package
Register Address: 45EH, 1118	IA32_MC23_ADDR	
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.		Package
Register Address: 45FH, 1119	IA32_MC23_MISC	
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.		Package
Register Address: 460H, 1120	IA32_MC24_CTL	
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.		Package
Register Address: 461H, 1121	IA32_MC24_STATUS	
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.		Package
Register Address: 462H, 1122	IA32_MC24_ADDR	
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.		Package
Register Address: 463H, 1123	IA32_MC24_MISC	
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.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 464H, 1124	IA32_MC25_CTL	
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.		Package
Register Address: 465H, 1125	IA32_MC25_STATUS	
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.		Package
Register Address: 466H, 1126	IA32_MC25_ADDR	
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.		Package
Register Address: 467H, 1127	IA32_MC2MISC	
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.		Package
Register Address: 468H, 1128	IA32_MC26_CTL	
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.		Package
Register Address: 469H, 1129	IA32_MC26_STATUS	
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.		Package
Register Address: 46AH, 1130	IA32_MC26_ADDR	
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.		Package
Register Address: 46BH, 1131	IA32_MC26_MISC	
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.		Package
Register Address: 46CH, 1132	IA32_MC27_CTL	
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.		Package
Register Address: 46DH, 1133	IA32_MC27_STATUS	
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.		Package
Register Address: 46EH, 1134	IA32_MC27_ADDR	
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.		Package
Register Address: 46FH, 1135	IA32_MC27_MISC	
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.		Package
Register Address: 470H, 1136	IA32_MC28_CTL	
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.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 471H, 1137	IA32_MC28_STATUS	
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.		Package
Register Address: 472H, 1138	IA32_MC28_ADDR	
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.		Package
Register Address: 473H, 1139	IA32_MC28_MISC	
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.		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
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.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
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.	
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status (R/W)		Thread
0	RIPV	
1	EIPV	
2	MCIP	
3	LMCE Signaled	
63:4	Reserved.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
62:56	Reserved.	
63	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).	Package
Register Address: 29DH, 669	IA32_MC29_CTL2	
See Table 2-2.		Package
Register Address: 29EH, 670	IA32_MC30_CTL2	
See Table 2-2.		Package
Register Address: 29FH, 671	IA32_MC31_CTL2	
See Table 2-2.		Package
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 20.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
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.	
Register Address: 41BH, 1051	IA32_MC6_MISC	
Misc MAC Information of Integrated I/O (R/O) See Section 16.3.2.4.		Package
5:0	Recoverable Address LSB	
8:6	Address Mode	
15:9	Reserved.	
31:16	PCI Express Requestor ID	
39:32	PCI Express Segment Number	
63:32	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
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.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
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.		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 476H, 1142	IA32_MC29_ADDR	
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.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
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.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
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.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
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.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
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.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
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.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
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.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
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.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
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.		Package
Register Address: 47FH, 1147	IA32_MC31_MISC	
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.		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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: C00H, 3072	MSR_PMON_GLOBAL_CTL	
Uncore Perfmon Per-Socket Global Control		Package
Register Address: C01H, 3073	MSR_PMON_GLOBAL_STATUS	
Uncore Perfmon Per-Socket Global Status		Package
Register Address: C06H, 3078	MSR_PMON_GLOBAL_CONFIG	
Uncore Perfmon Per-Socket Global Configuration		Package
Register Address: C15H, 3093	MSR_U_PMON_BOX_STATUS	
Uncore U-box Perfmon U-Box Wide Status		Package
Register Address: C35H, 3125	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU Perfmon Box Wide Status		Package
Register Address: D1AH, 3354	MSR_C0_PMON_BOX_FILTER1	
Uncore C-Box 0 Perfmon Box Wide Filter1		Package
Register Address: D3AH, 3386	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 Perfmon Box Wide Filter1		Package
Register Address: D5AH, 3418	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 Perfmon Box Wide Filter1		Package
Register Address: D7AH, 3450	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 Perfmon Box Wide Filter1		Package
Register Address: D9AH, 3482	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 Perfmon Box Wide Filter1		Package
Register Address: DBAH, 3514	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 Perfmon Box Wide Filter1		Package
Register Address: DDAH, 3546	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 Perfmon Box Wide Filter1		Package
Register Address: DFAH, 3578	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 Perfmon Box Wide Filter1		Package
Register Address: E04H, 3588	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 Perfmon Local Box Wide Control		Package
Register Address: E10H, 3600	MSR_C8_PMON_EVNTSEL0	
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0		Package
Register Address: E11H, 3601	MSR_C8_PMON_EVNTSEL1	
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1		Package
Register Address: E12H, 3602	MSR_C8_PMON_EVNTSEL2	
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2		Package
Register Address: E13H, 3603	MSR_C8_PMON_EVNTSEL3	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3		Package
Register Address: E14H, 3604	MSR_C8_PMON_BOX_FILTER	
Uncore C-Box 8 Perfmon Box Wide Filter		Package
Register Address: E16H, 3606	MSR_C8_PMON_CTR0	
Uncore C-Box 8 Perfmon Counter 0		Package
Register Address: E17H, 3607	MSR_C8_PMON_CTR1	
Uncore C-Box 8 Perfmon Counter 1		Package
Register Address: E18H, 3608	MSR_C8_PMON_CTR2	
Uncore C-Box 8 Perfmon Counter 2		Package
Register Address: E19H, 3609	MSR_C8_PMON_CTR3	
Uncore C-Box 8 Perfmon Counter 3		Package
Register Address: E1AH, 3610	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 Perfmon Box Wide Filter1		Package
Register Address: E24H, 3620	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 Perfmon Local Box Wide Control		Package
Register Address: E30H, 3632	MSR_C9_PMON_EVNTSEL0	
Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 0		Package
Register Address: E31H, 3633	MSR_C9_PMON_EVNTSEL1	
Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 1		Package
Register Address: E32H, 3634	MSR_C9_PMON_EVNTSEL2	
Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 2		Package
Register Address: E33H, 3635	MSR_C9_PMON_EVNTSEL3	
Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 3		Package
Register Address: E34H, 3636	MSR_C9_PMON_BOX_FILTER	
Uncore C-Box 9 Perfmon Box Wide Filter		Package
Register Address: E36H, 3638	MSR_C9_PMON_CTR0	
Uncore C-Box 9 Perfmon Counter 0		Package
Register Address: E37H, 3639	MSR_C9_PMON_CTR1	
Uncore C-Box 9 Perfmon Counter 1		Package
Register Address: E38H, 3640	MSR_C9_PMON_CTR2	
Uncore C-Box 9 Perfmon Counter 2		Package
Register Address: E39H, 3641	MSR_C9_PMON_CTR3	
Uncore C-Box 9 Perfmon Counter 3		Package
Register Address: E3AH, 3642	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 Perfmon Box Wide Filter1		Package
Register Address: E44H, 3652	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 Perfmon Local Box Wide Control		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E50H, 3664	MSR_C10_PMON_EVNTSEL0	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0		Package
Register Address: E51H, 3665	MSR_C10_PMON_EVNTSEL1	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1		Package
Register Address: E52H, 3666	MSR_C10_PMON_EVNTSEL2	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2		Package
Register Address: E53H, 3667	MSR_C10_PMON_EVNTSEL3	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3		Package
Register Address: E54H, 3668	MSR_C10_PMON_BOX_FILTER	
Uncore C-Box 10 Perfmon Box Wide Filter		Package
Register Address: E56H, 3670	MSR_C10_PMON_CTR0	
Uncore C-Box 10 Perfmon Counter 0		Package
Register Address: E57H, 3671	MSR_C10_PMON_CTR1	
Uncore C-Box 10 Perfmon Counter 1		Package
Register Address: E58H, 3672	MSR_C10_PMON_CTR2	
Uncore C-Box 10 Perfmon Counter 2		Package
Register Address: E59H, 3673	MSR_C10_PMON_CTR3	
Uncore C-Box 10 Perfmon Counter 3		Package
Register Address: E5AH, 3674	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 Perfmon Box Wide Filter1		Package
Register Address: E64H, 3684	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 Perfmon Local Box Wide Control		Package
Register Address: E70H, 3696	MSR_C11_PMON_EVNTSEL0	
Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0		Package
Register Address: E71H, 3697	MSR_C11_PMON_EVNTSEL1	
Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1		Package
Register Address: E72H, 3698	MSR_C11_PMON_EVNTSEL2	
Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2		Package
Register Address: E73H, 3699	MSR_C11_PMON_EVNTSEL3	
Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 3		Package
Register Address: E74H, 3700	MSR_C11_PMON_BOX_FILTER	
Uncore C-Box 11 Perfmon Box Wide Filter		Package
Register Address: E76H, 3702	MSR_C11_PMON_CTR0	
Uncore C-Box 11 Perfmon Counter 0		Package
Register Address: E77H, 3703	MSR_C11_PMON_CTR1	
Uncore C-Box 11 Perfmon Counter 1		Package
Register Address: E78H, 3704	MSR_C11_PMON_CTR2	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 11 Perfmon Counter 2		Package
Register Address: E79H, 3705	MSR_C11_PMON_CTR3	
Uncore C-Box 11 Perfmon Counter 3		Package
Register Address: E7AH, 3706	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 Perfmon Box Wide Filter1		Package
Register Address: E84H, 3716	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 Perfmon Local Box Wide Control		Package
Register Address: E90H, 3728	MSR_C12_PMON_EVNTSEL0	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0		Package
Register Address: E91H, 3729	MSR_C12_PMON_EVNTSEL1	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1		Package
Register Address: E92H, 3730	MSR_C12_PMON_EVNTSEL2	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2		Package
Register Address: E93H, 3731	MSR_C12_PMON_EVNTSEL3	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3		Package
Register Address: E94H, 3732	MSR_C12_PMON_BOX_FILTER	
Uncore C-Box 12 Perfmon Box Wide Filter		Package
Register Address: E96H, 3734	MSR_C12_PMON_CTR0	
Uncore C-Box 12 Perfmon Counter 0		Package
Register Address: E97H, 3735	MSR_C12_PMON_CTR1	
Uncore C-Box 12 Perfmon Counter 1		Package
Register Address: E98H, 3736	MSR_C12_PMON_CTR2	
Uncore C-Box 12 Perfmon Counter 2		Package
Register Address: E99H, 3737	MSR_C12_PMON_CTR3	
Uncore C-Box 12 Perfmon Counter 3		Package
Register Address: E9AH, 3738	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 Perfmon Box Wide Filter1		Package
Register Address: EA4H, 3748	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 Perfmon Local Box Wide Control		Package
Register Address: EB0H, 3760	MSR_C13_PMON_EVNTSEL0	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0		Package
Register Address: EB1H, 3761	MSR_C13_PMON_EVNTSEL1	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1		Package
Register Address: EB2H, 3762	MSR_C13_PMON_EVNTSEL2	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2		Package
Register Address: EB3H, 3763	MSR_C13_PMON_EVNTSEL3	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: EB4H, 3764	MSR_C13_PMON_BOX_FILTER	
Uncore C-Box 13 Perfmon Box Wide Filter		Package
Register Address: EB6H, 3766	MSR_C13_PMON_CTR0	
Uncore C-Box 13 Perfmon Counter 0		Package
Register Address: EB7H, 3767	MSR_C13_PMON_CTR1	
Uncore C-Box 13 Perfmon Counter 1		Package
Register Address: EB8H, 3768	MSR_C13_PMON_CTR2	
Uncore C-Box 13 Perfmon Counter 2		Package
Register Address: EB9H, 3769	MSR_C13_PMON_CTR3	
Uncore C-Box 13 Perfmon Counter 3		Package
Register Address: EBAH, 3770	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 Perfmon Box Wide Filter1		Package
Register Address: EC4H, 3780	MSR_C14_PMON_BOX_CTL	
Uncore C-Box 14 Perfmon Local Box Wide Control		Package
Register Address: ED0H, 3792	MSR_C14_PMON_EVNTSEL0	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0		Package
Register Address: ED1H, 3793	MSR_C14_PMON_EVNTSEL1	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1		Package
Register Address: ED2H, 3794	MSR_C14_PMON_EVNTSEL2	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2		Package
Register Address: ED3H, 3795	MSR_C14_PMON_EVNTSEL3	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3		Package
Register Address: ED4H, 3796	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 Perfmon Box Wide Filter		Package
Register Address: ED6H, 3798	MSR_C14_PMON_CTR0	
Uncore C-Box 14 Perfmon Counter 0		Package
Register Address: ED7H, 3799	MSR_C14_PMON_CTR1	
Uncore C-Box 14 Perfmon Counter 1		Package
Register Address: ED8H, 3800	MSR_C14_PMON_CTR2	
Uncore C-Box 14 Perfmon Counter 2		Package
Register Address: ED9H, 3801	MSR_C14_PMON_CTR3	
Uncore C-Box 14 Perfmon Counter 3		Package
Register Address: EDAH, 3802	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 Perfmon Box Wide Filter1		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	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.	Package
29	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.	Package
31:30	Reserved.	
32	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.	Package
34:33	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.	Package
39:35	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 186H, 390	IA32_PERFEVTSEL0	
Performance Event Select for Counter 0 (R/W) Supports all fields described inTable 2-2 and the fields below.		Thread
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.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
Performance Event Select for Counter 1 (R/W) Supports all fields described inTable 2-2 and the fields below.		Thread
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.	
Register Address: 188H, 392	IA32_PERFEVTSEL2	
Performance Event Select for Counter 2 (R/W) Supports all fields described inTable 2-2 and the fields below.		Thread
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.	
Register Address: 189H, 393	IA32_PERFEVTSEL3	
Performance Event Select for Counter 3 (R/W) Supports all fields described inTable 2-2 and the fields below.		Thread
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.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W)		Thread
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 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
8	FAR_BRANCH	
9	EN_CALL_STACK	
63:9	Reserved.	
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
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.	
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Thread
Register Address: 60BH, 1548	MSR_PKGC_IRTL1	
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.		Package
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-sate management.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKGC_IRTL2	
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.		Package
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-sate management.	
63:16	Reserved.	
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 Ratio and Power Level (R/O)		Package
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.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 Ratio and Power Level (R/O)		Package
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.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
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.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
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.	
Register Address: C80H, 3200	IA32_DEBUG_INTERFACE	
Silicon Debug Feature Control (R/W) See Table 2-2.		Package

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
63:29	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
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.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
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.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Encoded number of C-Box, derive value by "-1".	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSEL0	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
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.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
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.		Package
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.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package
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 0FFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 15.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
12:8	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).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 15.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
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.	
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.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
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.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
9	Reserved.	
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.	
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.	
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.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSEL0	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSEL0	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSEL0	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1824	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSEL0	
Uncore C-Box 3, Counter 0 Event Select MSR		Package

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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.	

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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 630H, 1584	MSR_PKG_C8_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 631H, 1585	MSR_PKG_C9_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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.	
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 35H, 53	MSR_CORE_THREAD_COUNT	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Configured State of Enabled Processor Core Count and Logical Processor Count (R/O) ▪ 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.		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.	
Register Address: 53H, 83	MSR_THREAD_ID_INFO	
A Hardware Assigned ID for the Logical Processor (R/O)		Thread
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.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
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.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
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.	
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
MC Bank Error Configuration (R/W)		Package
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.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
63:56	Maximum Ratio Limit for16C Maximum turbo ratio limit of 16 core active.	Package
Register Address: 1AFH, 431	MSR_TURBO_RATIO_LIMIT2	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 17C Maximum turbo ratio limit of 17 core active.	Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
15:8	Maximum Ratio Limit for 18C Maximum turbo ratio limit of 18 core active.	Package
62:16	Reserved.	Package
63	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).	Package
Register Address: 414H, 1044	IA32_MC5_CTL	
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.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
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.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
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.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
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.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
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.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
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.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
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.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
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.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
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.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
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.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
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.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
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.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
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.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
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.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
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.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
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.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
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.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
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.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
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.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
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.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
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.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
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.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
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.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
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.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
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.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
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.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
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.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
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.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
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.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
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.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
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.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
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.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
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.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
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.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
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.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
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.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
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.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
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.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
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.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
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.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
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.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
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.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
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.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
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.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
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.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
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.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 445H, 1093	IA32_MC17_STATUS	
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.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
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.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
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.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
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.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
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.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
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.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
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.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
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.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
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.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
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.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
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.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
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.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
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.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
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.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	
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.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
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.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
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.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
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.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 15.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	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).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 15.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy Consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61EH, 1566	MSR_PCIE_PLL_RATIO	
Configuration of PCIE PLL Relative to BCLK(R/W)		Package
1:0	PCIE Ratio (R/W) 00b: Use 5:5 mapping for100MHz operation (default). 01b: Use 5:4 mapping for125MHz operation. 10b: Use 5:3 mapping for166MHz operation. 11b: Use 5:2 mapping for250MHz operation.	Package
2	LPLL Select (R/W) If 1, use configured setting of PCIE Ratio.	Package
3	LONG RESET (R/W) If 1, wait an additional time-out before re-locking Gen2/Gen3 PLLs.	Package
63:4	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
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.		Package
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.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
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.	
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.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.		Thread
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.	
Register Address: C8EH, 3214	IA32_QM_CTR	
Monitoring Counter Register (R/O) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.		Thread
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.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 700H, 1792	MSR_PMON_GLOBAL_CTL	
Uncore Perfmon Per-Socket Global Control		Package
Register Address: 701H, 1793	MSR_PMON_GLOBAL_STATUS	
Uncore Perfmon Per-Socket Global Status		Package
Register Address: 702H, 1794	MSR_PMON_GLOBAL_CONFIG	
Uncore Perfmon Per-Socket Global Configuration		Package
Register Address: 703H, 1795	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-Box UCLK Fixed Counter Control		Package
Register Address: 704H, 1796	MSR_U_PMON_UCLK_FIXED_CTR	
Uncore U-Box UCLK Fixed Counter		Package
Register Address: 705H, 1797	MSR_U_PMON_EVNTSEL0	
Uncore U-Box Perfmon Event Select for U-Box Counter 0		Package
Register Address: 706H, 1798	MSR_U_PMON_EVNTSEL1	
Uncore U-Box Perfmon Event Select for U-Box Counter 1		Package
Register Address: 708H, 1800	MSR_U_PMON_BOX_STATUS	
Uncore U-Box Perfmon U-Box Wide Status		Package
Register Address: 709H, 1801	MSR_U_PMON_CTR0	
Uncore U-Box Perfmon Counter 0		Package
Register Address: 70AH, 1802	MSR_U_PMON_CTR1	
Uncore U-Box Perfmon Counter 1		Package
Register Address: 710H, 1808	MSR_PCU_PMON_BOX_CTL	
Uncore PCU Perfmon for PCU-Box-Wide Control		Package
Register Address: 711H, 1809	MSR_PCU_PMON_EVNTSEL0	
Uncore PCU Perfmon Event Select for PCU Counter 0		Package
Register Address: 712H, 1810	MSR_PCU_PMON_EVNTSEL1	
Uncore PCU Perfmon Event Select for PCU Counter 1		Package
Register Address: 713H, 1811	MSR_PCU_PMON_EVNTSEL2	
Uncore PCU Perfmon Event Select for PCU Counter 2		Package
Register Address: 714H, 1812	MSR_PCU_PMON_EVNTSEL3	
Uncore PCU Perfmon Event Select for PCU Counter 3		Package
Register Address: 715H, 1813	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU Perfmon Box-Wide Filter		Package
Register Address: 716H, 1814	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU Perfmon Box Wide Status		Package
Register Address: 717H, 1815	MSR_PCU_PMON_CTR0	
Uncore PCU Perfmon Counter 0		Package
Register Address: 718H, 1816	MSR_PCU_PMON_CTR1	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore PCU Perfmon Counter 1		Package
Register Address: 719H, 1817	MSR_PCU_PMON_CTR2	
Uncore PCU Perfmon Counter 2		Package
Register Address: 71AH, 1818	MSR_PCU_PMON_CTR3	
Uncore PCU Perfmon Counter 3		Package
Register Address: 720H, 1824	MSR_S0_PMON_BOX_CTL	
Uncore SBo 0 Perfmon for SBo 0 Box-Wide Control		Package
Register Address: 721H, 1825	MSR_S0_PMON_EVNTSEL0	
Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 0		Package
Register Address: 722H, 1826	MSR_S0_PMON_EVNTSEL1	
Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 1		Package
Register Address: 723H, 1827	MSR_S0_PMON_EVNTSEL2	
Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 2		Package
Register Address: 724H, 1828	MSR_S0_PMON_EVNTSEL3	
Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 3		Package
Register Address: 725H, 1829	MSR_S0_PMON_BOX_FILTER	
Uncore SBo 0 Perfmon Box-Wide Filter		Package
Register Address: 726H, 1830	MSR_S0_PMON_CTR0	
Uncore SBo 0 Perfmon Counter 0		Package
Register Address: 727H, 1831	MSR_S0_PMON_CTR1	
Uncore SBo 0 Perfmon Counter 1		Package
Register Address: 728H, 1832	MSR_S0_PMON_CTR2	
Uncore SBo 0 Perfmon Counter 2		Package
Register Address: 729H, 1833	MSR_S0_PMON_CTR3	
Uncore SBo 0 Perfmon Counter 3		Package
Register Address: 72AH, 1834	MSR_S1_PMON_BOX_CTL	
Uncore SBo 1 Perfmon for SBo 1 Box-Wide Control		Package
Register Address: 72BH, 1835	MSR_S1_PMON_EVNTSEL0	
Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 0		Package
Register Address: 72CH, 1836	MSR_S1_PMON_EVNTSEL1	
Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 1		Package
Register Address: 72DH, 1837	MSR_S1_PMON_EVNTSEL2	
Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 2		Package
Register Address: 72EH, 1838	MSR_S1_PMON_EVNTSEL3	
Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 3		Package
Register Address: 72FH, 1839	MSR_S1_PMON_BOX_FILTER	
Uncore SBo 1 Perfmon Box-Wide Filter		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 730H, 1840	MSR_S1_PMON_CTR0	
Uncore SBo 1 Perfmon Counter 0		Package
Register Address: 731H, 1841	MSR_S1_PMON_CTR1	
Uncore SBo 1 Perfmon Counter 1		Package
Register Address: 732H, 1842	MSR_S1_PMON_CTR2	
Uncore SBo 1 Perfmon Counter 2		Package
Register Address: 733H, 1843	MSR_S1_PMON_CTR3	
Uncore SBo 1 Perfmon Counter 3		Package
Register Address: 734H, 1844	MSR_S2_PMON_BOX_CTL	
Uncore SBo 2 Perfmon for SBo 2 Box-Wide Control		Package
Register Address: 735H, 1845	MSR_S2_PMON_EVNTSEL0	
Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 0		Package
Register Address: 736H, 1846	MSR_S2_PMON_EVNTSEL1	
Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 1		Package
Register Address: 737H, 1847	MSR_S2_PMON_EVNTSEL2	
Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 2		Package
Register Address: 738H, 1848	MSR_S2_PMON_EVNTSEL3	
Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 3		Package
Register Address: 739H, 1849	MSR_S2_PMON_BOX_FILTER	
Uncore SBo 2 Perfmon Box-Wide Filter		Package
Register Address: 73AH, 1850	MSR_S2_PMON_CTR0	
Uncore SBo 2 Perfmon Counter 0		Package
Register Address: 73BH, 1851	MSR_S2_PMON_CTR1	
Uncore SBo 2 Perfmon Counter 1		Package
Register Address: 73CH, 1852	MSR_S2_PMON_CTR2	
Uncore SBo 2 Perfmon Counter 2		Package
Register Address: 73DH, 1853	MSR_S2_PMON_CTR3	
Uncore SBo 2 Perfmon Counter 3		Package
Register Address: 73EH, 1854	MSR_S3_PMON_BOX_CTL	
Uncore SBo 3 Perfmon for SBo 3 Box-Wide Control		Package
Register Address: 73FH, 1855	MSR_S3_PMON_EVNTSEL0	
Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 0		Package
Register Address: 740H, 1856	MSR_S3_PMON_EVNTSEL1	
Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 1		Package
Register Address: 741H, 1857	MSR_S3_PMON_EVNTSEL2	
Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 2		Package
Register Address: 742H, 1858	MSR_S3_PMON_EVNTSEL3	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 3		Package
Register Address: 743H, 1859	MSR_S3_PMON_BOX_FILTER	
Uncore SBo 3 Perfmon Box-Wide Filter		Package
Register Address: 744H, 1860	MSR_S3_PMON_CTR0	
Uncore SBo 3 Perfmon Counter 0		Package
Register Address: 745H, 1861	MSR_S3_PMON_CTR1	
Uncore SBo 3 Perfmon Counter 1		Package
Register Address: 746H, 1862	MSR_S3_PMON_CTR2	
Uncore SBo 3 Perfmon Counter 2		Package
Register Address: 747H, 1863	MSR_S3_PMON_CTR3	
Uncore SBo 3 Perfmon Counter 3		Package
Register Address: E00H, 3584	MSR_C0_PMON_BOX_CTL	
Uncore C-Box 0 Perfmon for Box-Wide Control		Package
Register Address: E01H, 3585	MSR_C0_PMON_EVNTSEL0	
Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 0		Package
Register Address: E02H, 3586	MSR_C0_PMON_EVNTSEL1	
Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 1		Package
Register Address: E03H, 3587	MSR_C0_PMON_EVNTSEL2	
Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 2		Package
Register Address: E04H, 3588	MSR_C0_PMON_EVNTSEL3	
Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 3		Package
Register Address: E05H, 3589	MSR_C0_PMON_BOX_FILTER0	
Uncore C-Box 0 Perfmon Box Wide Filter 0		Package
Register Address: E06H, 3590	MSR_C0_PMON_BOX_FILTER1	
Uncore C-Box 0 Perfmon Box Wide Filter 1		Package
Register Address: E07H, 3591	MSR_C0_PMON_BOX_STATUS	
Uncore C-Box 0 Perfmon Box Wide Status		Package
Register Address: E08H, 3592	MSR_C0_PMON_CTR0	
Uncore C-Box 0 Perfmon Counter 0		Package
Register Address: E09H, 3593	MSR_C0_PMON_CTR1	
Uncore C-Box 0 Perfmon Counter 1		Package
Register Address: E0AH, 3594	MSR_C0_PMON_CTR2	
Uncore C-Box 0 Perfmon Counter 2		Package
Register Address: E0BH, 3595	MSR_C0_PMON_CTR3	
Uncore C-Box 0 Perfmon Counter 3		Package
Register Address: E10H, 3600	MSR_C1_PMON_BOX_CTL	
Uncore C-Box 1 Perfmon for Box-Wide Control		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E11H, 3601	MSR_C1_PMON_EVNTSEL0	
Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 0		Package
Register Address: E12H, 3602	MSR_C1_PMON_EVNTSEL1	
Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 1		Package
Register Address: E13H, 3603	MSR_C1_PMON_EVNTSEL2	
Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 2		Package
Register Address: E14H, 3604	MSR_C1_PMON_EVNTSEL3	
Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 3		Package
Register Address: E15H, 3605	MSR_C1_PMON_BOX_FILTER0	
Uncore C-Box 1 Perfmon Box Wide Filter 0		Package
Register Address: E16H, 3606	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 Perfmon Box Wide Filter1		Package
Register Address: E17H, 3607	MSR_C1_PMON_BOX_STATUS	
Uncore C-Box 1 Perfmon Box Wide Status		Package
Register Address: E18H, 3608	MSR_C1_PMON_CTR0	
Uncore C-Box 1 Perfmon Counter 0		Package
Register Address: E19H, 3609	MSR_C1_PMON_CTR1	
Uncore C-Box 1 Perfmon Counter 1		Package
Register Address: E1AH, 3610	MSR_C1_PMON_CTR2	
Uncore C-Box 1 Perfmon Counter 2		Package
Register Address: E1BH, 3611	MSR_C1_PMON_CTR3	
Uncore C-Box 1 Perfmon Counter 3		Package
Register Address: E20H, 3616	MSR_C2_PMON_BOX_CTL	
Uncore C-Box 2 Perfmon for Box-Wide Control		Package
Register Address: E21H, 3617	MSR_C2_PMON_EVNTSEL0	
Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 0		Package
Register Address: E22H, 3618	MSR_C2_PMON_EVNTSEL1	
Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 1		Package
Register Address: E23H, 3619	MSR_C2_PMON_EVNTSEL2	
Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 2		Package
Register Address: E24H, 3620	MSR_C2_PMON_EVNTSEL3	
Uncore C-Box 2 Perfmon Event select for C-Box 2 Counter 3		Package
Register Address: E25H, 3621	MSR_C2_PMON_BOX_FILTER0	
Uncore C-Box 2 Perfmon Box Wide Filter 0		Package
Register Address: E26H, 3622	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 Perfmon Box Wide Filter1		Package
Register Address: E27H, 3623	MSR_C2_PMON_BOX_STATUS	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 2 Perfmon Box Wide Status		Package
Register Address: E28H, 3624	MSR_C2_PMON_CTR0	
Uncore C-Box 2 Perfmon Counter 0		Package
Register Address: E29H, 3625	MSR_C2_PMON_CTR1	
Uncore C-Box 2 Perfmon Counter 1		Package
Register Address: E2AH, 3626	MSR_C2_PMON_CTR2	
Uncore C-Box 2 Perfmon Counter 2		Package
Register Address: E2BH, 3627	MSR_C2_PMON_CTR3	
Uncore C-Box 2 Perfmon Counter 3		Package
Register Address: E30H, 3632	MSR_C3_PMON_BOX_CTL	
Uncore C-Box 3 Perfmon for Box-Wide Control		Package
Register Address: E31H, 3633	MSR_C3_PMON_EVNTSEL0	
Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 0		Package
Register Address: E32H, 3634	MSR_C3_PMON_EVNTSEL1	
Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 1		Package
Register Address: E33H, 3635	MSR_C3_PMON_EVNTSEL2	
Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 2		Package
Register Address: E34H, 3636	MSR_C3_PMON_EVNTSEL3	
Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 3		Package
Register Address: E35H, 3637	MSR_C3_PMON_BOX_FILTER0	
Uncore C-Box 3 Perfmon Box Wide Filter 0		Package
Register Address: E36H, 3638	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 Perfmon Box Wide Filter1		Package
Register Address: E37H, 3639	MSR_C3_PMON_BOX_STATUS	
Uncore C-Box 3 Perfmon Box Wide Status		Package
Register Address: E38H, 3640	MSR_C3_PMON_CTR0	
Uncore C-Box 3 Perfmon Counter 0		Package
Register Address: E39H, 3641	MSR_C3_PMON_CTR1	
Uncore C-Box 3 Perfmon Counter 1		Package
Register Address: E3AH, 3642	MSR_C3_PMON_CTR2	
Uncore C-Box 3 Perfmon Counter 2		Package
Register Address: E3BH, 3643	MSR_C3_PMON_CTR3	
Uncore C-Box 3 Perfmon Counter 3		Package
Register Address: E40H, 3648	MSR_C4_PMON_BOX_CTL	
Uncore C-Box 4 Perfmon for Box-Wide Control		Package
Register Address: E41H, 3649	MSR_C4_PMON_EVNTSEL0	
Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E42H, 3650	MSR_C4_PMON_EVNTSEL1	
Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 1		Package
Register Address: E43H, 3651	MSR_C4_PMON_EVNTSEL2	
Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 2		Package
Register Address: E44H, 3652	MSR_C4_PMON_EVNTSEL3	
Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 3		Package
Register Address: E45H, 3653	MSR_C4_PMON_BOX_FILTER0	
Uncore C-Box 4 Perfmon Box Wide Filter 0		Package
Register Address: E46H, 3654	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 Perfmon Box Wide Filter1		Package
Register Address: E47H, 3655	MSR_C4_PMON_BOX_STATUS	
Uncore C-Box 4 Perfmon Box Wide Status		Package
Register Address: E48H, 3656	MSR_C4_PMON_CTR0	
Uncore C-Box 4 Perfmon Counter 0		Package
Register Address: E49H, 3657	MSR_C4_PMON_CTR1	
Uncore C-Box 4 Perfmon Counter 1		Package
Register Address: E4AH, 3658	MSR_C4_PMON_CTR2	
Uncore C-Box 4 Perfmon Counter 2		Package
Register Address: E4BH, 3659	MSR_C4_PMON_CTR3	
Uncore C-Box 4 Perfmon Counter 3		Package
Register Address: E50H, 3664	MSR_C5_PMON_BOX_CTL	
Uncore C-Box 5 Perfmon for Box-Wide Control		Package
Register Address: E51H, 3665	MSR_C5_PMON_EVNTSEL0	
Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 0		Package
Register Address: E52H, 3666	MSR_C5_PMON_EVNTSEL1	
Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 1		Package
Register Address: E53H, 3667	MSR_C5_PMON_EVNTSEL2	
Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 2		Package
Register Address: E54H, 3668	MSR_C5_PMON_EVNTSEL3	
Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 3		Package
Register Address: E55H, 3669	MSR_C5_PMON_BOX_FILTER0	
Uncore C-Box 5 Perfmon Box Wide Filter 0		Package
Register Address: E56H, 3670	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 Perfmon Box Wide Filter 1		Package
Register Address: E57H, 3671	MSR_C5_PMON_BOX_STATUS	
Uncore C-Box 5 Perfmon Box Wide Status		Package
Register Address: E58H, 3672	MSR_C5_PMON_CTR0	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 5 Perfmon Counter 0		Package
Register Address: E59H, 3673	MSR_C5_PMON_CTR1	
Uncore C-Box 5 Perfmon Counter 1		Package
Register Address: E5AH, 3674	MSR_C5_PMON_CTR2	
Uncore C-Box 5 Perfmon Counter 2		Package
Register Address: E5BH, 3675	MSR_C5_PMON_CTR3	
Uncore C-Box 5 Perfmon Counter 3		Package
Register Address: E60H, 3680	MSR_C6_PMON_BOX_CTL	
Uncore C-Box 6 Perfmon for Box-Wide Control		Package
Register Address: E61H, 3681	MSR_C6_PMON_EVNTSEL0	
Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 0		Package
Register Address: E62H, 3682	MSR_C6_PMON_EVNTSEL1	
Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 1		Package
Register Address: E63H, 3683	MSR_C6_PMON_EVNTSEL2	
Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 2		Package
Register Address: E64H, 3684	MSR_C6_PMON_EVNTSEL3	
Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 3		Package
Register Address: E65H, 3685	MSR_C6_PMON_BOX_FILTER0	
Uncore C-Box 6 Perfmon Box Wide Filter 0		Package
Register Address: E66H, 3686	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 Perfmon Box Wide Filter 1		Package
Register Address: E67H, 3687	MSR_C6_PMON_BOX_STATUS	
Uncore C-Box 6 Perfmon Box Wide Status		Package
Register Address: E68H, 3688	MSR_C6_PMON_CTR0	
Uncore C-Box 6 Perfmon Counter 0		Package
Register Address: E69H, 3689	MSR_C6_PMON_CTR1	
Uncore C-Box 6 Perfmon Counter 1		Package
Register Address: E6AH, 3690	MSR_C6_PMON_CTR2	
Uncore C-Box 6 Perfmon Counter 2		Package
Register Address: E6BH, 3691	MSR_C6_PMON_CTR3	
Uncore C-Box 6 Perfmon Counter 3		Package
Register Address: E70H, 3696	MSR_C7_PMON_BOX_CTL	
Uncore C-Box 7 Perfmon for Box-Wide Control		Package
Register Address: E71H, 3697	MSR_C7_PMON_EVNTSEL0	
Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 0		Package
Register Address: E72H, 3698	MSR_C7_PMON_EVNTSEL1	
Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 1		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E73H, 3699	MSR_C7_PMON_EVNTSEL2	
Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 2		Package
Register Address: E74H, 3700	MSR_C7_PMON_EVNTSEL3	
Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 3		Package
Register Address: E75H, 3701	MSR_C7_PMON_BOX_FILTER0	
Uncore C-Box 7 Perfmon Box Wide Filter 0		Package
Register Address: E76H, 3702	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 Perfmon Box Wide Filter 1		Package
Register Address: E77H, 3703	MSR_C7_PMON_BOX_STATUS	
Uncore C-Box 7 Perfmon Box Wide Status		Package
Register Address: E78H, 3704	MSR_C7_PMON_CTR0	
Uncore C-Box 7 Perfmon Counter 0		Package
Register Address: E79H, 3705	MSR_C7_PMON_CTR1	
Uncore C-Box 7 Perfmon Counter 1		Package
Register Address: E7AH, 3706	MSR_C7_PMON_CTR2	
Uncore C-Box 7 Perfmon Counter 2		Package
Register Address: E7BH, 3707	MSR_C7_PMON_CTR3	
Uncore C-Box 7 Perfmon Counter 3		Package
Register Address: E80H, 3712	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 Perfmon Local Box Wide Control		Package
Register Address: E81H, 3713	MSR_C8_PMON_EVNTSEL0	
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0		Package
Register Address: E82H, 3714	MSR_C8_PMON_EVNTSEL1	
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1		Package
Register Address: E83H, 3715	MSR_C8_PMON_EVNTSEL2	
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2		Package
Register Address: E84H, 3716	MSR_C8_PMON_EVNTSEL3	
Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3		Package
Register Address: E85H, 3717	MSR_C8_PMON_BOX_FILTER0	
Uncore C-Box 8 Perfmon Box Wide Filter 0		Package
Register Address: E86H, 3718	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 Perfmon Box Wide Filter 1		Package
Register Address: E87H, 3719	MSR_C8_PMON_BOX_STATUS	
Uncore C-Box 8 Perfmon Box Wide Status		Package
Register Address: E88H, 3720	MSR_C8_PMON_CTR0	
Uncore C-Box 8 Perfmon Counter 0		Package
Register Address: E89H, 3721	MSR_C8_PMON_CTR1	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 8 Perfmon Counter 1		Package
Register Address: E8AH, 3722	MSR_C8_PMON_CTR2	
Uncore C-Box 8 Perfmon Counter 2		Package
Register Address: E8BH, 3723	MSR_C8_PMON_CTR3	
Uncore C-Box 8 Perfmon Counter 3		Package
Register Address: E90H, 3728	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 Perfmon Local Box Wide Control		Package
Register Address: E91H, 3729	MSR_C9_PMON_EVNTSEL0	
Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 0		Package
Register Address: E92H, 3730	MSR_C9_PMON_EVNTSEL1	
Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 1		Package
Register Address: E93H, 3731	MSR_C9_PMON_EVNTSEL2	
Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 2		Package
Register Address: E94H, 3732	MSR_C9_PMON_EVNTSEL3	
Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 3		Package
Register Address: E95H, 3733	MSR_C9_PMON_BOX_FILTER0	
Uncore C-Box 9 Perfmon Box Wide Filter 0		Package
Register Address: E96H, 3734	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 Perfmon Box Wide Filter 1		Package
Register Address: E97H, 3735	MSR_C9_PMON_BOX_STATUS	
Uncore C-Box 9 Perfmon Box Wide Status		Package
Register Address: E98H, 3736	MSR_C9_PMON_CTR0	
Uncore C-Box 9 Perfmon Counter 0		Package
Register Address: E99H, 3737	MSR_C9_PMON_CTR1	
Uncore C-Box 9 Perfmon Counter 1		Package
Register Address: E9AH, 3738	MSR_C9_PMON_CTR2	
Uncore C-Box 9 Perfmon Counter 2		Package
Register Address: E9BH, 3739	MSR_C9_PMON_CTR3	
Uncore C-Box 9 Perfmon Counter 3		Package
Register Address: EA0H, 3744	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 Perfmon Local Box Wide Control		Package
Register Address: EA1H, 3745	MSR_C10_PMON_EVNTSEL0	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0		Package
Register Address: EA2H, 3746	MSR_C10_PMON_EVNTSEL1	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1		Package
Register Address: EA3H, 3747	MSR_C10_PMON_EVNTSEL2	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: EA4H, 3748	MSR_C10_PMON_EVNTSEL3	
Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3		Package
Register Address: EA5H, 3749	MSR_C10_PMON_BOX_FILTER0	
Uncore C-Box 10 Perfmon Box Wide Filter 0		Package
Register Address: EA6H, 3750	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 Perfmon Box Wide Filter 1		Package
Register Address: EA7H, 3751	MSR_C10_PMON_BOX_STATUS	
Uncore C-Box 10 Perfmon Box Wide Status		Package
Register Address: EA8H, 3752	MSR_C10_PMON_CTR0	
Uncore C-Box 10 Perfmon Counter 0		Package
Register Address: EA9H, 3753	MSR_C10_PMON_CTR1	
Uncore C-Box 10 perfmon Counter 1		Package
Register Address: EAAH, 3754	MSR_C10_PMON_CTR2	
Uncore C-Box 10 Perfmon Counter 2		Package
Register Address: EABH, 3755	MSR_C10_PMON_CTR3	
Uncore C-Box 10 Perfmon Counter 3		Package
Register Address: EB0H, 3760	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 Perfmon Local Box Wide Control		Package
Register Address: EB1H, 3761	MSR_C11_PMON_EVNTSEL0	
Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0		Package
Register Address: EB2H, 3762	MSR_C11_PMON_EVNTSEL1	
Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1		Package
Register Address: EB3H, 3763	MSR_C11_PMON_EVNTSEL2	
Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2		Package
Register Address: EB4H, 3764	MSR_C11_PMON_EVNTSEL3	
Uncore C-box 11 Perfmon Event Select for C-Box 11 Counter 3		Package
Register Address: EB5H, 3765	MSR_C11_PMON_BOX_FILTER0	
Uncore C-Box 11 Perfmon Box Wide Filter 0		Package
Register Address: EB6H, 3766	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 Perfmon Box Wide Filter 1		Package
Register Address: EB7H, 3767	MSR_C11_PMON_BOX_STATUS	
Uncore C-Box 11 Perfmon Box Wide Status		Package
Register Address: EB8H, 3768	MSR_C11_PMON_CTR0	
Uncore C-Box 11 Perfmon Counter 0		Package
Register Address: EB9H, 3769	MSR_C11_PMON_CTR1	
Uncore C-Box 11 Perfmon Counter 1		Package
Register Address: EBAH, 3770	MSR_C11_PMON_CTR2	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 11 Perfmon Counter 2		Package
Register Address: EBBH, 3771	MSR_C11_PMON_CTR3	
Uncore C-Box 11 Perfmon Counter 3		Package
Register Address: EC0H, 3776	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 Perfmon Local Box Wide Control		Package
Register Address: EC1H, 3777	MSR_C12_PMON_EVNTSEL0	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0		Package
Register Address: EC2H, 3778	MSR_C12_PMON_EVNTSEL1	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1		Package
Register Address: EC3H, 3779	MSR_C12_PMON_EVNTSEL2	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2		Package
Register Address: EC4H, 3780	MSR_C12_PMON_EVNTSEL3	
Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3		Package
Register Address: EC5H, 3781	MSR_C12_PMON_BOX_FILTER0	
Uncore C-Box 12 Perfmon Box Wide Filter 0		Package
Register Address: EC6H, 3782	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 Perfmon Box Wide Filter 1		Package
Register Address: EC7H, 3783	MSR_C12_PMON_BOX_STATUS	
Uncore C-Box 12 Perfmon Box Wide Status		Package
Register Address: EC8H, 3784	MSR_C12_PMON_CTR0	
Uncore C-Box 12 Perfmon Counter 0		Package
Register Address: EC9H, 3785	MSR_C12_PMON_CTR1	
Uncore C-Box 12 Perfmon Counter 1		Package
Register Address: ECAH, 3786	MSR_C12_PMON_CTR2	
Uncore C-Box 12 Perfmon Counter 2		Package
Register Address: ECBH, 3787	MSR_C12_PMON_CTR3	
Uncore C-Box 12 Perfmon Counter 3		Package
Register Address: ED0H, 3792	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 Perfmon local box wide control.		Package
Register Address: ED1H, 3793	MSR_C13_PMON_EVNTSEL0	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0		Package
Register Address: ED2H, 3794	MSR_C13_PMON_EVNTSEL1	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1		Package
Register Address: ED3H, 3795	MSR_C13_PMON_EVNTSEL2	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2		Package
Register Address: ED4H, 3796	MSR_C13_PMON_EVNTSEL3	
Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: ED5H, 3797	MSR_C13_PMON_BOX_FILTER0	
Uncore C-Box 13 Perfmon Box Wide Filter 0		Package
Register Address: ED6H, 3798	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 Perfmon Box Wide Filter 1		Package
Register Address: ED7H, 3799	MSR_C13_PMON_BOX_STATUS	
Uncore C-Box 13 Perfmon Box Wide Status		Package
Register Address: ED8H, 3800	MSR_C13_PMON_CTR0	
Uncore C-Box 13 Perfmon Counter 0		Package
Register Address: ED9H, 3801	MSR_C13_PMON_CTR1	
Uncore C-Box 13 Perfmon Counter 1		Package
Register Address: EDAH, 3802	MSR_C13_PMON_CTR2	
Uncore C-Box 13 Perfmon Counter 2		Package
Register Address: EDBH, 3803	MSR_C13_PMON_CTR3	
Uncore C-Box 13 Perfmon Counter 3		Package
Register Address: EE0H, 3808	MSR_C14_PMON_BOX_CTL	
Uncore C-Box 14 Perfmon Local Box Wide Control		Package
Register Address: EE1H, 3809	MSR_C14_PMON_EVNTSEL0	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0		Package
Register Address: EE2H, 3810	MSR_C14_PMON_EVNTSEL1	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1		Package
Register Address: EE3H, 3811	MSR_C14_PMON_EVNTSEL2	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2		Package
Register Address: EE4H, 3812	MSR_C14_PMON_EVNTSEL3	
Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3		Package
Register Address: EE5H, 3813	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 Perfmon Box Wide Filter 0		Package
Register Address: EE6H, 3814	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 Perfmon Box Wide Filter 1		Package
Register Address: EE7H, 3815	MSR_C14_PMON_BOX_STATUS	
Uncore C-Box 14 Perfmon Box Wide Status		Package
Register Address: EE8H, 3816	MSR_C14_PMON_CTR0	
Uncore C-Box 14 Perfmon Counter 0		Package
Register Address: EE9H, 3817	MSR_C14_PMON_CTR1	
Uncore C-Box 14 Perfmon Counter 1		Package
Register Address: EEAH, 3818	MSR_C14_PMON_CTR2	
Uncore C-Box 14 Perfmon Counter 2		Package
Register Address: EEBH, 3819	MSR_C14_PMON_CTR3	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 14 Perfmon Counter 3		Package
Register Address: EF0H, 3824	MSR_C15_PMON_BOX_CTL	
Uncore C-Box 15 Perfmon Local Box Wide Control		Package
Register Address: EF1H, 3825	MSR_C15_PMON_EVNTSEL0	
Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 0		Package
Register Address: EF2H, 3826	MSR_C15_PMON_EVNTSEL1	
Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 1		Package
Register Address: EF3H, 3827	MSR_C15_PMON_EVNTSEL2	
Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 2		Package
Register Address: EF4H, 3828	MSR_C15_PMON_EVNTSEL3	
Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 3		Package
Register Address: EF5H, 3829	MSR_C15_PMON_BOX_FILTER0	
Uncore C-Box 15 Perfmon Box Wide Filter 0		Package
Register Address: EF6H, 3830	MSR_C15_PMON_BOX_FILTER1	
Uncore C-Box 15 Perfmon Box Wide Filter 1		Package
Register Address: EF7H, 3831	MSR_C15_PMON_BOX_STATUS	
Uncore C-Box 15 Perfmon Box Wide Status		Package
Register Address: EF8H, 3832	MSR_C15_PMON_CTR0	
Uncore C-Box 15 Perfmon Counter 0		Package
Register Address: EF9H, 3833	MSR_C15_PMON_CTR1	
Uncore C-Box 15 Perfmon Counter 1		Package
Register Address: EFAH, 3834	MSR_C15_PMON_CTR2	
Uncore C-Box 15 Perfmon Counter 2		Package
Register Address: EFBH, 3835	MSR_C15_PMON_CTR3	
Uncore C-Box 15 Perfmon Counter 3		Package
Register Address: F00H, 3840	MSR_C16_PMON_BOX_CTL	
Uncore C-Box 16 Perfmon for Box-Wide Control		Package
Register Address: F01H, 3841	MSR_C16_PMON_EVNTSEL0	
Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 0		Package
Register Address: F02H, 3842	MSR_C16_PMON_EVNTSEL1	
Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 1		Package
Register Address: F03H, 3843	MSR_C16_PMON_EVNTSEL2	
Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 2		Package
Register Address: F04H, 3844	MSR_C16_PMON_EVNTSEL3	
Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 3		Package
Register Address: F05H, 3845	MSR_C16_PMON_BOX_FILTER0	
Uncore C-Box 16 Perfmon Box Wide Filter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: F06H, 3846	MSR_C16_PMON_BOX_FILTER1	
Uncore C-Box 16 Perfmon Box Wide Filter 1		Package
Register Address: F07H, 3847	MSR_C16_PMON_BOX_STATUS	
Uncore C-Box 16 Perfmon Box Wide Status		Package
Register Address: F08H, 3848	MSR_C16_PMON_CTR0	
Uncore C-Box 16 Perfmon Counter 0		Package
Register Address: F09H, 3849	MSR_C16_PMON_CTR1	
Uncore C-Box 16 Perfmon Counter 1		Package
Register Address: F0AH, 3850	MSR_C16_PMON_CTR2	
Uncore C-Box 16 Perfmon Counter 2		Package
Register Address: F0BH, 3851	MSR_C16_PMON_CTR3	
Uncore C-Box 16 Perfmon Counter 3		Package
Register Address: F10H, 3856	MSR_C17_PMON_BOX_CTL	
Uncore C-Box 17 Perfmon for Box-Wide Control		Package
Register Address: F11H, 3857	MSR_C17_PMON_EVNTSEL0	
Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 0		Package
Register Address: F12H, 3858	MSR_C17_PMON_EVNTSEL1	
Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 1		Package
Register Address: F13H, 3859	MSR_C17_PMON_EVNTSEL2	
Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 2		Package
Register Address: F14H, 3860	MSR_C17_PMON_EVNTSEL3	
Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 3		Package
Register Address: F15H, 3861	MSR_C17_PMON_BOX_FILTER0	
Uncore C-Box 17 Perfmon Box Wide Filter 0		Package
Register Address: F16H, 3862	MSR_C17_PMON_BOX_FILTER1	
Uncore C-Box 17 Perfmon Box Wide Filter1		Package
Register Address: F17H, 3863	MSR_C17_PMON_BOX_STATUS	
Uncore C-Box 17 Perfmon Box Wide Status		Package
Register Address: F18H, 3864	MSR_C17_PMON_CTR0	
Uncore C-Box 17 Perfmon Counter 0		Package
Register Address: F19H, 3865	MSR_C17_PMON_CTR1	
Uncore C-Box 17 Perfmon Counter 1		Package
Register Address: F1AH, 3866	MSR_C17_PMON_CTR2	
Uncore C-Box 17 Perfmon Counter 2		Package
Register Address: F1BH, 3867	MSR_C17_PMON_CTR3	
Uncore C-Box 17 Perfmon Counter 3		Package

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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 20.6.2.2, "Global Counter Control Facilities."		Thread
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.	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 20.6.2.2, "Global Counter Control Facilities."		Thread
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	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W)		Thread
6:0	Reserved.	
MAXPHYADDR[1]-1:7	Base physical address.	
63:MAXPHYADDR	Reserved.	
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W)		Thread
6:0	Reserved.	
31:7	MaskOrTableOffset	
63:32	Output Offset.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
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.	
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.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
0	Reserved, writes ignored.	
1	ContexEn, writes ignored.	
2	TriggerEn, writes ignored.	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
3	Reserved	
4	Error (R/W)	
5	Stopped	
63:6	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved.	
63:5	CR3[63:5] value to match.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
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.		Package
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. 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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5core active.	Package

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6core active.	Package
63:48	Reserved.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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 MWAT(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)	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
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.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
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.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) 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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
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.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) See Table 2-26.	
27:24	TCC Activation Offset (R/W) See Table 2-26.	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:28	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C	Package
15:8	Maximum Ratio Limit for 2C	Package
23:16	Maximum Ratio Limit for 3C	Package
31:24	Maximum Ratio Limit for 4C	Package
39:32	Maximum Ratio Limit for 5C	Package
47:40	Maximum Ratio Limit for 6C	Package
55:48	Maximum Ratio Limit for 7C	Package
63:56	Maximum Ratio Limit for 8C	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C	Package
15:8	Maximum Ratio Limit for 10C	Package
23:16	Maximum Ratio Limit for 11C	Package
31:24	Maximum Ratio Limit for 12C	Package
39:32	Maximum Ratio Limit for 13C	Package
47:40	Maximum Ratio Limit for 14C	Package
55:48	Maximum Ratio Limit for 15C	Package
63:56	Maximum Ratio Limit for 16C	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 15.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	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).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 15.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
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.		Package
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.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
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.	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 15.4.2, "Enabling HWP."		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities."		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 15.4.4, "Managing HWP."		Thread
7:0	Minimum Performance (R/W)	
15:8	Maximum Performance (R/W)	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
23:16	Desired Performance (R/W)	
63:24	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 15.4.5, "HWP Feedback."		Thread
1:0	Reserved.	
2	Excursion to Minimum (R/O)	
63:3	Reserved.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.		Thread
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.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	Reserved.	
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	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).	Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
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.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
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.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
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.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
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.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
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.		Package

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 41DH, 1053	IA32_MC7_STATUS	
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.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
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.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
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.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
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.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
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.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
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.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
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.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
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.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
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.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
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.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
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.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
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.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
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.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
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.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
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.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
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.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
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.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
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.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
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.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
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.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
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.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	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).	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
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.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
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.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
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.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
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.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
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.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 419H, 1049	IA32_MC6_STATUS	
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.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
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.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
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.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
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.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
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.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
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.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
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.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
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.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
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.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
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.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
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.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
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.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
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.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
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.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
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.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
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.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
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.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
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.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
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.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
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.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
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.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
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.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
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.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
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.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 432H, 1074	IA32_MC12_ADDR	
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.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
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.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
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.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
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.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
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.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
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.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
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.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
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.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
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.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
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.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
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.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
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.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
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.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
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.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
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.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
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.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
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.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
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.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
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.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
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.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
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.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
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.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
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.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
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.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
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.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
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.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
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.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
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.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
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.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
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.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
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.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
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.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
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.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
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.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
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.		Package
Register Address: C81H, 3201	IA32_L3_QOS_CFG	
Cache Allocation Technology Configuration (R/W)		Package
0	CAT Enable. Set 1 to enable Cache Allocation Technology.	
63:1	Reserved.	
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—13TH GENERATION INTEL® CORE™ PROCESSORS, 1ST—5TH GENERATION INTEL® XEON® SCALABLE PROCESSOR FAMILIES, INTEL® CORE™ ULTRA 7 PROCESSORS, 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.

The 5th generation Intel® Xeon® Scalable Processor Family is based on Emerald Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_CFH.

The Intel® Core™ Ultra 7 processor is based on Meteor Lake hybrid architecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_AAH.

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 2-39[1]. 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 the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
MTRR Capability (R/O, Architectural) See Table 2-2		Thread
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	

1. 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 the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal threshold #1 Status (R/O) See Table 2-2.	
7	Thermal threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
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.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1		Package

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.		Thread
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org.		Core
0	Reserved.	
1	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).	Package
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.	
Register Address: 300H, 768	MSR_SGXOWNEREPOCH0	
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.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNEREPOCH1	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 20.2.4, "Architectural Performance Monitoring Version 4."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Thread
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Thread
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Thread
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Thread
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
54:35	Reserved	
55	Trace_ToPA_PMI	Thread
57:56	Reserved.	
58	LBR_Frz	Thread
59	CTR_Frz	Thread
60	ASCI	Thread
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 20.2.4, "Architectural Performance Monitoring Version 4."		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Thread

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI.	Thread
57:56	Reserved.	
58	Set 1 to clear LBR_Frz.	Thread
59	Set 1 to clear CTR_Frz.	Thread
60	Set 1 to clear ASCI.	Thread
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 20.2.4, "Architectural Performance Monitoring Version 4."		
0	Set 1 to cause Ovf_PMC0 = 1.	Thread
1	Set 1 to cause Ovf_PMC1 = 1.	Thread
2	Set 1 to cause Ovf_PMC2 = 1.	Thread
3	Set 1 to cause Ovf_PMC3 = 1.	Thread
4	Set 1 to cause Ovf_PMC4=1 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to cause Ovf_PMC5=1 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to cause Ovf_PMC6=1 (if CPUID.0AH:EAX[15:8] > 6).	Thread
7	Set 1 to cause Ovf_PMC7=1 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to cause Ovf_FixedCtr0 = 1.	Thread
33	Set 1 to cause Ovf_FixedCtr1 = 1.	Thread
34	Set 1 to cause Ovf_FixedCtr2 = 1.	Thread
54:35	Reserved.	
55	Set 1 to cause Trace_ToPA_PMI = 1.	Thread
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	Thread
59	Set 1 to cause CTR_Frz = 1.	Thread
60	Set 1 to cause ASCI = 1.	Thread
61	Set 1 to cause Ovf_Uncore.	Thread
62	Set 1 to cause Ovf_BufDSSAVE.	Thread

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Thread
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W)		Thread
2:0	Event Code Select	
3	Reserved	
4	Event Code Select High	
7:5	Reserved.	
19:8	IDQ_Bubble_Length Specifier	
22:20	IDQ_Bubble_Width Specifier	
63:23	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Thread
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.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Thread
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Thread
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
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.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved	
63:5	CR3[63:5] value to match	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Thread

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_COUNTER	
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.		Platform
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.Enery_Status_Unit.	
63:32	Reserved.	
Register Address: 64EH, 1614	MSR_PPERF	
Productive Performance Count (R/O)		Thread
63:0	Hardware's view of workload scalability. See Section 15.4.5.1.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)	Package
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).	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) 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 (R0) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
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.	
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 the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 652H, 1618	MSR_PKG_HDC_CONFIG	
HDC Configuration (R/W)		Package
2:0	PKG_Cx_Monitor Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY.	
63: 3	Reserved.	
Register Address: 653H, 1619	MSR_CORE_HDC_RESIDENCY	
Core HDC Idle Residency (R/O)		Core
63:0	Core_Cx_Duty_Cycle_Cnt	
Register Address: 655H, 1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle (R/O)		Package
63:0	Pkg_C2_Duty_Cycle_Cnt	
Register Address: 656H, 1622	MSR_PKG_HDC_DEEP_RESIDENCY	
Package Cx HDC Idle Residency (R/O)		Package
63:0	Pkg_Cx_Duty_Cycle_Cnt	
Register Address: 658H, 1624	MSR_WEIGHTED_CORE_C0	
Core-count Weighted C0 Residency (R/O)		Package
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.	
Register Address: 659H, 1625	MSR_ANY_CORE_C0	
Any Core C0 Residency (R/O)		Package
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.	
Register Address: 65AH, 1626	MSR_ANY_GFXE_C0	
Any Graphics Engine C0 Residency (R/O)		Package
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.	
Register Address: 65BH, 1627	MSR_CORE_GFXE_OVERLAP_C0	
Core and Graphics Engine Overlapped C0 Residency (R/O)		Package
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.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
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.		Platform
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.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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, and the unit is specified in MSR_RAPL_POWER_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.	
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.	
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.12.		Thread
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	
Last Branch Record 19From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	Inefficient Operation Status (R0) When set, processor graphics frequency is operating below target frequency.	
15:13	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.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/Platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) 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 the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 6D0H, 1744	MSR_LASTBRANCH_16_TO_IP	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.12.		Thread
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 15.4.2, "Enabling HWP."		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 15.4.3, "HWP Performance Range and Dynamic Capabilities."		Thread
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Section 15.4.4, "Managing HWP."		Package
Register Address: 773H, 1907	IA32_HWP_INTERRUPT	
See Section 15.4.6, "HWP Notifications."		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 15.4.4, "Managing HWP."		Thread
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)	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:43	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 15.4.5, "HWP Feedback."		Thread
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Thread
Register Address: DA0H, 3488	IA32_XSS	
See Table 2-2.		Thread
Register Address: DB0H, 3504	IA32_PKG_HDC_CTL	
See Section 15.5.2, "Package level Enabling HDC."		Package
Register Address: DB1H, 3505	IA32_PM_CTL1	
See Section 15.5.3, "Logical-Processor Level HDC Control."		Thread
Register Address: DB2H, 3506	IA32_THREAD_STALL	
See Section 15.5.4.1, "IA32_THREAD_STALL."		Thread
Register Address: DC0H, 3520	MSR_LBR_INFO_0	
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.9.1, "LBR Stack."		Thread
Register Address: DC1H, 3521	MSR_LBR_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC2H, 3522	MSR_LBR_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC3H, 3523	MSR_LBR_INFO_3	
Last Branch Record 3 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC4H, 3524	MSR_LBR_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC5H, 3525	MSR_LBR_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC6H, 3526	MSR_LBR_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC7H, 3527	MSR_LBR_INFO_7	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Last Branch Record 7 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC8H, 3528	MSR_LBR_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC9H, 3529	MSR_LBR_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCAH, 3530	MSR_LBR_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCBH, 3531	MSR_LBR_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCCH, 3532	MSR_LBR_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCDH, 3533	MSR_LBR_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCEH, 3534	MSR_LBR_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCFH, 3535	MSR_LBR_INFO_15	
Last Branch Record 15 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD0H, 3536	MSR_LBR_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD1H, 3537	MSR_LBR_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD2H, 3538	MSR_LBR_INFO_18	
Last Branch Record 18 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD3H, 3539	MSR_LBR_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: DD4H, 3540	MSR_LBR_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD5H, 3541	MSR_LBR_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD6H, 3542	MSR_LBR_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD7H, 3543	MSR_LBR_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD8H, 3544	MSR_LBR_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD9H, 3545	MSR_LBR_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDAH, 3546	MSR_LBR_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDBH, 3547	MSR_LBR_INFO_27	
Last Branch Record 27 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDCH, 3548	MSR_LBR_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDDH, 3549	MSR_LBR_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDEH, 3550	MSR_LBR_INFO_30	
Last Branch Record 30 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDFH, 3551	MSR_LBR_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread

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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSEL0	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSEL0	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSEL0	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	

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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSEL0	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSEL0	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
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.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	

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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 80H, 128	MSR_TRACE_HUB_STH_ACPIBAR_BASE	
NPK Address Used by AET Messages (R/W)		Package
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.	
Register Address: 1F4H, 500	MSR_PRMRR_PHYS_BASE	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MemType PRMRR BASE MemType.	
11:3	Reserved.	
45:12	Base PRMRR Base Address.	
63:46	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)		Core
9:0	Reserved.	
10	Lock Lock bit for the PRMRR.	
11	VLD Enable bit for the PRMRR.	
45:12	Mask PRMRR MASK bits.	

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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:46	Reserved.	
Register Address: 1FBH, 507	MSR_PRMRR_VALID_CONFIG	
Valid PRMRR Configurations (R/W)		Core
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.	
Register Address: 2F4H, 756	MSR_UNCORE_PRMRR_PHYS_BASE[1]	
(R/W) The PRMRR range is used to protect the processor reserved memory from unauthorized reads and writes. Any IO access to this range is aborted. This register controls the location of the PRMRR range by indicating its starting address. It functions in tandem with the PRMRR mask register.		Package
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.	
Register Address: 2F5H, 757	MSR_UNCORE_PRMRR_PHYS_MASK[1]	
(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.		Package
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.	
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.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package
6:0	MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.	
7	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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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 supersede prior generation tables.

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
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. Available only if CPUID.(EAX=07H, ECX=0H): ECX[30] = 1.	
18	SGX Global Functions Enable (R/WL)	
63:21	Reserved.	
Register Address: 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.	

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
26	CntAndMode When set to '1', the overall branch monitoring event triggering condition is true only if all enabled counters' threshold conditions are true. When '0', the threshold tripping condition is true if any enabled counters' threshold is true.	
63:27	Reserved.	
Register Address: 351H, 849	MSR_BR_DETECT_STATUS	
Branch Monitoring Global Status (R/W)		
0	Branch Monitoring Event Signaled When set to '1', Branch Monitoring event signaling is blocked until this bit is cleared by software.	
1	LBRsValid This status bit is set to '1' if the LBR state is considered valid for sampling by branch monitoring software.	
7:2	Reserved.	
8	CntrHit0 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.	
9	CntrHit1 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.	

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
15:10	Reserved. Reserved for additional branch monitoring counters threshold hit status.	
25:16	CountWindow 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.	
31:26	Reserved. Reserved for future extension of CountWindow.	
39:32	Count0 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). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	
47:40	Count1 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). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	
63:48	Reserved.	
Register Address: 354H—355H, 852—853	MSR_BR_DETECT_COUNTER_CONFIG_i	
Branch Monitoring Detect Counter Configuration (R/W)		
0	CntrEn Enable counter.	
7:1	CntrEvSel Event select (other values #GP) '0000000 = RETs. '0000001 = RET-CALL bias. '0000010 = RET mispredicts. '0000011 = Branch (all) mispredicts. '0000100 = Indirect branch mispredicts. '0000101 = Far branch instructions.	
14:8	CntrThreshold 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.	

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Package C3 Residency Counter (R/O)		Package
63:0	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package
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.	
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Core C1 Residency Counter (R/O)		Core
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.	
Register Address: 662H, 1634	MSR_CORE_C3_RESIDENCY	
Core C3 Residency Counter (R/O)		Core
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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSEL0	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSEL0	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 708H, 1800	MSR_UNC_CBO_1_PERFEVTSEL0	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 709H, 1801	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 70AH, 1802	MSR_UNC_CBO_1_PERFCTR0	

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 70BH, 1803	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_2_PERFEVTSEL0	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 713H, 1811	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_3_PERFEVTSEL0	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 719H, 1817	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 71AH, 1818	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 71BH, 1819	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_4_PERFEVTSEL0	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 722H, 1826	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 723H, 1827	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_5_PERFEVTSEL0	
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 729H, 1833	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 72AH, 1834	MSR_UNC_CBO_5_PERFCTR0	
Uncore C-Box 5, Performance Counter 0		Package
Register Address: 72BH, 1835	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_6_PERFEVTSEL0	
Uncore C-Box 6, Counter 0 Event Select MSR		Package

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 731H, 1841	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_6_PERFCTR0	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 733H, 1843	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_7_PERFEVTSEL0	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 739H, 1849	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 73AH, 1850	MSR_UNC_CBO_7_PERFCTR0	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 73BH, 1851	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
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.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
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 supersede prior generation tables.

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
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.	
Register Address: 48H, 72	IA32_SPEC_CTRL	
See Table 2-2.		Core
Register Address: 49H, 73	IA32_PREDICT_CMD	
See Table 2-2.		Thread
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Thread
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Thread
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Thread
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	
See Table 2-2.		Thread
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O) This MSR indicates if WRMSR 0x79 failed to configure PRM memory and gives a hint to debug BIOS.		Package
15:0	Error Codes (R/O)	Package
30:16	Reserved.	
31	MCU Partial Success (R/O) When set to 1, WRMSR 0x79 skipped part of the functionality during BIOS.	Thread
Register Address: A5H, 165	MSR_FIT_BIOS_ERROR	
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.		Thread
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.	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
62:17	Reserved.	
63	LOCK (R/W) When set to 1, writes to this MSR will be skipped.	
Register Address: 10BH, 267	IA32_FLUSH_CMD	
See Table 2-2.		Thread
Register Address: 151H, 337	MSR_BIOS_DONE	
BIOS Done (R/WO)		Thread
0	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.	Thread
1	Package BIOS Done Indication (R/O) When set to 1, all threads in the package have bit 0 of this MSR set.	Package
31:2	Reserved.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Write Data to a Crash Log Configuration		Thread
0	CDDIS: CrashDump_Disable If set, indicates that Crash Dump is disabled.	
63:1	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 30CH, 780	IA32_FIXED_CTR3	
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.		Thread
Register Address: 329H, 809	MSR_PERF_METRICS	
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).		Thread
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.	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 3F2H, 1010	MSR_PEBS_DATA_CFG	
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.		Thread
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.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W)		Core
0	L1 Scrubbing Enable When set to 1, enable L1 scrubbing.	
31:1	Reserved.	
Register Address: 657H, 1623	MSR_FAST_UNCORE_MSRS_CTL	
Fast WRMSR/RDMSR Control MSR (R/W)		Thread
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.	
Register Address: 65EH, 1630	MSR_FAST_UNCORE_MSRS_STATUS	
Indication of Uncore MSRs, Post Write Activates		Thread

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 65FH, 1631	MSR_FAST_UNCORE_MSRS_CAPABILITY	
Fast WRMSR/RDMSR Enumeration MSR (R/O)		Thread
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.	
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Table 2-2.		Package
Register Address: 775H, 1909	IA32_PECI_HWP_REQUEST_INFO	
See Table 2-2.		Thread
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Table 2-2.		Thread

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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O)		Package
15:0	Error Codes	
31:16	Reserved.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) This MSR indicates if WRMSR 79H failed to configure PRM memory and gives a hint to debug BIOS.		Thread
30:0	Reserved.	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
31	MCU Partial Success When set to 1, WRMSR 79H skipped part of the functionality during BIOS.	
63:32	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
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.		Package
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.	
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTLS3	
IA32_VMX_PROCBASED_CTLS3 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.		Core
0	LOADIWKEY This control determines whether executions of LOADIWKEY cause VM exits.	
63:1	Reserved.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.		Package
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.	
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.	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
62:32	Not in use.	
63	Reserved.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PL0_SSP	
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.		
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 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.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		
Register Address: 990H, 2448	IA32_COPY_STATUS[1]	
See Table 2-2.		Thread
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS[1]	
See Table 2-2.		Platform
Register Address: C82H, 3202	IA32_L2_QOS_CFG	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
IA32_CR_L2_QOS_CFG This MSR provides software an enumeration of the parameters that L2 QoS (Intel RDT) support in any particular implementation.		Core
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.	
Register Address: D10H—D17H, 3220—3351	IA32_L2_QOS_MASK_[0-7]	
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."		Package
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.	
Register Address: D91H, 3473	IA32_COPY_LOCAL_TO_PLATFORM[1]	
See Table 2-2.		Thread
Register Address: D92H, 3474	IA32_COPY_PLATFORM_TO_LOCAL[1]	
See Table 2-2.		Thread

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 2-44[1] 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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
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.	
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTLS	
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: C7H, 199	IA32_PMC6	
General Performance Counter 6 (R/W) See Table 2-2.		Core
Register Address: C8H, 200	IA32_PMC7	
General Performance Counter 7 (R/W) See Table 2-2.		Core
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
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.		Package
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.	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
4	UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).	
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.	
Register Address: E1H, 225	IA32_UMWAIT_CONTROL	
UMWAIT Control (R/W) See Table 2-2.		
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 18CH, 396	IA32_PERFEVTSEL6	
See Table 2-20.		Core
Register Address: 18DH, 397	IA32_PERFEVTSEL7	
See Table 2-20.		Core
Register Address: 195H, 405	IA32_OVERCLOCKING_STATUS	
Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR. See Table 2-2.		Package
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
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.		Package
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.	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 493H, 1171	IA32_VMX_EXIT_CTLS2	
See Table 2-2.		
Register Address: 4C7H, 1223	IA32_A_PMC6	
Full Width Writable IA32_PMC6 Alias (R/W) See Table 2-2.		
Register Address: 4C8H, 1224	IA32_A_PMC7	
Full Width Writable IA32_PMC7 Alias (R/W) See Table 2-2.		
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	
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.		Package
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.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
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.		Module

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
Memory Encryption Capability MSR See Table 2-2.		
Register Address: 1200H—121FH, 4608—4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H—151FH, 5376—5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H—161FH, 5632—5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		
Register Address: 17D2H, 6098	IA32_THREAD_FEEDBACK_CHAR	
Thread Feedback Characteristics (R/O) See Table 2-2.		
Register Address: 17D4H, 6100	IA32_HW_FEEDBACK_THREAD_CONFIG	
Hardware Feedback Thread Configuration (R/W) See Table 2-2.		
Register Address: 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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 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.	
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W) See Table 2-39.		Thread
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W)		Thread
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.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core
Register Address: D10H—D17H, 3220—3351	IA32_L2_QOS_MASK_[0-7]	
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."		Core

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: D10H—D1FH, 3220—3359	IA32_L2_QOS_MASK_[0-15]	
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."		Module
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.	
Register Address: 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.	
Register Address: 14C1H—14C6H, 5313 —5318	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Core
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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	
Register Address: 2000H, 8192	MSR_UNC_CBO_0_PERFEVTSEL0	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 2001H, 8193	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 2002H, 8194	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 2003H, 8195	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 2008H, 8200	MSR_UNC_CBO_1_PERFEVTSEL0	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 2009H, 8201	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 200AH, 8202	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 200BH, 8203	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 2010H, 8208	MSR_UNC_CBO_2_PERFEVTSEL0	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 2011H, 8209	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 2012H, 8210	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 2013H, 8211	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 2018H, 8216	MSR_UNC_CBO_3_PERFEVTSEL0	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 2019H, 8217	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 201AH, 8218	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 201BH, 8219	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 2020H, 8224	MSR_UNC_CBO_4_PERFEVTSEL0	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 2021H, 8225	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 2022H, 8226	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 2023H, 8227	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 2028H, 8232	MSR_UNC_CBO_5_PERFEVTSEL0	
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 2029H, 8233	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 202AH, 8234	MSR_UNC_CBO_5_PERFCTR0	
Uncore C-Box 5, Performance Counter 0		Package
Register Address: 202BH, 8235	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 2030H, 8240	MSR_UNC_CBO_6_PERFEVTSEL0	
Uncore C-Box 6, Counter 0 Event Select MSR		Package
Register Address: 2031H, 8241	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 2032H, 8242	MSR_UNC_CBO_6_PERFCTR0	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 2033H, 8243	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 2038H, 8248	MSR_UNC_CBO_7_PERFEVTSEL0	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 2039H, 8249	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 203AH, 8250	MSR_UNC_CBO_7_PERFCTR0	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 203BH, 8251	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: 2040H, 8256	MSR_UNC_CBO_8_PERFEVTSEL0	
Uncore C-Box 8, Counter 0 Event Select MSR		Package
Register Address: 2041H, 8257	MSR_UNC_CBO_8_PERFEVTSEL1	
Uncore C-Box 8, Counter 1 Event Select MSR		Package
Register Address: 2042H, 8258	MSR_UNC_CBO_8_PERFCTR0	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Uncore C-Box 8, Performance Counter 0		Package
Register Address: 2043H, 8259	MSR_UNC_CBO_8_PERFCTR1	
Uncore C-Box 8, Performance Counter 1		Package
Register Address: 2048H, 8264	MSR_UNC_CBO_9_PERFEVTSEL0	
Uncore C-Box 9, Counter 0 Event Select MSR		Package
Register Address: 2049H, 8265	MSR_UNC_CBO_9_PERFEVTSEL1	
Uncore C-Box 9, Counter 1 Event Select MSR		Package
Register Address: 204AH, 8266	MSR_UNC_CBO_9_PERFCTR0	
Uncore C-Box 9, Performance Counter 0		Package
Register Address: 204BH, 8267	MSR_UNC_CBO_9_PERFCTR1	
Uncore C-Box 9, Performance Counter 1		Package
Register Address: 2FD0H, 12240	MSR_UNC_ARB_0_PERFEVTSEL0	
Uncore Arb Unit 0, Counter 0 Event Select MSR		Package
Register Address: 2FD1H, 12241	MSR_UNC_ARB_0_PERFEVTSEL1	
Uncore Arb Unit 0, Counter 1 Event Select MSR		Package
Register Address: 2FD2H, 12242	MSR_UNC_ARB_0_PERFCTR0	
Uncore Arb Unit 0, Performance Counter 0		Package
Register Address: 2FD3H, 12243	MSR_UNC_ARB_0_PERFCTR1	
Uncore Arb Unit 0, Performance Counter 1		Package
Register Address: 2FD4H, 12244	MSR_UNC_ARB_0_PERF_STATUS	
Uncore Arb Unit 0, Performance Status		Package
Register Address: 2FD5H, 12245	MSR_UNC_ARB_0_PERF_CTRL	
Uncore Arb Unit 0, Performance Control		Package
Register Address: 2FD8H, 12248	MSR_UNC_ARB_1_PERFEVTSEL0	
Uncore Arb Unit 1, Counter 0 Event Select MSR		Package
Register Address: 2FD9H, 12249	MSR_UNC_ARB_1_PERFEVTSEL1	
Uncore Arb Unit 1, Counter 1 Event Select MSR		Package
Register Address: 2FDAH, 12250	MSR_UNC_ARB_1_PERFCTR0	
Uncore Arb Unit 1, Performance Counter 0		Package
Register Address: 2FDBH, 12251	MSR_UNC_ARB_1_PERFCTR1	
Uncore Arb Unit 1, Performance Counter 1		Package
Register Address: 2FDCH, 12252	MSR_UNC_ARB_1_PERF_STATUS	
Uncore Arb Unit 1, Performance Status		Package
Register Address: 2FDDH, 12253	MSR_UNC_ARB_1_PERF_CTRL	
Uncore Arb Unit 1, Performance Control		Package
Register Address: 2FDEH, 12254	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 2FDFH, 12255	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	
Register Address: 2FF0H, 12272	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
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.	
Register Address: 2FF2H, 12274	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
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.	
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
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.		Core
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 MWAT(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.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count.	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
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.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) 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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
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.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
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.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
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: ▪ 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.		Package
7:0	RATIO_0 Defines ratio limits.	
15:8	RATIO_1 Defines ratio limits.	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
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.		Package
7:0	NUMCORE_0 Defines the active core ranges for each frequency point.	
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.	
Register Address: 280H, 640	IA32_MC0_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 400H, 1024	IA32_MC0_CTL	
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.		Core

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 401H, 1025	IA32_MC0_STATUS	
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.		Core
Register Address: 402H, 1026	IA32_MC0_ADDR	
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.		Core
Register Address: 403H, 1027	IA32_MC0_MISC	
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.		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
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.		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
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.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
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.		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
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.		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
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.		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
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.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
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.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
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.		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
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.		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
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.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
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.		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
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.		Package
Register Address: 411H, 1041	IA32_MC4_STATUS	
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.		Package
Register Address: 412H, 1042	IA32_MC4_ADDR	
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.		Package
Register Address: 413H, 1043	IA32_MC4_MISC	
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.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
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.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
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.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
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.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
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.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
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.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
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.		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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 41AH, 1050	IA32_MC6_ADDR	
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.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
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.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
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.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
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.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
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.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
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.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
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.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
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.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
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.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
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.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
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.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
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.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
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.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
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.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
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.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
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.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
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.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
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.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
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.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
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.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
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.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
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.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
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.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
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.		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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 433H, 1075	IA32_MC12_MISC	
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.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
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.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
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.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
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.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
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.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
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.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
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.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
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.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
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.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
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.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
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.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
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.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
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.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
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.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
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.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
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.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
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.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
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.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
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.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
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.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
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.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
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.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
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.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
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.		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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 44CH, 1100	IA32_MC19_CTL	
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.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
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.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
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.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
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.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 15.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	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).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 15.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	

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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
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.		Package
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.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.		Thread
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.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_2	
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W). If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8.		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: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	**Bit Description**	**Scope**
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 15 enforcement.	
63:20	Reserved.	

2.17.7 MSRs Specific to the 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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 612H, 1554	MSR_PACKAGE_ENERGY_TIME_STATUS	
Package energy consumed by the entire CPU (R/W)		Package
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.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
Allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.		Package
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 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.	
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM Power Parameters (R/W)		Package

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 (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		

2.17.8 MSRs Specific to the 4th and 5th Generation Intel® Xeon® Scalable Processor Families

The 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_8FH) and the 5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_CFH) both support the MSRs listed in Section 2.17, "MSRs In the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 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 and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register (R/W)		Core
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.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) See Table 2-45.		Thread
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTLS	
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
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.		Core
0	Reserved: returns zero.	
1	Reserved: returns zero.	
2	INTEGRITY_CAPABILITIES When set to 1, the processor supports MSR_INTEGRITY_CAPABILITIES.	
3	RSM_IN_CPL0_ONLY Indicates that RSM will only be allowed in CPL0 and will #GP for all non-CPL0 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 and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: E1H, 225	IA32_UMWAIT_CONTROL	
UMWAIT Control (R/W) See Table 2-2.		
Register Address: EDH, 237	MSR_RAR_CONTROL	
RAR Control (R/W)		Thread
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.	
Register Address: EEH, 238	MSR_RAR_ACTION_VECTOR_BASE	
Pointer to RAR Action Vector (R/W)		Thread
63:MAXPHYADDR	Reserved.	
MAXPHYADDR-1:6	VECTOR_PHYSICAL_ADDRESS Pointer to the physical address of the 64B aligned RAR action vector.	
5:0	Reserved.	
Register Address: EFH, 239	MSR_RAR_PAYLOAD_TABLE_BASE	
Pointer to Base of RAR Payload Table (R/W)		Thread
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.	
Register Address: F0H, 240	MSR_RAR_INFO	
Read Only RAR Information (RO)		Thread
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.	
Register Address: 105H, 261	MSR_CORE_BIST	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Core BIST (R/W) Controls Array BIST activation and status checking as part of FUSA.		Core
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).	
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 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.	
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
	Primary Maximum Turbo Ratio Limit (R/W) See Table 2-46.	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
	See Table 2-50.	Package
Register Address: 1C4H, 452	IA32_XFD	
	Extended Feature Detect (R/W) See Table 2-2.	
Register Address: 1C5H, 453	IA32_XFD_ERR	
	XFD Error Code (R/W) See Table 2-2.	
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	
	COPY_SCAN_HASHES (W)	Die
63:0	SCAN_HASH_ADDR Contains the linear address of the SCAN Test HASH Binary loaded into memory.	
Register Address: 2C3H, 707	MSR_SCAN_HASHES_STATUS	
	SCAN_HASHES_STATUS (R/O)	
15:0	CHUNK_SIZE Chunk size of the test in KB.	Die
23:16	NUM_CHUNKS Total number of chunks.	Die
31:24	Reserved: all zeros.	
39:32	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.	Thread
50:40	Reserved: set to all zeros.	
62:51	MAX_CORE_LIMIT Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	Die

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
63	Valid Valid bit is set when COPY_SCAN_HASHES has completed successfully.	Die
Register Address: 2C4H, 708	MSR_AUTHENTICATE_AND_COPY_CHUNK	
AUTHENTICATE_AND_COPY_CHUNK (W)		Die
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.	
Register Address: 2C5H, 709	MSR_CHUNKS_AUTHENTICATION_STATUS	
CHUNKS_AUTHENTICATION_STATUS (R/O)		
7:0	VALID_CHUNKS Total number of Valid (authenticated) chunks.	Die
15:8	TOTAL_CHUNKS Total number of chunks.	Die
31:16	Reserved: all zeros.	
39:32	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.	Thread
63:40	Reserved: set to all zeros.	
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
ACTIVATE_SCAN (W)		Thread
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.	
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.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
SCAN_STATUS (R/O)		

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
7:0	CHUNK_NUM SCAN Chunk that was reached.	Core
15:8	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive). Maps to same field in WRMSR(ACTIVATE_SCAN).	Core
31:16	Reserved: return all zeros.	
39:32	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.	Thread
61:40	Reserved: return all zeros.	
62	SCAN_CONTROL_ERROR Scan-System-Controller malfunction.	Core
63	SCAN_SIGNATURE_ERROR Core failed SCAN-SIGNATURE checking for this chunk.	Core
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
SCAN_MODULE_ID (R/O)		Module
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.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
LAST_SAF_WP (R/O)		Core
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.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
	INTEGRITY_CAPABILITIES (R/O)	Module
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.	
Register Address: 410H, 1040	IA32_MC4_CTL	
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.		Package
Register Address: 411H, 1041	IA32_MC4_STATUS	
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.		Package
Register Address: 412H, 1042	IA32_MC4_ADDR	
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.		Package
Register Address: 413H, 1043	IA32_MC4_MISC	
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.		Package
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTLS3	
Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		
Register Address: 493H, 1171	IA32_VMX_EXIT_CTLS2	
Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Table 2-2.		
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
31:0	Energy in 61 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_STATUS	
Platform Energy Status (R/O)		Package
31:0	TOTAL_ENERGY_CONSUMED Total energy consumption in J (32.0), in 10nsec units.	
63:32	TIME_STAMP Time stamp (U32.0).	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W-L)		Package
16:0	POWER_LIMIT_1 The average power limit value that the platform must not exceed over a time window as specified by the 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_RAPL_POWER_UNIT.	
17	POWER_LIMIT_1_EN When set, the processor can apply control policies such that the platform average power does not exceed the Power_Limit_1 value over an exponential weighted moving average of the time window.	
18	CRITICAL_POWER_CLAMP_1 When set, the processor can go below the OS-requested P States to maintain the power below the specified Power_Limit_1 value.	
25:19	POWER_LIMIT_1_TIME This indicates the time window over which the Power_Limit_1 value should be maintained. 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, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:26	Reserved.	
48:32	POWER_LIMIT_2 This is the Duration Power limit value that the platform must not exceed. The unit is specified in MSR_RAPL_POWER_UNIT.	
49	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 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
50	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.	
57:51	POWER_LIMIT_2_TIME This indicates the time window over which the Power_Limit_2 value should be maintained. This field has the same format as the POWER_LIMIT_1_TIME field.	
62:58	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 665H, 1637	MSR_PLATFORM_POWER_INFO	
Platform Power Information (R/W)		Package
16:0	MAX_PPL1 Maximum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
31:17	MIN_PPL1 Minimum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
48:32	MAX_PPL2 Maximum PP L2 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
55:49	MAX_TW Maximum time window. The unit is specified in MSR_RAPL_POWER_UNIT.	
62:56	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 666H, 1638	MSR_PLATFORM_RAPL_SOCKET_PERF_STATUS	
Platform RAPL Socket Performance Status (R/O)		Package
31:0	Count of limited performance due to platform RAPL limit.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PL0_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 and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 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.		
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
Memory Encryption Capability MSR See Table 2-2.		
Register Address: 985H, 2437	IA32_UINTR_RR	
User Interrupt Request Register (R/W) See Table 2-2.		
Register Address: 986H, 2438	IA32_UINTR_HANDLER	
User Interrupt Handler Address (R/W) See Table 2-2.		
Register Address: 987H, 2439	IA32_UINTR_STACKADJUST	
User Interrupt Stack Adjustment (R/W) See Table 2-2.		
Register Address: 988H, 2440	IA32_UINTR_MISC	
User-Interrupt Target-Table Size and Notification Vector (R/W) See Table 2-2.		
Register Address: 989H, 2441	IA32_UINTR_PD	
User Interrupt PID Address (R/W) See Table 2-2.		
Register Address: 98AH, 2442	IA32_UINTR_TT	
User-Interrupt Target Table (R/W) See Table 2-2.		

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SEL0	
Uncore B-box 1 perfmon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 perfmon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 perfmon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTR2	
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vperfmon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTR3	
Uncore B-box 1 perfmon counter MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box perfmon local box overflow control MSR.		Package
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
See Table 2-2.		
Register Address: C90H—C9EH, 3216—3230	IA32_L3_QOS_MASK_0 through IA32_L3_QOS_MASK_14	
See Table 2-50.		Package
Register Address: D10H—D17H, 3344—3351	IA32_L2_QOS_MASK_[0-7]	
IA32_CR_L2_QOS_MASK_[0-7] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. See Table 2-2.		Core
Register Address: D93H, 3475	IA32_PASID	
See Table 2-2.		
Register Address: 1200H—121FH, 4608—4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 1406H, 5126	IA32_MCU_CONTROL	
See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H–151FH, 5376–5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H–161FH, 5632–5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		

2.17.9 MSRs Introduced in the Intel® Core™ Ultra 7 Processor Supporting Performance Hybrid Architecture

Table 2-53 lists additional MSRs for the Intel Core Ultra 7 processor with a CPUID Signature DisplayFamily_Display-Model value of 06_AAH. Table 2-54 lists the MSRs unique to the processor P-core. Table 2-55 lists the MSRs unique to the processor E-core.

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
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."	
63:30	Reserved.	
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread. See Table 2-2.		
Register Address: 80H, 128	MSR_TRACE_HUB_STH_ACPIBAR_BASE	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
MSR_TRACE_HUB_STH_ACPIBAR_BASE (R/W) This register is used by BIOS to program Trace Hub STH base address that will be used by AET messages.		Thread
0	LOCK Lock bit. If set, this MSR cannot be re-written anymore. The lock bit has to be set in order for the AET packets to be directed to Trace Hub MMIO.	
17:1	Reserved.	
45:18	ADDRESS AET target address in Trace Hub MMIO space.	
63:46	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration (R/W)		Core
3:0	PKG_C_STATE_LIMIT 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 may be 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	
7:4	MAX_CORE_C_STATE Possible values are: 0000—reserved; 0001—C1; 0010—C3, 0011—C6.	
9:8	Reserved.	
10	IO_MWAIT_REDIRECTION_ENABLE When set, will map IO_read instructions sent to IO registers PMG_IO_BASE_ADDR.PMB0+0/1/2 to MWAIT(C2,3,4) instructions; applies to deepc4 too.	
14:11	Reserved.	
15	CFG_LOCK When set, locks bits 15:0 of this register for further writes, until the next reset occurs.	
24:16	Reserved.	
25	C3_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
26	C1_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	ENABLE_C3_UNDEMOTION Enable Un-Demotion from Demoted C3.	
28	ENABLE_C1_UNDEMOTION Enable Un-Demotion from Demoted C1.	
29	ENABLE_PKGC_AUTODEMOTION Enable Package C-State Auto-Demotion. It enables use of the history of past package C-state depth and residence, as a factor in determining C-State depth.	
30	ENABLE_PKGC_UNDEMOTION Enable Package C-State Un-Demotion. It enables considering cases where demotion was the incorrect decision in determining C-State depth.	
31	TIMED_MWAIT_ENABLE When set, enables Timed MWAIT feature. MWAIT would #GP on attempts to do setup MWAIT timer if this bit is not set.	
63:32	Reserved.	
Register Address: E4H, 228	MSR_IO_CAPTURE_BASE	
IO Capture Base (R/W) Power Management IO Redirection in C-state. See http://biosbits.org.		Core
15:0	LVL_2_BASE_ADDRESS Specifies the base address visible to software for IO redirection. If MSR_PKG_CST_CONFIG_CONTROL.IO_MWAIT_REDIRECTION_ENABLE, 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	CST_RANGE 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.IO_MWAIT_REDIRECTION_ENABLE: 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.	
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Feature Configuration (R/W)		Core
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
1	AESNI_DISABLE This bit disables Advanced Encryption Standard feature on this processor core. To disable AES, BIOS will write '11 to this MSR on every core.	
63:2	Reserved.	
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Feature Enable (R/W) Miscellaneous enables for thread specific features.		Thread
0	CPUID_GP_ON_CPL_GT_0 Causes CPUID to #GP if CPL greater than 0 and not in SMM.	
63:1	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target (R/W) Legacy register holding temperature related constants for Platform use.		Package
6:0	TCC Offset Time Window Describes the RATL averaging time window.	
7	TCC Offset Clamping Bit When enabled will allow RATL throttling below P1.	
15:8	Temperature Control Offset Fan Temperature Target Offset (a.k.a. T-Control) indicates the relative offset from the Thermal Monitor Trip Temperature at which fans should be engaged.	
23:16	TCC Activation Temperature The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
30:24	TCC Activation Offset 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.	
31	LOCKED When set, this entire register becomes read-only.	
63:2	Reserved.	
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
PREFETCH Control (R/W) Prefetch disable bits.		Thread
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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	DCU_NEXT_PAGE_PREFETCH_DISABLE If 1, disables Next Page prefetcher.	
5	AMP_PREFETCH_DISABLE If 1, disables L2 Adaptive Multipath Probability (AMP) prefetcher.	
6	LLC_PAGE_PREFETCH_DISABLE If 1, disables the LLC Page prefetcher.	
7	AOP_PREFETCH_DISABLE	
8	STREAM_PREFETCH_CODE_FETCH_DISABLE	
63:9	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
OFFCORE_RSP_0 (R/W) Offcore Response Event Select Register		Thread
0	TRUE_DEMAND_CACHE_LOAD Demand Data Rd = DCU reads (includes partials) that is not tagged homeless.	
1	DEMAND_RFO Demand Instruction fetch = IFU Fetches. ItoM or RFO that is not tagged homeless.	
2	DEMAND_CODE_READ Demand Instruction fetch = IFU Fetches. CRd or CRd_UC.	
3	CORE_MODIFIED_WRITEBACK WBMtoI or WBMtoE.	
4	HW_PREFETCH_MLC_LOAD L2 prefetcher requests triggered by reads from MEC (except those triggered by I-side).	
5	HW_PREFETCH_MLC_RFO L2 prefetcher requests triggered by RFOs.	
6	HW_PREFETCH_MLC_CODE L2 prefetcher requests triggered by I-side requests.	
7	HW_PREFETCH_LLC_LOAD LLC prefetch requests triggered by DRd.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
8	HW_PREFETCH_LLC_RFO LLC prefetch requests triggered by RFO.	
9	HW_PREFETCH_LLC_CODE LLC prefetch requests triggered by CRd.	
10	L1_HWPREFETCH Covers Hardware PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
11	ALL_STREAMING_STORE Write Combining. WCiL or WCiLF.	
12	CORE_NON_MODIFIED_WB WBEFtoI or WBEFtoE.	
13	LLC_PREFETCH LLC prefetch of load/code/RFO.	
14	L1_SWPREFETCH Covers Software PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
15	OTHER Includes CLFlush, CLFlushOPT, CLDemote, CLWB, Enqueue SetMonitor, PortIn, IntA, Lock, SplitLock, Unlock, SpCyc, ClrMonitor, PortOut, IntPriUp, IntLog, IntPhy, EOI, RdCurr, WbStoI, LLCWBInv, LLCInv, NOP, PCOMMIT.	
16	ANY_RESP Match on any response.	
17	SUPPLIER_NONE No Supplier Details. DATA_PRE [6:3] = 0.	
18	LLC_HIT_M_STATE LLC/L3, M-state, DATA_PRE [6:3] = 2.	
19	LLC_HIT_E_STATE LLC/L3, E-state, DATA_PRE [6:3] = 4.	
20	LLC_HIT_S_STATE LLC/L3, S-state, DATA_PRE [6:3] = 6.	
21	LLC_HIT_F_STATE LLC/L3, F-state, DATA_PRE [6:3] = 8.	
22	FAR_MEM_LOCAL Far Memory, Local, DATA_PRE [6:3] = 1.	
23	FAR_MEM_REMOTE_0_HOP Far Memory, Remote 0-hop, DATA_PRE [6:3] = 3.	
24	FAR_MEM_REMOTE_1_HOP Far Memory, Remote 1-hop, DATA_PRE [6:3] = 5.	
25	FAR_MEM_REMOTE_2_PLUS_HOP Far Memory, Rem 2+ hop, DATA_PRE [6:3] = 7.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
26	NEAR_MEM_MISS_LOCAL_NODE LLC Miss Local Node. Near Memory, Local DATA_PRE [6:3] = E.	
27	NEAR_MEM_REMOTE_0_HOP Near Memory, Remote 0-hop, DATA_PRE [6:3] = B	
28	NEAR_MEM_REMOTE_1_HOP Near Memory, Remote 1-hop, DATA_PRE [6:3] = D.	
29	NEAR_MEM_REMOTE_2_PLUS_HOP Near Memory, Remote 2+ hop, DATA_PRE [6:3] = F.	
30	SPL_HIT Snoop Info: SPL-hit, DATA_PRE [2:0] = 6.	
31	SNOOP_NONE No details as to Snoop-related info. Snoop Info: None, DATA_PRE [2:0] = 0.	
32	NOT_NEEDED No snoop was needed to satisfy the request. Snoop Info: Not needed, DATA_PRE [2:0] = 1.	
33	MISS No snoop was needed to satisfy the request. Snoop Info: Miss, DATA_PRE [2:0] = 2.	
34	HIT_NO_FWD A snoop was needed and it Hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. Snoop Info: Hit No Fwd, DATA_PRE [2:0] = 3.	
35	HIT_EF_WITH_FWD A snoop was needed and data was Forwarded from a remote socket. Snoop Info: Hit EF w/Fwd, DATA_PRE [2:0] = 4.	
36	HITM A snoop was needed and it HitMed in local or remote cache. HitM denotes a cache-line was modified before snoop effect. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
37	NON_DRAM Target was non-DRAM system address. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
38	GO_ERR GO-ERR, RspData[3:0] = 0100.	
39	GO_NO_GO GO-NoGO, RspData[3:0] = 0111.	
40	INPKG_MEM_LOCAL In-package Memory, Local, DATA_PRE [6:3] = 9.	
41	INPKG_MEM_NONLOCAL In-package Memory, Non-Local, DATA_PRE [6:3] = C.	
43:42	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
44	UC_LOAD PRd or UCRdF.	
45	UC_STORE WiL.	
46	PARTIAL_STREAMING_STORES WCiL.	
47	FULL_STREAMING_STORES WCiLF.	
48	L1_MODIFIED_WB EVICTION EXTTYPE from MEC.	
49	L2_MODIFIED_WB WBMtoI or WBMtoE.	
50	PSMI MemPushWr_NS (PSMI only).	
51	ITOM ItoM.	
63:52	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
OFFCORE_RSP_1 (R/W) Offcore Response Event Select Register. See MSR_OFFCORE_RSP_0 (at1A6H).		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control (R/W) Various model-specific features enumeration. See http://biosbits.org.		Package
0	Reserved.	
1	ENABLE_HWP_VOTING_RIGHT When set (1), The CPU will take into account thread HWP requests for threads that have voting rights only (ignores thread requests if they do not have voting rights). When reset(0), The CPU will take into account all thread HWP requests, even for threads that don't have voting rights. Setting this bit will cause the HWP Base feature bit to be reported in CPUID as present; clearing will cause it to be reported as non-present.	
5:2	Reserved.	
6	ENABLE_HWP Setting this bit will cause the HWP Base feature bit to report as present in CPUID; clearing this bit will cause CPUID to report the feature as non-present.	
7	ENABLE_HWP_INTERRUPT Setting this bit will cause the HWP Interrupt feature CPUID[6].EAX[8] bit to report as present; clearing will report as non-present.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
8	ENABLE_OUT_OF_BAND_AUTONOMOUS Setting this bit will cause the HWP Autonomous feature bit to report as present; clearing will report as non-present.	
11:9	Reserved.	
12	ENABLE_HWP_EPP Enable HWP EPP. Setting this bit (1) will cause the HWP CPUID[6].EAX[10] Energy Performance Preference bit to report as present (1); clearing will report as non-present (0).	
13	LOCK Setting this bit will prevent the BIOS specific bits from changing until the next reset. i.e., only Bits [0,22] which are meant for OS use can be changed once the LOCK bit is set.	
63:14	Reserved.	
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
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.		Package
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.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Crash Log Control (R/W) Write data to a Crash Log configuration.		Thread
0	CDDIS CrashDump_Disable: If set, indicates that Crash Dump is disabled.	
1	EN_GPRS Collect GPRs on a crash dump. Only meaningful when CDDIS is zero.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
2	EN_GPRS_IN_SMM Collect GPRs in SMM on a crash dump. Only meaningful when CDDIS is zero. EN_GPRS will override this control,	
3	TRIPLE_FAULT_SHUTDOWN Collect a crash log on a triple fault shutdown. Only meaningful when CDDIS is zero.	
63:4	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask (R/W)		Core
9:0	Reserved.	
10	LOCK Once set, this bit prevents software from modifying the PRMRR.	
11	VALID This bit serves as the enable for the PRMRR; the PRMRR must be LOCKed before it can be enabled.	
19:12	Reserved.	
45:20	MASK PRMRR Address Mask.	
63:46	Reserved.	
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register (R/W) See http://biosbits.org.		Package
0	ENABLE_BIDIR_PROCHOT Used to enable or disable the response to PROCHOT# input. When set/enabled, the platform can force the CPU to throttle to a lower power condition such as Pn/Pm by asserting prochot#. When clear/disabled (default), the CPU ignores the status of the prochot input signal.	
1	C1E_ENABLE 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).	
2	SAPM_IMC_C2_POLICY This bit determines if self-refresh activation is allowed when entering Package C2 State. If it is set to 0b, PCODE will keep the FORCE_SR_OFF bit asserted in Package C2 State and allow its negation according to the defined latency negotiations with the PCH and Display Engine in Package C3 and deeper states. Otherwise, self-refresh is allowed in Package C2 State.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
3	FAST_BRK_SNP_EN This bit controls the VID swing rate for the OTHER_SNP_WAKE events that are detected by the iMPH. This is the event that is detected by the iMPH when a non-DMI snoopable request is observed while UCLK domain is not functional. 0b: Use slow VID swing rate. 1b: Use fast VID swing rate.	
17:4	Reserved.	
18	PWR_PERF_PLTFRM_OVR Power performance platform override.	
19	EE_TURBO_DISABLE 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	RTH_DISABLE 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.	
21	DIS_PROCHOT_OUT Prochot output disable.	
22	PROCHOT_RESPONSE Prochhot configurable response enable.	
23	VR_THERM_ALERT_DISABLE_LOCK When set to 1, locks PROCHOT related bits of this MSR. Once set, a reset is required to clear this bit.	
24	VR_THERM_ALERT_DISABLE When set to 1, disables the VR_THERMAL_ALERT signaling.	
25	DISABLE_RING_EE Disable Ring EE.	
26	DISABLE_SA_OPTIMIZATION Disable SA optimization.	
27	DISABLE_OOK Disable OOK.	
28	DISABLE_AUTONOMOUS Disable HWP autonomous mode.	
29	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
30	CSTATE_PREWAKE_DISABLE C-state pre-wake disable.	
63:31	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
19:4	Reserved.	
45:20	BASE PRMRR base address.	
63:46	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
MC29_CTL. See Table 2-2.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
MC29_STATUS. See Table 2-2.		Package
Register Address: 476H, 1142	IA32_MC29_ADDR	
MC29_ADDR. See Table 2-2.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
MC29_MISC. See Table 2-2.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
MC30_CTL. See Table 2-2.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
MC30_STATUS. See Table 2-2.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
MC30_ADDR. See Table 2-2.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
MC30_MISC. See Table 2-2.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
MC31_CTL. See Table 2-2.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
MC31_STATUS. See Table 2-2.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
MC31_ADDR. See Table 2-2.		Package
Register Address: 47FH, 1151	IA32_MC31_MISC	
MC31_MISC. See Table 2-2.		Package

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (R/W) Reports SMM capability enhancement.		Package
0	LOCK When set, locks this register from further changes.	
1	SMM_CPU_SAVE_EN If 0, SMI/RSM will save/restore state in SMRAM If 1, SMI/RSM will save/restore state from SRAM.	
2	SMM_CODE_CHK_EN When clear (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set, 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.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) (R/W) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.		Package
15:0	CURRENT_LIMIT PL4 Value in 0.125 A increments. This field is locked by MSR_VR_CURRENT_CONFIG.LOCK. When the LOCK bit is set to 1, this field becomes Read Only.	
30:16	Reserved.	
31	LOCK This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.	
63:32	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Min/Max Ratio Limits for Uncore LLC and Ring.		Package
6:0	MAX_CLR_RATIO Maximum allowed ratio for the Ring and Last Level Cache (LLC).	
7	Reserved.	
14:8	MIN_CLR_RATIO Minimum allowed ratio for the Ring and Last Level Cache (LLC).	
63:15	Reserved.	
Register Address: 638H, 1592	MSR_PP0_POWER_LIMIT	
MSR_PP0_POWER_LIMIT (R/W) PP0 RAPL power unit control.		Package

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
14:0	IA_PP_PWR_LIM This is the power limitation on the IA cores power plane. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[PWR_UNIT].	
15	PWR_LIM_CTRL_EN This bit must be set in order to limit the power of the IA cores power plane. 0b: IA cores power plane power limitation is disabled. 1b: IA cores power plane power limitation is enabled.	
16	PP_CLAMP_LIM Power Plane Clamping limitation; allow going below P1. 0b: PBM is limited between P1 and P0. 1b: PBM can go below P1.	
23:17	CTRL_TIME_WIN x = CTRL_TIME_WIN[23:22] y = CTRL_TIME_WIN[21:17] The timing interval window is Floating Point number given by 1.x * power(2,y). The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[TIME_UNIT]. The maximal time window is bounded by PACKAGE_POWER_SKU_MSR[PKG_MAX_WIN]. The minimum time window is 1 unit of measurement (as defined above).	
30:24	Reserved.	
31	PP_PWR_LIM_LOCK When set, all settings in this register are locked and are treated as Read Only.	
63:32	Reserved.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Core Performance Limit Reasons Indicator of Frequency Clipping in Processor Cores. (Frequency refers to processor core frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	RSR_LIMIT (R/O) Residency State Regulation Status. When set, frequency is reduced below the operating system request due to residency state regulation limit.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).	
7	VR_TDC (R/O) VR Therm Design Current Status. When set, frequency is reduced below the operating system request due to VR thermal design current limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced below the operating system request due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
12	MAX_TURBO_LIMIT (R/O) Max Turbo Limit Status. When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
13	TURBO_ATTEN (R/O) Turbo Transition Attenuation Status. 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 (R/W) 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 (R/W) 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 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
20	RSR_LIMIT_LOG (R/W) 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	RATL_LOG (R/W) Running average thermal limit Log, RW, When set by PCODE indicates that Running average thermal limit has cause IA frequency clipping. Software should write to this bit to clear the status in this bit.	
22	VR_THERMALERT_LOG (R/W) 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_TDC_LOG (R/W) 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 (R/W) 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	PBM_PL1_LOG (R/W) 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	PBM_PL2_LOG (R/W) 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 (R/W) 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_ATTEN_LOG (R/W) 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.	
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.		Package
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.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W) 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.		Package
14:0	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 (a.k.a TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
15	POWER_LIMIT_1_EN 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	CRITICAL_POWER_CLAMP_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.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
23:17	POWER_LIMIT_1_TIME 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	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	POWER_LIMIT_2_EN 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.	
48	CRITICAL_POWER_CLAMP_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.	
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
MSR_GRAPHICS_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Processor Graphics. (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	INEFFICIENT_OPERATION (R/O) Inefficient Operation Status. When set, processor graphics frequency is operating below target frequency.	
15:13	Reserved.	
16	PROCHOT_LOG (R/W) 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 (R/W) 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	RATL_LOG (R/W) 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_THERMALERT_LOG (R/W) 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_TDC_LOG (R/W) 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.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
24	OTHER_LOG (R/W) 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	PBM_PL1_LOG (R/W) 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	PBM_PL2_LOG (R/W) 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 (R/W) 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.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
MSR_RING_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Ring Interconnect. (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
15:12	Reserved.	
16	PROCHOT_LOG (R/W) 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 (R/W) 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	RATL_LOG (R/W) 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_THERMALERT_LOG (R/W) 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_TDC_LOG (R/W) 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 (R/W) 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	PBM_PL1_LOG (R/W) 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 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
27	PBM_PL2_LOG (R/W) 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.	
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR_CORE_MKTME_ACTIVATE (R/O) MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS].		Core
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
63:36	Reserved.	

The MSRs listed in Table 2-54 are unique to the Intel Core Ultra 7 processor P-core. These MSRs are not supported on the processor E-core.

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter 3 (R/W)		Thread
47:0	FIXED_COUNTER Top-down Microarchitecture Analysis unhalted number of available slots counter.	
63:48	Reserved.	
Register Address: 329H, 809	MSR_PERF_METRICS	
Performance Metrics (R/W) This register provides built-in support for Top-down Micro-architecture Analysis (TMA) metrics. It exposes the four TMA Level 1 metrics where the lower 32 bits are divided into four 8 bit fields, each of which is an integer percentage of the total TOPDOWN.SLOTS (as reported by fixed counter 3).		Thread
7:0	RETIRING Percent of utilized by uops that eventually retire (commit).	
15:8	BAD_SPECULATION Percent of Wasted due to incorrect speculation, covering Utilized by uops that do not retire, or Recovery Bubbles (unutilized slots).	

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
39:32	MULTI_UOPS Frontend bound.	
47:40	BRANCH_MISPREDICTS Frontend bound.	
55:48	FRONTEND_LATENCY Frontend bound.	
63:56	MEMORY_BOUND Frontend bound.	
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core

The MSRs listed in Table 2-48 are unique to the Intel Core Ultra 7 processor E-core. These MSRs are not supported on the processor P-core.

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
SAF Control (W/O) Extension to SAF.		Package
0	INVALIDATE_CURRENT_STRIDE Invalidate all chunks in current stride.	
63:1	Reserved.	
Register Address: D18H—D1FH, 3352—3359	IA32_L2_MASK_[8-15]	
IA32_L2_MASK_[8-15] (R/W) 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."		Module
15:0	WAY_MASK Capacity Bit Mask. Available ways vectors for class of service of IA core. '1 in bit indicates allocation to the way is allowed. '0 indicates allocation to the way is not allowed.	

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
31:16	Reserved.	
Register Address: 1309H—130BH, 4873—4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		Thread
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.	
Register Address: 14C1H—14C8H, 5313 —5320	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Thread
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.	
Register Address: 1A8EH, 6798	MSR_STLB_FILL_TRANSLATION	
STLB Fill Translation (W/O) STLB QoS MSR to fill translations into STLB.		Core
3:0	CLOS Class of service to use for the fill.	
9:4	Reserved.	
10	X Set to 1 when LA is to an executable page.	
11	RW Set to 1 when LA is to a writeable page.	
63:12	LA Logical address to use for fill.	

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 2-56. 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 2-56 and Table 2-57. 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 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Module

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination." See Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 18.17, "Time-Stamp Counter," and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O)	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
0	Lock. (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation. (R/WL)	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	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.	Package
29	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.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W)		Package

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
2:0	Package C-State Limit (R/W) 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. 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. 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.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO registers at MSR_PMG_IO_CAPTURE_BASE[15:0] to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/O) When set, locks bits [15:0] of this register for further writes until the next reset occurs.	
25	Reserved.	
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	Reserved.	
28	C1 State Auto Undemotion Enable (R/W) When set, enables Undemotion from Demoted C1.	
29	PKG C-State Auto Demotion Enable (R/W) When set, enables Package C state demotion.	
63:30	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Capture Base (R/W)		Tile

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
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.	
Register Address: 140H, 320	MISC_FEATURE_ENABLES	
MISC_FEATURE_ENABLES		Thread
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.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
See Table 2-2.		Thread
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
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.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
Performance Monitoring Event Select Register (R/W) See Table 2-2.		Thread
7:0	Event Select.	
15:8	UMask.	
16	USR.	
17	OS.	
18	Edge.	
19	PC.	
20	INT.	
21	AnyThread.	
22	EN.	
23	INV.	

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
31:24	CMASK.	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Package
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Thread
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Module
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Module
0	Thermal Status (R/O)	
1	Thermal Status Log (R/WC0)	
2	PROTCHOT # or FORCEPR# Status (R/O)	
3	PROTCHOT # or FORCEPR# Log (R/WC0)	
4	Critical Temperature Status (R/O)	
5	Critical Temperature Status Log (R/WC0)	
6	Thermal Threshold #1 Status (R/O)	
7	Thermal Threshold #1 Log (R/WC0)	
8	Thermal Threshold #2 Status (R/O)	
9	Thermal Threshold #2 Log (R/WC0)	
10	Power Limitation Status (R/O)	
11	Power Limitation Log (RWC0)	
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.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Thread
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.	
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) See Table 2-3.	
37:35	Reserved.	
38	Turbo Mode Disable (R/W)	
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R)	
29:24	Target Offset (R/W)	
63:30	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher.	Core
1	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher.	Core
63:2	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Shared

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Shared
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode for Groups of Cores (R/W)		Package
0	Reserved.	
7:1	Maximum Number of Cores in Group 0 Number active processor cores which operates under the maximum ratio limit for group 0.	Package
15:8	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.	Package
20:16	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”.	Package
23:21	Group Ratio Delta for Group 1 An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.	Package
28:24	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”.	Package
31:29	Group Ratio Delta for Group 2 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.	Package
36:32	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”.	Package
39:37	Group Ratio Delta for Group 3 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.	Package
44:40	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”.	Package
47:45	Group Ratio Delta for Group 4 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.	Package
52:48	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”.	Package

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
55:53	Group Ratio Delta for Group 5 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.	Package
60:56	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".	Package
63:61	Group Ratio Delta for Group 6 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.	Package
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Thread
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 18.9.2, "Filtering of Last Branch Records."		Thread
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.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
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.		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W)		Thread
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.	

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
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.	
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.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record from Linear IP (R)		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record to Linear IP (R)		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 18.4.1, "IA32_DEBUGCTL MSR."		Package
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2.		Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2.		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2.		Thread

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2.		Thread
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C3 Residency Counter (R/O)	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
63:0	Package C6 Residency Counter (R/O)	Package
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
63:0	Package C7 Residency Counter (R/O)	Package
Register Address: 3FCH, 1020	MSR_MC0_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Module
63:0	Module C0 Residency Counter (R/O)	
Register Address: 3FDH, 1021	MSR_MC6_RESIDENCY	
63:0	Module C6 Residency Counter (R/O)	Module
Register Address: 3FFH, 1023	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Core
63:0	CORE C6 Residency Counter (R/O)	
Register Address: 400H, 1024	IA32_MC0_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 402H, 1026	IA32_MC0_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 412H, 1042	IA32_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 ADDRV 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.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2.		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 15.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	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).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 15.10.1, "RAPL Interfaces."	Package

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
63:20	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C2 Residency Counter (R/O)	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 15.10.3, "Package RAPL Domain."		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 15.10.5, "DRAM RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PP0_POWER_LIMIT	
PP0 RAPL Power Limit Control (R/W) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 15.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O) See Table 2-25.		Package

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0)	
1	Thermal Status (R0)	
5:2	Reserved.	
6	VR Therm Alert Status (R0)	
7	Reserved.	
8	Electrical Design Point Status (R0)	
63:9	Reserved.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread

Table 2-56. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2		Thread

Table 2-57 lists model-specific registers that are supported by the Intel® Xeon Phi™ processor 7215, 7285, 7295 series based on the Knights Mill microarchitecture.

Table 2-57. 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: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
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.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	

Table 2-57. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.		Core
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		Core
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.		Core
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	

Table 2-57. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Scope**
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLS	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.		Core
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.		Core
Register Address: 491H, 1169	IA32_VMX_FMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core

2.19 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-58 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 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Register Address: 0H, 0	IA32_P5_MC_ADDR		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_LINE_SIZE		
See Section 9.10.5, "Monitor/Mwait Address Range Determination."		3, 4, 6	Shared
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER		
Time Stamp Counter See Table 2-2.		0, 1, 2, 3, 4, 6	Unique

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
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.			
Register Address: 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.		0, 1, 2, 3, 4, 6	Shared
Register Address: 1BH, 27	IA32_APIC_BASE		
APIC Location and Status (R/W) See Table 2-2. See Section 11.4.4, "Local APIC Status and Location."		0, 1, 2, 3, 4, 6	Unique
Register Address: 2AH, 42	MSR_EBC_HARD_POWERON		
Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.		0, 1, 2, 3, 4, 6	Shared
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.		
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.		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
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.		
Register Address: 2BH, 43	MSR_EBC_SOFT_POWERON		
Processor Soft Power-On Configuration (R/W) Enables and disables processor features.		0, 1, 2, 3, 4, 6	Shared
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.		
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.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
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.		2,3, 4, 6	Shared
15:0	Reserved.		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
18:16	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: Encoding Scalable Bus Speed 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.		
31:24	Core Clock Frequency to System Bus Frequency Ratio (R) The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.		
63:25	Reserved.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
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.		0, 1	Shared
20:0	Reserved.		
23:21	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: Encoding Scalable Bus Speed 000B 100 MHz All others values reserved.		
63:24	Reserved.		
Register Address: 3AH, 58	IA32_FEATURE_CONTROL		
Control Features in IA-32 Processor (R/W) See Table 2-2. (If CPUID.01H:ECX.[bit 5])		3, 4, 6	Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
BIOS Update Trigger Register (W) See Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID		
BIOS Update Signature ID (R/W) See Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL		
SMM Monitor Configuration (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: FEH, 254	IA32_MTRRCAP		
MTRR Information See Section 12.11.1, "MTRR Feature Identification."		0, 1, 2, 3, 4, 6	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS		
CS Register Target for CPL 0 Code (R/W) See Table 2-2 and Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP		
Stack Pointer for CPL 0 Stack (R/W) See Table 2-2 and Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP		
CPL 0 Code Entry Point (R/W) See Table 2-2 and Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 179H, 377	IA32_MCG_CAP		
Machine Check Capabilities (R) See Table 2-2 and Section 16.3.1.1, "IA32_MCG_CAP MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17AH, 378	IA32_MCG_STATUS		
Machine Check Status (R) See Table 2-2 and Section 16.3.1.2, "IA32_MCG_STATUS MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17BH, 379	IA32_MCG_CTL		
Machine Check Feature Enable (R/W) See Table 2-2 and Section 16.3.1.3, "IA32_MCG_CTL MSR."			
Register Address: 180H, 384	MSR_MCG_RAX		
Machine Check EAX/RAX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 181H, 385	MSR_MCG_RBX		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Machine Check EBX/RBX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 182H, 386	MSR_MCG_RCX		
Machine Check ECX/RCX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 183H, 387	MSR_MCG_RDX		
Machine Check EDX/RDX Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 184H, 388	MSR_MCG_RSI		
Machine Check ESI/RSI Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 185H, 389	MSR_MCG_RDI		
Machine Check EDI/RDI Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 186H, 390	MSR_MCG_RBP		
Machine Check EBP/RBP Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 187H, 391	MSR_MCG_RSP		
Machine Check ESP/RSP Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 188H, 392	MSR_MCG_RFLAGS		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Machine Check EFLAGS/RFLAG Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 189H, 393	MSR_MCG_RIP		
Machine Check EIP/RIP Save State See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 18AH, 394	MSR_MCG_MISC		
Machine Check Miscellaneous See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 18BH—18FH, 395—399	MSR_MCG_RESERVED1—MSR_MCG_RESERVED5		
Reserved.			
Register Address: 190H, 400	MSR_MCG_R8		
Machine Check R8 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 191H, 401	MSR_MCG_R9		
Machine Check R9D/R9 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 192H, 402	MSR_MCG_R10		
Machine Check R10 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
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.		
Register Address: 193H, 403	MSR_MCG_R11		
Machine Check R11 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 194H, 404	MSR_MCG_R12		
Machine Check R12 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 195H, 405	MSR_MCG_R13		
Machine Check R13 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 196H, 406	MSR_MCG_R14		
Machine Check R14 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 197H, 407	MSR_MCG_R15		
Machine Check R15 See Section 16.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
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.		
Register Address: 198H, 408	IA32_PERF_STATUS		
See Table 2-2. See Section 15.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique
Register Address: 199H, 409	IA32_PERF_CTL		
See Table 2-2. See Section 15.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Register Address: 19AH, 410	IA32_CLOCK_MODULATION		
Thermal Monitor Control (R/W) See Table 2-2 and Section 15.8.3, "Software Controlled Clock Modulation."		0, 1, 2, 3, 4, 6	Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT		
Thermal Interrupt Control (R/W) See Section 15.8.2, "Thermal Monitor," and Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 19CH, 412	IA32_THERM_STATUS		
Thermal Monitor Status (R/W) See Section 15.8.2, "Thermal Monitor," and Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 19DH, 413	MSR_THERM2_CTL		
Thermal Monitor 2 Control			
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.		3	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.		4, 6	Shared
Register Address: 1A0H, 416	IA32_MISC_ENABLE		
Enable Miscellaneous Processor Features (R/W)		0, 1, 2, 3, 4, 6	Shared
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.		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
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. 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	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.	3	
17:14	Reserved.		
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	3, 4, 6	

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
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	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.	3, 4, 6	
23	xTPR Message Disable (R/W) See Table 2-2.		Shared
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	XD Bit Disable (R/W) See Table 2-3.		Unique
63:35	Reserved.		
Register Address: 1A1H, 417	MSR_PLATFORM_BRV		
Platform Feature Requirements (R)		3, 4, 6	Shared
17:0	Reserved.		
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.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
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."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		
63:0	From Linear IP Linear address of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D8H, 472	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 18.13.3, "Last Exception Records."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the target of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D8H, 472	MSR_LER_TO_LIP		
63:0	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D9H, 473	MSR_DEBUGCTLA		
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."		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DAH, 474	MSR_LASTBRANCH_TOS		
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.		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DBH, 475	MSR_LASTBRANCH_0		
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."		0, 1, 2	Unique

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Register Address: 1DCH, 476	MSR_LASTBRANCH_1		
Last Branch Record 1 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DDH, 477	MSR_LASTBRANCH_2		
Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DEH, 478	MSR_LASTBRANCH_3		
Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0		
Variable Range Base MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs".		0, 1, 2, 3, 4, 6	Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7		
Variable Range Mask MTRR See Section 12.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs".		0, 1, 2, 3, 4, 6	Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs".		0, 1, 2, 3, 4, 6	Shared

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000		
Fixed Range MTRR See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 277H, 631	IA32_PAT		
Page Attribute Table See Section 12.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE		
Default Memory Types (R/W) See Table 2-2 and Section 12.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		0, 1, 2, 3, 4, 6	Shared
Register Address: 300H, 768	MSR_BPU_COUNTER0		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 301H, 769	MSR_BPU_COUNTER1		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 302H, 770	MSR_BPU_COUNTER2		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 303H, 771	MSR_BPU_COUNTER3		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 304H, 772	MSR_MS_COUNTER0		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 305H, 773	MSR_MS_COUNTER1		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 306H, 774	MSR_MS_COUNTER2		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 307H, 775	MSR_MS_COUNTER3		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 308H, 776	MSR_FLAME_COUNTER0		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 309H, 777	MSR_FLAME_COUNTER1		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30AH, 778	MSR_FLAME_COUNTER2		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30BH, 779	MSR_FLAME_COUNTER3		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30CH, 780	MSR_IQ_COUNTER0		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30DH, 781	MSR_IQ_COUNTER1		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30EH, 782	MSR_IQ_COUNTER2		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30FH, 783	MSR_IQ_COUNTER3		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 310H, 784	MSR_IQ_COUNTER4		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 311H, 785	MSR_IQ_COUNTER5		
See Section 20.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 360H, 864	MSR_BPU_CCCR0		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 361H, 865	MSR_BPU_CCCR1		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 362H, 866	MSR_BPU_CCCR2		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 363H, 867	MSR_BPU_CCCR3		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 364H, 868	MSR_MS_CCCR0		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 365H, 869	MSR_MS_CCCR1		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 366H, 870	MSR_MS_CCCR2		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 367H, 871	MSR_MS_CCCR3		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 368H, 872	MSR_FLAME_CCCR0		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique[1]
Register Address: 369H, 873	MSR_FLAME_CCCR1		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36AH, 874	MSR_FLAME_CCCR2		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36BH, 875	MSR_FLAME_CCCR3		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36CH, 876	MSR_IQ_CCCR0		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36DH, 877	MSR_IQ_CCCR1		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36EH, 878	MSR_IQ_CCCR2		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36FH, 879	MSR_IQ_CCCR3		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 370H, 880	MSR_IQ_CCCR4		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 371H, 881	MSR_IQ_CCCR5		
See Section 20.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A0H, 928	MSR_BSU_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A1H, 929	MSR_BSU_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A2H, 930	MSR_FSB_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A3H, 931	MSR_FSB_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A4H, 932	MSR_FIRM_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A5H, 933	MSR_FIRM_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A6H, 934	MSR_FLAME_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A7H, 935	MSR_FLAME_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A8H, 936	MSR_DAC_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A9H, 937	MSR_DAC_ESCR1		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AAH, 938	MSR_MOB_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ABH, 939	MSR_MOB_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ACH, 940	MSR_PMH_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ADH, 941	MSR_PMH_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AEH, 942	MSR_SAAT_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AFH, 943	MSR_SAAT_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B0H, 944	MSR_U2L_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B1H, 945	MSR_U2L_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B2H, 946	MSR_BPU_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B3H, 947	MSR_BPU_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B4H, 948	MSR_IS_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B5H, 949	MSR_IS_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B6H, 950	MSR_ITLB_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B7H, 951	MSR_ITLB_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B8H, 952	MSR_CRU_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B9H, 953	MSR_CRU_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BAH, 954	MSR_IQ_ESCR0		
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.		0, 1, 2	Shared

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Register Address: 3BBH, 955	MSR_IQ_ESCR1		
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.		0, 1, 2	Shared
Register Address: 3BCH, 956	MSR_RAT_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BDH, 957	MSR_RAT_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BEH, 958	MSR_SSU_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C0H, 960	MSR_MS_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C1H, 961	MSR_MS_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C2H, 962	MSR_TBPU_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C3H, 963	MSR_TBPU_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C4H, 964	MSR_TC_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C5H, 965	MSR_TC_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C8H, 968	MSR_IX_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C9H, 969	MSR_IX_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CAH, 970	MSR_ALF_ESCR0		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CBH, 971	MSR_ALF_ESCR1		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CCH, 972	MSR_CRU_ESCR2		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CDH, 973	MSR_CRU_ESCR3		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E0H, 992	MSR_CRU_ESCR4		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E1H, 993	MSR_CRU_ESCR5		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F0H, 1008	MSR_TC_PRECISE_EVENT		
See Section 20.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)		
Processor Event Based Sampling (PEBS) (R/W) Controls the enabling of processor event sampling and replay tagging.		0, 1, 2, 3, 4, 6	Shared
12:0	See https://perfmon-events.intel.com/.		
23:13	Reserved.		
24	UOP Tag Enables replay tagging when set.		
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.		
Register Address: 3F2H, 1010	MSR_PEBS_MATRIX_VERT		
See https://perfmon-events.intel.com/.		0, 1, 2, 3, 4, 6	Shared
Register Address: 400H, 1024	IA32_MC0_CTL		
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 401H, 1025	IA32_MC0_STATUS		
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		0, 1, 2, 3, 4, 6	Shared
Register Address: 402H, 1026	IA32_MC0_ADDR		
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 403H, 1027	IA32_MC0_MISC		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
See Section 16.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC0_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 404H, 1028	IA32_MC1_CTL		
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 405H, 1029	IA32_MC1_STATUS		
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		0, 1, 2, 3, 4, 6	Shared
Register Address: 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 ADDRV 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.		0, 1, 2, 3, 4, 6	Shared
Register Address: 407H, 1031	IA32_MC1_MISC		
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 MISCV 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.			Shared
Register Address: 408H, 1032	IA32_MC2_CTL		
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 409H, 1033	IA32_MC2_STATUS		
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		0, 1, 2, 3, 4, 6	Shared
Register Address: 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 ADDRV 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.			
Register Address: 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 MISCV 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.			
Register Address: 40CH, 1036	IA32_MC3_CTL		
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS		
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40EH, 1038	IA32_MC3_ADDR		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
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 ADDRV 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.		0, 1, 2, 3, 4, 6	Shared
Register Address: 40FH, 1039	IA32_MC3_MISC		
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.		0, 1, 2, 3, 4, 6	Shared
Register Address: 410H, 1040	IA32_MC4_CTL		
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 411H, 1041	IA32_MC4_STATUS		
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		0, 1, 2, 3, 4, 6	Shared
Register Address: 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 ADDRV 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.			
Register Address: 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.			
Register Address: 480H, 1152	IA32_VMX_BASIC		
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		3, 4, 6	Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS		
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		3, 4, 6	Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS		
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS		
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 485H, 1157	IA32_VMX_MISC		
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and Table 2-2.		3, 4, 6	Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0		
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1		
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 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," and Table 2-2.		3, 4, 6	Unique
Register Address: 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," and Table 2-2.		3, 4, 6	Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM		
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration," and Table 2-2.		3, 4, 6	Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2		
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 600H, 1536	IA32_DS_AREA		
DS Save Area (R/W) See Table 2-2 and Section 20.6.3.4, "Debug Store (DS) Mechanism."		0, 1, 2, 3, 4, 6	Unique
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP		
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. 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."		3, 4, 6	Unique
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP		
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique[1]
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP		
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."		3, 4, 6	Unique
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP		
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP		

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	**Bit Description**	**Model Availability**	**Shared/ Unique[1]**
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: C000_0080H	IA32_EFER		
Extended Feature Enables See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0081H	IA32_STAR		
System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0082H	IA32_LSTAR		
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0084H	IA32_FMASK		
System Call Flag Mask (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0100H	IA32_FS_BASE		
Map of BASE Address of FS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0101H	IA32_GS_BASE		
Map of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE		
Swap Target of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique

Table 2-58. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique[1]
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 2-59 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 2-59. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/ Unique
Register Address: 107CCH	MSR_IFSB_BUSQ0		
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."		3, 4	Shared
Register Address: 107CDH	MSR_IFSB_BUSQ1		
IFSB BUSQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107CEH	MSR_IFSB_SNPQ0		
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."		3, 4	Shared
Register Address: 107CFH	MSR_IFSB_SNPQ1		
IFSB SNPQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D0H	MSR_EFSB_DRDY0		
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."		3, 4	Shared
Register Address: 107D1H	MSR_EFSB_DRDY1		
EFSB DRDY Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D2H	MSR_IFSB_CTL6		
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."		3, 4	Shared
Register Address: 107D3H	MSR_IFSB_CNTR7		
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."		3, 4	Shared

The MSRs listed in Table 2-60 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 2-60 are shared between logical processors in the same core, but are replicated for each core.

Table 2-60. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/ Unique
Register Address: 107CCH	MSR_EMON_L3_CTR_CTL0		
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."		6	Shared
Register Address: 107CDH	MSR_EMON_L3_CTR_CTL1		
GBUSQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107CEH	MSR_EMON_L3_CTR_CTL2		
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."		6	Shared
Register Address: 107CFH	MSR_EMON_L3_CTR_CTL3		
GSNPQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D0H	MSR_EMON_L3_CTR_CTL4		
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."		6	Shared
Register Address: 107D1H	MSR_EMON_L3_CTR_CTL5		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D2H	MSR_EMON_L3_CTR_CTL6		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D3H	MSR_EMON_L3_CTR_CTL7		
FSB Event Control and Counter Register (R/W)		6	Shared

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 2-61. 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 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 0H, 0	P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique
Register Address: 1H, 1	P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 9.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 18.17, "Time-Stamp Counter," and Table 2-2.		Unique
Register Address: 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.		Shared
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 11.4.4, "Local APIC Status and Location," and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
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.	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
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)	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in IA-32 Processor (R/W) See Table 2-2.		Unique
Register Address: 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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1	
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2	
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3	
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4	
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5	
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6	
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the scalable bus clock speed.		Shared
2:0	▪ 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 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Unique
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Shared
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
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.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
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.	
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	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
See Table 2-2.		Shared
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2 and Section 15.8.2, "Thermal Monitor."		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2 and Section 15.8.2, "Thermal Monitor".		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
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.	
Register Address: 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	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
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.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
12	Reserved.	
13	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.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
22	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.	Shared
33:23	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Shared
63:35	Reserved.	
Register Address: 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 MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in Table 2-2.		Unique
Register Address: 1DDH, 477	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.		Unique
Register Address: 1DEH, 478	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.		Unique
Register Address: 200H, 512	MTRRphysBase0	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Memory Type Range Registers		Unique
Register Address: 201H, 513	MTRRphysMask0	
Memory Type Range Registers		Unique
Register Address: 202H, 514	MTRRphysBase1	
Memory Type Range Registers		Unique
Register Address: 203H, 515	MTRRphysMask1	
Memory Type Range Registers		Unique
Register Address: 204H, 516	MTRRphysBase2	
Memory Type Range Registers		Unique
Register Address: 205H, 517	MTRRphysMask2	
Memory Type Range Registers		Unique
Register Address: 206H, 518	MTRRphysBase3	
Memory Type Range Registers		Unique
Register Address: 207H, 519	MTRRphysMask3	
Memory Type Range Registers		Unique
Register Address: 208H, 520	MTRRphysBase4	
Memory Type Range Registers		Unique
Register Address: 209H, 521	MTRRphysMask4	
Memory Type Range Registers		Unique
Register Address: 20AH, 522	MTRRphysBase5	
Memory Type Range Registers		Unique
Register Address: 20BH, 523	MTRRphysMask5	
Memory Type Range Registers		Unique
Register Address: 20CH, 524	MTRRphysBase6	
Memory Type Range Registers		Unique
Register Address: 20DH, 525	MTRRphysMask6	
Memory Type Range Registers		Unique
Register Address: 20EH, 526	MTRRphysBase7	
Memory Type Range Registers		Unique
Register Address: 20FH, 527	MTRRphysMask7	
Memory Type Range Registers		Unique
Register Address: 250H, 592	MTRRfix64K_00000	
Memory Type Range Registers		Unique
Register Address: 258H, 600	MTRRfix16K_80000	
Memory Type Range Registers		Unique
Register Address: 259H, 601	MTRRfix16K_A0000	
Memory Type Range Registers		Unique

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
Register Address: 268H, 616	MTRRfix4K_C0000	
Memory Type Range Registers		Unique
Register Address: 269H, 617	MTRRfix4K_C8000	
Memory Type Range Registers		Unique
Register Address: 26AH, 618	MTRRfix4K_D0000	
Memory Type Range Registers		Unique
Register Address: 26BH, 619	MTRRfix4K_D8000	
Memory Type Range Registers		Unique
Register Address: 26CH, 620	MTRRfix4K_E0000	
Memory Type Range Registers		Unique
Register Address: 26DH, 621	MTRRfix4K_E8000	
Memory Type Range Registers		Unique
Register Address: 26EH, 622	MTRRfix4K_F0000	
Memory Type Range Registers		Unique
Register Address: 26FH, 623	MTRRfix4K_F8000	
Memory Type Range Registers		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2 and Section 12.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		Unique
Register Address: 400H, 1024	IA32_MC0_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 402H, 1026	IA32_MC0_ADDR	
See Section 16.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 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 ADDRV 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.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 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 ADDRV 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.		Unique
Register Address: 40CH, 1036	MSR_MC4_CTL	
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 40DH, 1037	MSR_MC4_STATUS	
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 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 ADDRV 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.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
IA32_MC3_CTL	See Section 16.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 411H, 1041	IA32_MC3_STATUS	
IA32_MC3_STATUS	See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."	
Register Address: 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 ADDRV 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.		Unique
Register Address: 413H, 1043	MSR_MC3_MISC	
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.		Unique
Register Address: 414H, 1044	MSR_MC5_CTL	
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	MSR_MC5_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.		Unique
Register Address: 416H, 1046	MSR_MC5_ADDR	
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.		Unique
Register Address: 417H, 1047	MSR_MC5_MISC	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ 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.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information." (If CPUID.01H:ECX.[bit 5])		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX.[bit 5])		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
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])		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls." (If CPUID.01H:ECX.[bit 5])		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls." (If CPUID.01H:ECX.[bit 5])		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data." (If CPUID.01H:ECX.[bit 5])		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX.[bit 5])		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
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])		Unique
Register Address: 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.[bit 5])		Unique
Register Address: 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.[bit 5])		Unique
Register Address: 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.[bit 5])		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
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_CTLS[bit 63])		Unique
Register Address: 600H, 1536	IA32_DS_AREA	

Table 2-61. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	**Bit Description**	**Shared/ Unique**
DS Save Area (R/W) See Table 2-2 and Section 20.6.3.4, "Debug Store (DS) Mechanism."		Unique
31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.	
63:32	Reserved.	
Register Address: C000_0080H	IA32_EFER	
See Table 2-2.		Unique
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 2-62. MSRs in Pentium M Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER
See Section 18.17, "Time-Stamp Counter," and see Table 2-2.	
Register Address: 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.	
Register Address: 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.

Table 2-62. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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.
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)
Register Address: 40H, 64	MSR_LASTBRANCH_0

Table 2-62. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 41H, 65	MSR_LASTBRANCH_1
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 42H, 66	MSR_LASTBRANCH_2
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 43H, 67	MSR_LASTBRANCH_3
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 44H, 68	MSR_LASTBRANCH_4
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 45H, 69	MSR_LASTBRANCH_5
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 46H, 70	MSR_LASTBRANCH_6
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 47H, 71	MSR_LASTBRANCH_7
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 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.
Register Address: 11EH, 281	MSR_BBL_CR_CTL3
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.
4:1	Reserved.

Table 2-62. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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.
Register Address: 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.
Register Address: 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.
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.
Register Address: 198H, 408	IA32_PERF_STATUS

Table 2-62. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
See Table 2-2.	
Register Address: 199H, 409	IA32_PERF_CTL
See Table 2-2.	
Register Address: 19AH, 410	IA32_CLOCK_MODULATION
Clock Modulation (R/W). See Table 2-2 and Section 15.8.3, "Software Controlled Clock Modulation."	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT
Thermal Interrupt Control (R/W) See Table 2-2 and Section 15.8.2, "Thermal Monitor."	
Register Address: 19CH, 412	IA32_THERM_STATUS
Thermal Monitor Status (R/W) See Table 2-2 and Section 15.8.2, "Thermal Monitor."	
Register Address: 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.
Register Address: 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	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.

Table 2-62. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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.
Register Address: 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: ▪ MSR_LASTBRANCH_0_FROM_IP (at 40H). ▪ Section 18.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 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)."	
Register Address: 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."	
Register Address: 1DEH, 478	MSR_LER_FROM_LIP

Table 2-62. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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."	
Register Address: 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."	
Register Address: 400H, 1024	IA32_MC0_CTL
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 401H, 1025	IA32_MC0_STATUS
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."	
Register Address: 402H, 1026	IA32_MC0_ADDR
See Section 14.3.2.3., "IA32_MCi_ADDR MSRs". The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 404H, 1028	IA32_MC1_CTL
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 405H, 1029	IA32_MC1_STATUS
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."	
Register Address: 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 ADDRV 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.	
Register Address: 408H, 1032	IA32_MC2_CTL
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 409H, 1033	IA32_MC2_STATUS
See Chapter 16.3.2.2, "IA32_MCi_STATUS MSRS."	
Register Address: 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 ADDRV 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.	
Register Address: 40CH, 1036	MSR_MC4_CTL
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 40DH, 1037	MSR_MC4_STATUS
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."	
Register Address: 40EH, 1038	MSR_MC4_ADDR

Table 2-62. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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 ADDRV 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.	
Register Address: 410H, 1040	MSR_MC3_CTL
See Section 16.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 411H, 1041	MSR_MC3_STATUS
See Section 16.3.2.2, "IA32_MCi_STATUS MSRS."	
Register Address: 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 ADDRV 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.	
Register Address: 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 2-63. MSRs in the P6 Family Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	TSC
See Section 18.17, "Time-Stamp Counter."	
Register Address: 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.

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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.
Register Address: 1BH, 27	APIC_BASE
Section 11.4.4, "Local APIC Status and Location."	
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.
Register Address: 2AH, 42	EBL_CR_POWERON
Processor Hard Power-On Configuration (R/W) Enables and disables processor features, and (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.
4	BERR# Enable for Initiator Bus Requests (R/W) 1 = Enabled. 0 = Disabled.

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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)
27	Clock Frequency Ratio
63:28	Reserved.[1]
Register Address: 33H, 51	MSR_TEST_CTRL

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
Test Control Register	
29:0	Reserved.
30	Streaming Buffer Disable
31	Disable LOCK# Assertion for split locked access.
Register Address: 79H, 121	BIOS_UPDT_TRIG
BIOS Update Trigger Register.	
Register Address: 88H, 136	BBL_CR_D0[63:0]
Chunk 0 data register D[63:0]: used to write to and read from the L2.	
Register Address: 89H, 137	BBL_CR_D1
Chunk 1 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8AH, 138	BBL_CR_D2
Chunk 2 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8BH, 139	BIOS_SIGN/BBL_CR_D3
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.	
Register Address: C1H, 193	PerfCtr0 (PERFCTR0)
Performance Counter Register See Table 2-2.	
Register Address: C2H, 194	PerfCtr1 (PERFCTR1)
Performance Counter Register See Table 2-2.	
Register Address: FEH, 254	MTRRcap
Memory Type Range Registers	
Register Address: 116H, 278	BBL_CR_ADDR
Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.	
2:0	Reserved; set to 0.
31:3	Address bits [35:3].
63:32	Reserved.
Register Address: 118H, 280	BBL_CR_DECC
Data ECC register D[7:0]: used to write ECC and read ECC to/from L2.	
Register Address: 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.	

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
4:0	L2 Command: 01100 = Data Read w/ LRU update (RLU). 01110 = Tag Read w/ Data Read (TRR). 01111 = Tag Inquire (TI). 00010 = L2 Control Register Read (CR). 00011 = L2 Control Register Write (CW). 010 + MESI encode = Tag Write w/ Data Read (TWR). 111 + MESI encode = Tag Write w/ Data Write (TWW). 100 + MESI encode = Tag Write (TW).
6:5	
7	State to L2
9:8	Reserved.
11:10	Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11 Way to L2
13:12	Modified - 11,Exclusive - 10, Shared - 01, Invalid - 00 Way from L2
15:14	State from L2.
16	Reserved.
17	L2 Hit.
18	Reserved.
20:19	User supplied ECC.
21	Processor number: [2] Disable = 1. Enable = 0. Reserved.
63:22	Reserved.
Register Address: 11AH, 282	BBL_CR_TRIG
Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.	
Register Address: 11BH, 283	BBL_CR_BUSY
Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY.	
Register Address: 11EH, 286	BBL_CR_CTL3
Control register 3: used to configure the L2 Cache.	
0	L2 Configured (read/write).
4:1	L2 Cache Latency (read/write).
5	ECC Check Enable (read/write).
6	Address Parity Check Enable (read/write).
7	CRTN Parity Check Enable (read/write).
8	L2 Enabled (read/write).

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
10:9	L2 Associativity (read only): 00 = Direct Mapped. 01 = 2 Way. 10 = 4 Way. 11 = Reserved.
12:11	Number of L2 banks (read only).
17:13	Cache size per bank (read/write): 00001 = 256 KBytes. 00010 = 512 KBytes. 00100 = 1 MByte. 01000 = 2 MBytes. 10000 = 4 MBytes.
18	Cache State error checking enable (read/write).
19	Reserved.
22:20	L2 Physical Address Range support: 111 = 64 GBytes. 110 = 32 GBytes. 101 = 16 GBytes. 100 = 8 GBytes. 011 = 4 GBytes. 010 = 2 GBytes. 001 = 1 GByte. 000 = 512 MBytes.
23	L2 Hardware Disable (read only).
24	Reserved.
25	Cache bus fraction (read only).
63:26	Reserved.
Register Address: 174H, 372	SYSENTER_CS_MSR
CS register target for CPL 0 code	
Register Address: 175H, 373	SYSENTER_ESP_MSR
Stack pointer for CPL 0 stack	
Register Address: 176H, 374	SYSENTER_EIP_MSR
CPL 0 code entry point	
Register Address: 179H, 377	MCG_CAP
Machine Check Global Control Register	
Register Address: 17AH, 378	MCG_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.	
Register Address: 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).	
Register Address: 186H, 390	PerfEvtSel0 (EVNTSEL0)

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
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.
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BP0 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)
Register Address: 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.

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BP0 pin.
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)
Register Address: 1D9H, 473	DEBUGCTLMSR
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.
Register Address: 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.	
Register Address: 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.	
Register Address: 1DDH, 477	LASTINTFROMIP
Last INT from IP	
Register Address: 1DEH, 478	LASTINTTOIP
Last INT to IP	
Register Address: 200H, 512	MTRRphysBase0
Memory Type Range Registers	
Register Address: 201H, 513	MTRRphysMask0
Memory Type Range Registers	

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
Register Address: 202H, 514	MTRRphysBase1
Memory Type Range Registers	
Register Address: 203H, 515	MTRRphysMask1
Memory Type Range Registers	
Register Address: 204H, 516	MTRRphysBase2
Memory Type Range Registers	
Register Address: 205H, 517	MTRRphysMask2
Memory Type Range Registers	
Register Address: 206H, 518	MTRRphysBase3
Memory Type Range Registers	
Register Address: 207H, 519	MTRRphysMask3
Memory Type Range Registers	
Register Address: 208H, 520	MTRRphysBase4
Memory Type Range Registers	
Register Address: 209H, 521	MTRRphysMask4
Memory Type Range Registers	
Register Address: 20AH, 522	MTRRphysBase5
Memory Type Range Registers	
Register Address: 20BH, 523	MTRRphysMask5
Memory Type Range Registers	
Register Address: 20CH, 524	MTRRphysBase6
Memory Type Range Registers	
Register Address: 20DH, 525	MTRRphysMask6
Memory Type Range Registers	
Register Address: 20EH, 526	MTRRphysBase7
Memory Type Range Registers	
Register Address: 20FH, 527	MTRRphysMask7
Memory Type Range Registers	
Register Address: 250H, 592	MTRRfix64K_00000
Memory Type Range Registers	
Register Address: 258H, 600	MTRRfix16K_80000
Memory Type Range Registers	
Register Address: 259H, 601	MTRRfix16K_A0000
Memory Type Range Registers	
Register Address: 268H, 616	MTRRfix4K_C0000
Memory Type Range Registers	
Register Address: 269H, 617	MTRRfix4K_C8000

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
Memory Type Range Registers	
Register Address: 26AH, 618	MTRRfix4K_D0000
Memory Type Range Registers	
Register Address: 26BH, 619	MTRRfix4K_D8000
Memory Type Range Registers	
Register Address: 26CH, 620	MTRRfix4K_E0000
Memory Type Range Registers	
Register Address: 26DH, 621	MTRRfix4K_E8000
Memory Type Range Registers	
Register Address: 26EH, 622	MTRRfix4K_F0000
Memory Type Range Registers	
Register Address: 26FH, 623	MTRRfix4K_F8000
Memory Type Range Registers	
Register Address: 2FFH, 767	MTRRdefType
Memory Type Range Registers	
2:0	Default memory type
10	Fixed MTRR enable
11	MTRR Enable
Register Address: 400H, 1024	MC0_CTL
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).	
Register Address: 401H, 1025	MC0_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 MC0_STATUS only, this bit is hardcoded to 1.)
61	MC_STATUS_UC
62	MC_STATUS_0
63	MC_STATUS_V
Register Address: 402H, 1026	MC0_ADDR
Register Address: 403H, 1027	MC0_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 404H, 1028	MC1_CTL

Table 2-63. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	**Bit Description**
Register Address: 405H, 1029	MC1_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 406H, 1030	MC1_ADDR
Register Address: 407H, 1031	MC1_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 408H, 1032	MC2_CTL
Register Address: 409H, 1033	MC2_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 40AH, 1034	MC2_ADDR
Register Address: 40BH, 1035	MC2_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 40CH, 1036	MC4_CTL
Register Address: 40DH, 1037	MC4_STATUS
Bit definitions same as MC0_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.	
Register Address: 40EH, 1038	MC4_ADDR
Defined in MCA architecture but not implemented in P6 Family processors.	
Register Address: 40FH, 1039	MC4_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 410H, 1040	MC3_CTL
Register Address: 411H, 1041	MC3_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 412H, 1042	MC3_ADDR
Register Address: 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 2-64. MSRs in the Pentium Processor

Register Address: Hex, Decimal Register Information	Register Name
Register Address: 0H, 0	P5_MC_ADDR
See Section 16.10.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 16.10.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 10H, 16	TSC
See Section 18.17, "Time-Stamp Counter."	
Register Address: 11H, 17	CESR
See Section 20.6.9.1, "Control and Event Select Register (CESR)."	
Register Address: 12H, 18	CTR0
Section 20.6.9.3, "Events Counted."	
Register Address: 13H, 19	CTR1
Section 20.6.9.3, "Events Counted."	