



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

Documentation Changes

April 2022

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: 252046-069



Notices & Disclaimers

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

No product or component can be absolutely secure.

Your costs and results may vary.

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

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

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

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

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

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

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



Contents

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



Revision History

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



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



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

§

Preface

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

Affected Documents

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

Nomenclature

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

Summary Tables of Changes

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

Codes Used in Summary Tables

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

Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 11, Volume 1
2	Updates to Chapter 15, Volume 1
3	Updates to Chapter 18, Volume 1
4	Updates to Chapter 2, Volume 2A
5	Updates to Chapter 3, Volume 2A
6	Updates to Chapter 4, Volume 2B
7	Updates to Chapter 5, Volume 2C
8	Updates to Chapter 4, Volume 3A
9	Updates to Chapter 14, Volume 3B
10	Updates to Chapter 17, Volume 3B
11	Updates to Chapter 19, Volume 3B
12	Updates to Chapter 24, Volume 3C
13	Updates to Chapter 25, Volume 3C
14	Updates to Chapter 27, Volume 3C
15	Updates to Chapter 34, Volume 3D
16	Updates to Appendix B, Volume 3D
17	Updates to Appendix C, Volume 3D
18	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 11, Volume 1

Change bars and green 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: Update to section 11.5.2.6, "Inexact-Result (Precision) Exception (#P)", to correct the PE flag bit number from 2 to 5.

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 SSE2 EXTENSIONS

SSE2 extensions use the single instruction multiple data (SIMD) execution model that is used with MMX technology and SSE extensions. 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`, SSE2 extensions are present.

SSE2 extensions add 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 the SSE extensions
 - Additional cacheability-control and instruction-ordering instructions
- Modifications to existing IA-32 instructions to support 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 SSE/SSE2 extensions, programmers can develop algorithms that finely mix packed single- and double-precision floating-point data and 64- and 128-bit packed integer data.
- SSE2 extensions enhance the support introduced with SSE extensions for controlling the cacheability of SIMD data. 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.

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

SSE2 extensions are accessible from all IA-32 execution modes: protected mode, real address mode, virtual 8086 mode.

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

For additional information about SSE2 extensions, see:

- *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A & 2B*, provide a detailed description of individual SSE3 instructions.
- Chapter 13, “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 the SSE and SSE2 extensions into an operating-system environment.

11.2 SSE2 PROGRAMMING ENVIRONMENT

Figure 11-1 shows the programming environment for SSE2 extensions. No new registers or other instruction execution state are defined with SSE2 extensions. SSE2 instructions use the XMM registers, the 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.

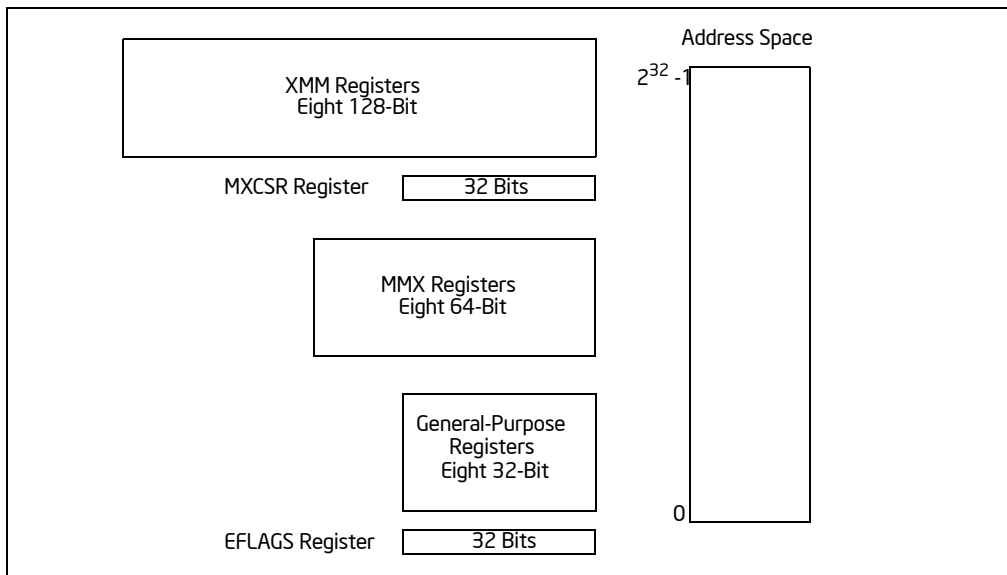


Figure 11-1. Streaming SIMD Extensions 2 Execution Environment

- **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 SSE2 in 64-Bit Mode and Compatibility Mode

In compatibility mode, SSE2 extensions function like they do 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 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 SSE2 Extensions with SSE, MMX Technology and x87 FPU Programming Environment

SSE2 extensions do not introduce any new state to the IA-32 execution environment beyond that of SSE. SSE2 extensions represent an enhancement of SSE extensions; they are fully compatible and share the same state information. 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 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 SSE/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 the SSE2 extensions. See Section 10.2.3.4, “Denormals-Are-Zeros,” for a description of this flag.

11.3 SSE2 DATA TYPES

SSE2 extensions 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.)

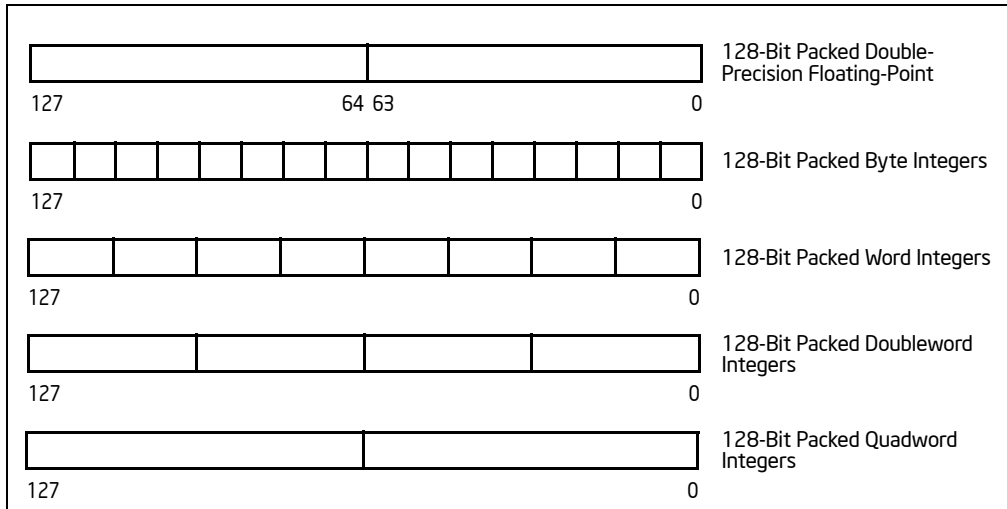


Figure 11-2. Data Types Introduced with the SSE2 Extensions

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 which 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 SSE2 INSTRUCTIONS

The 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 the SSE extensions
- 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.

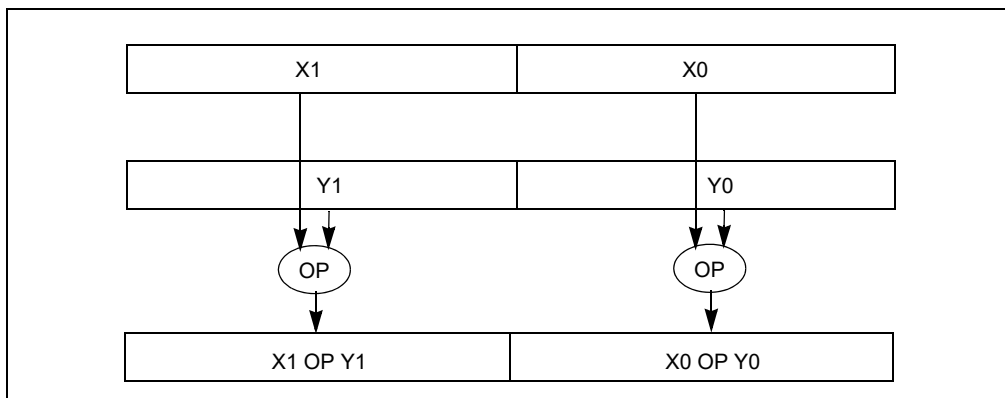


Figure 11-3. Packed Double-Precision Floating-Point Operations

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 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.

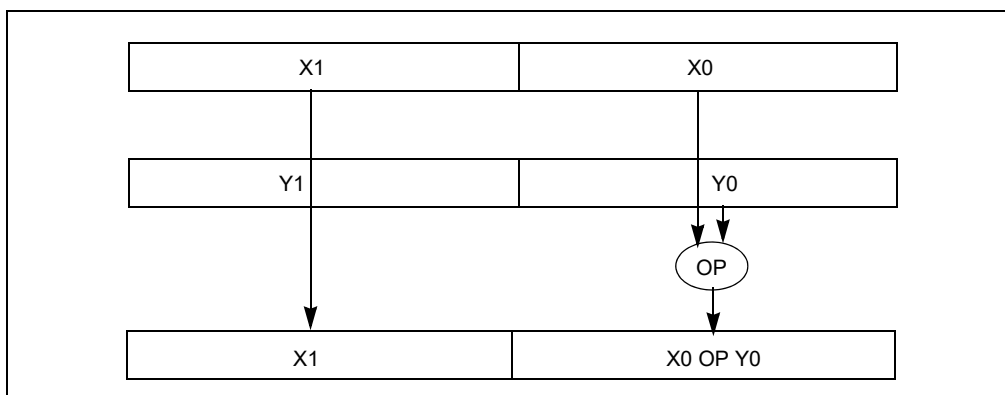


Figure 11-4. Scalar Double-Precision Floating-Point Operations

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 SSE2 Arithmetic Instructions

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 SSE2 Logical Instructions

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 SSE2 Comparison Instructions

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 SSE2 Shuffle and Unpack Instructions

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

The `SHUFPS` (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 `SHUFPS` instruction can swap two packed double-precision floating-point values.

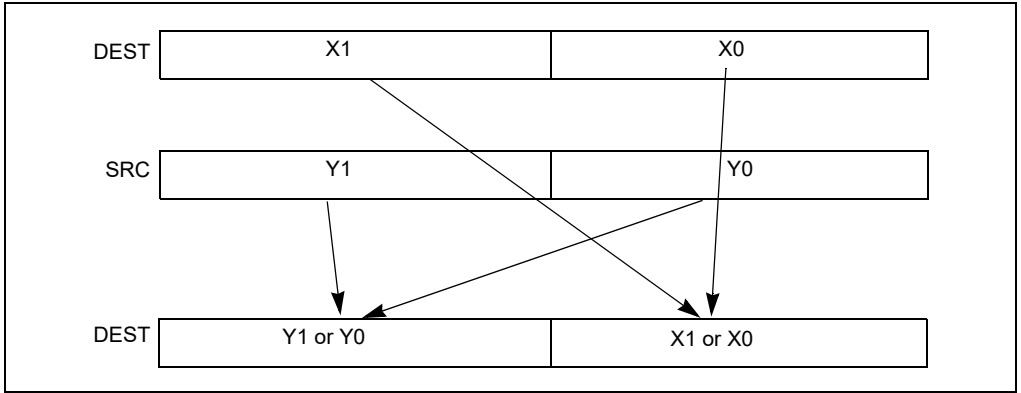


Figure 11-5. SHUFPS Instruction, Packed Shuffle Operation

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).

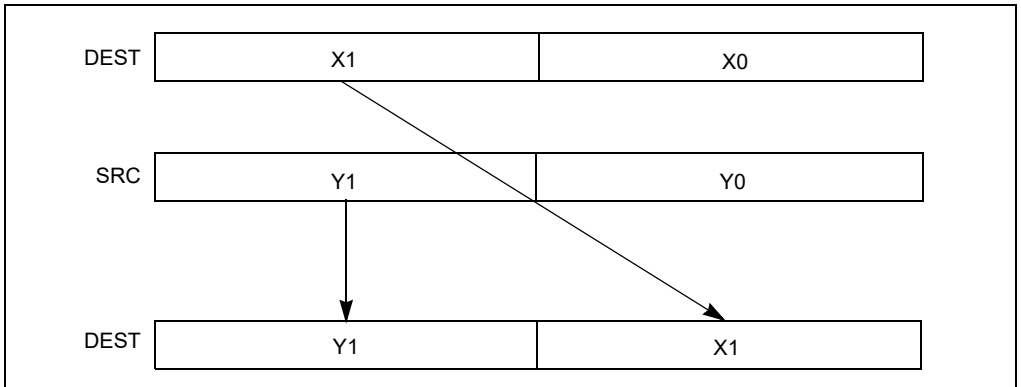


Figure 11-6. UNPCKHPD Instruction, High Unpack and Interleave Operation

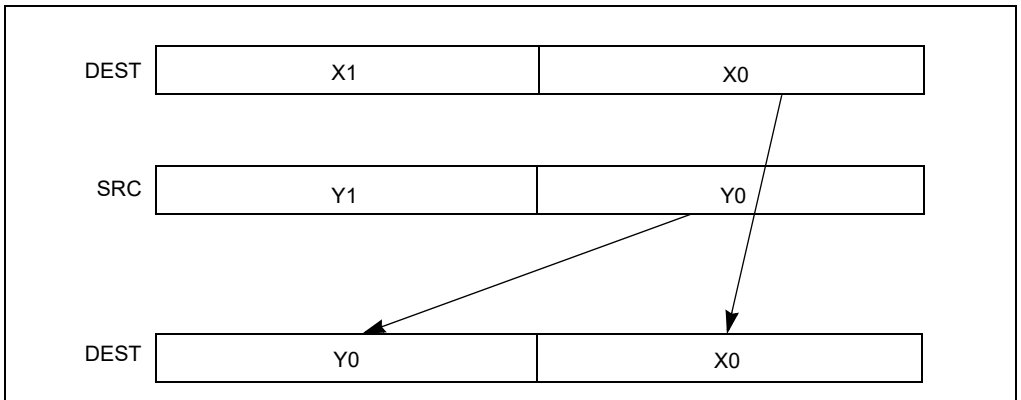


Figure 11-7. UNPCKLPD Instruction, Low Unpack and Interleave Operation

11.4.1.6 SSE2 Conversion Instructions

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 CVTSS2PD (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.

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.

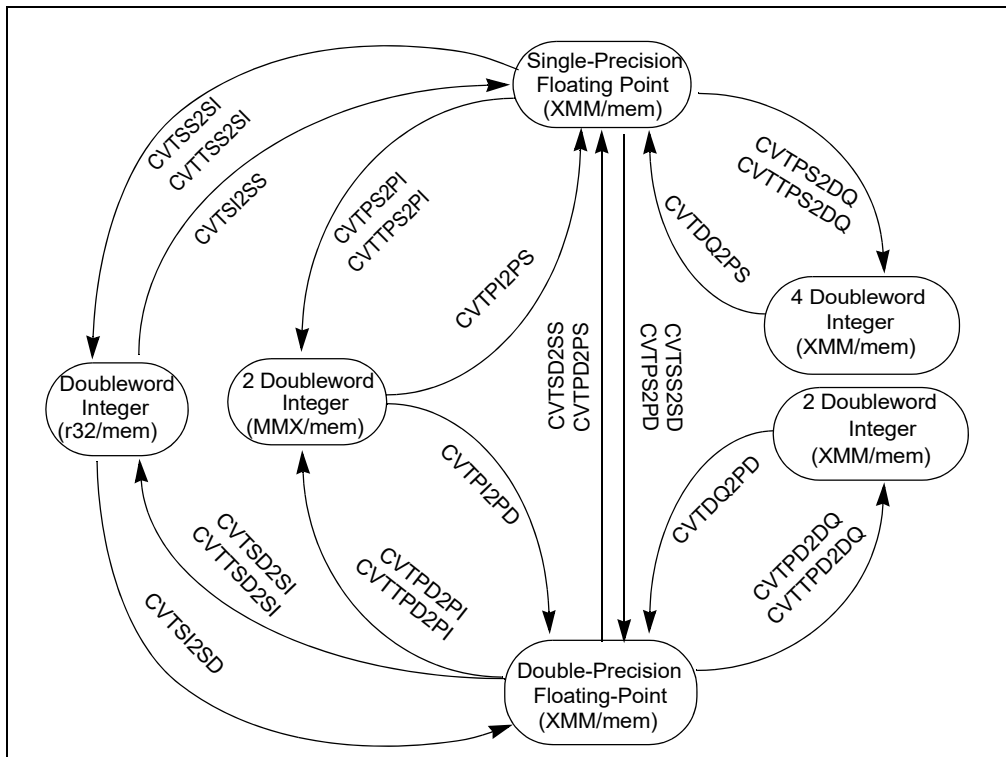


Figure 11-8. SSE and SSE2 Conversion Instructions

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 SSE and SSE2 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 SSE and SSE2 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 CVTSSD2SI (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 SSE and SSE2 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, CVTSSI2SS, CVTSS2SI, and CVTSS2SI) introduced with 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 SSE and SSE2 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 SSE2 64-Bit and 128-Bit SIMD Integer Instructions

SSE2 extensions add 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 PSHUFW (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 SSE extensions (with the exception of the PSHUFW instruction) have been extended by SSE2 extensions 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 PADDQ instruction operates on 8 packed bytes, the 128-bit version operates on 16 packed bytes.

11.4.4 Cacheability Control and Memory Ordering Instructions

SSE2 extensions 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 the SSE2 extensions. However, the instruction can be implemented in IA-32 processors that do not implement the SSE2 extensions. 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

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

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 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.

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

11.4.5 Branch Hints

SSE2 extensions designate 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 SSE, SSE2, AND SSE3 EXCEPTIONS

SSE/SSE2/SSE3 extensions generate two general types of exceptions:

- Non-numeric exceptions
- SIMD floating-point exceptions¹

SSE/SSE2/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 13 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 SSE/SSE2/SSE3 instructions and for guidelines for handling these exceptions.

SSE/SSE2/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 Binary 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 SSE/SSE2/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 SSE/SSE2/SSE3 extensions.

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

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-

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

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, CVTSD2SS, CVTSD2PS, 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.

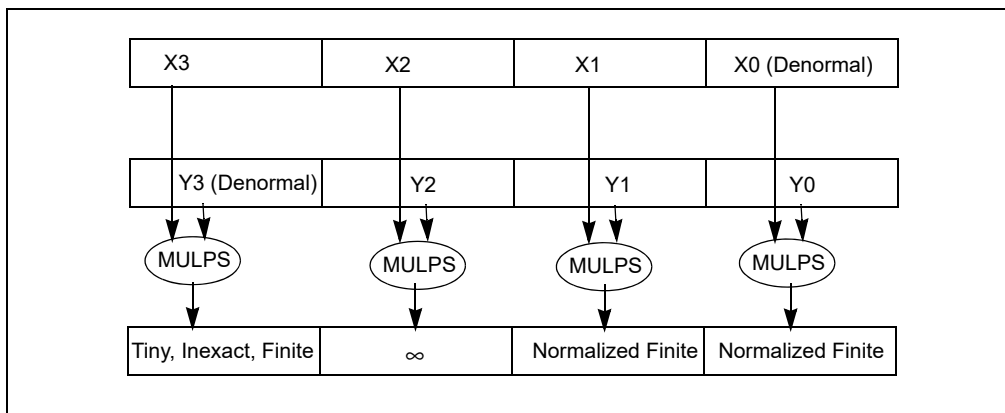


Figure 11-9. Example Masked Response for Packed Operations

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 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 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 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 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 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 SSE/SSE2/SSE3¹ 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 SSE/SSE2 EXTENSIONS

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

Chapter 13 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 SSE/SSE2/SSE3 extensions.

11.6.1 General Guidelines for Using SSE/SSE2 Extensions

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

- Ensure that the processor supports the SSE and SSE2 extensions.
- Ensure that your operating system supports the SSE and SSE2 extensions. (Operating system support for the SSE extensions implies support for SSE2 extension 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 the SSE and SSE2 extensions.
- Employ the optimization and scheduling techniques described in the *Intel Pentium 4 Optimization Reference Manual* (see Section 1.4, "Related Literature," for the order number for this manual).

11.6.2 Checking for SSE/SSE2 Support

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

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 the SSE and/or SSE2 extensions (true if CPUID.01H:EDX.SSE[bit 25] = 1 and/or CPUID.01H:EDX.SSE2[bit 26] = 1).

Operating system must provide system level support for handling SSE state, exceptions before an application can use the SSE and/or SSE2 extensions (see *Chapter 13* in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).

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

If the processor attempts to execute an unsupported 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 SSE, executing an 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 SSE/SSE2 Extensions

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 SSE/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 SSE, SSE2, MMX, and x87 FPU instructions in the same code module or when mixing code modules that contain these instructions:

- Those 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 SSE and SSE2 instructions that operate on MMX registers (such as the CVTTPS2PI, CVTTPD2PI, CVTPI2PS, CVTPI2PD, CVTTPD2PI, 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 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 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

SSE and SSE2 extensions 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, the SSE and SSE2 extensions 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 the SSE and SSE2 extensions. 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

SSE and SSE2 extensions 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 SSE/SSE2 Procedures and Functions

SSE and SSE2 extensions 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

SSE2 extensions extend 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: PSHUFW, PSHUFLW, and PSHUFD.
- Use of the 64-bit shift by bit instructions (PSRLQ, PSSLQ) can be extended to 128 bits in either of two ways:
 - Use of PSRLQ and PSSLQ, 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 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

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. `PREFETCHh` 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.

`PREFETCHh` 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 `PREFETCHh` 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 the SSE/SSE2 Instructions

Table 11-3 describes the effects of instruction prefixes on SSE and SSE2 instructions. (Table 11-3 also applies to SIMD integer and SIMD floating-point instructions in 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 SSE/SSE2/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 SSE, SSE2, and SSE3 Instructions

Prefix Type	Effect on 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).
Branch Hint Prefixes(E2H and E3H)	Reserved and may result in unpredictable behavior.

2. Updates to Chapter 15, Volume 1

Change bars and green text show changes to Chapter 15 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter: Added clarifying statement to section 15.6.3, "Broadcast".

15.1 OVERVIEW

The Intel AVX-512 family comprises a collection of instruction set extensions, including AVX-512 Foundation, AVX-512 Exponential and Reciprocal instructions, AVX-512 Conflict, AVX-512 Prefetch, and additional 512-bit SIMD instruction extensions. Intel AVX-512 instructions are natural extensions to Intel AVX and Intel AVX2. Intel AVX-512 introduces the following architectural enhancements:

- Support for 512-bit wide vectors and SIMD register set. 512-bit register state is managed by the operating system using XSAVE/XRSTOR instructions introduced in 45 nm Intel 64 processors (see *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*, and *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).
- Support for 16 new, 512-bit SIMD registers (for a total of 32 SIMD registers, ZMM0 through ZMM31) in 64-bit mode. The extra 16 registers state is managed by the operating system using XSAVE/XRSTOR/XSAVEOPT.
- Support for 8 new opmask registers (k0 through k7) used for conditional execution and efficient merging of destination operands. The opmask register state is managed by the operating system using the XSAVE/XRSTOR/XSAVEOPT instructions.
- A new encoding prefix (referred to as EVEX) to support additional vector length encoding up to 512 bits. The EVEX prefix builds upon the foundations of the VEX prefix to provide compact, efficient encoding for functionality available to VEX encoding plus the following enhanced vector capabilities:
 - Opmasks.
 - Embedded broadcast.
 - Instruction prefix-embedded rounding control.
 - Compressed address displacements.

15.1.1 512-Bit Wide SIMD Register Support

Intel AVX-512 instructions support 512-bit wide SIMD registers (ZMM0-ZMM31). The lower 256-bits of the ZMM registers are aliased to the respective 256-bit YMM registers and the lower 128-bit are aliased to the respective 128-bit XMM registers.

15.1.2 32 SIMD Register Support

Intel AVX-512 instructions also support 32 SIMD registers in 64-bit mode (XMM0-XMM31, YMM0-YMM31 and ZMM0-ZMM31). The number of available vector registers in 32-bit mode is still 8.

15.1.3 Eight Opmask Register Support

Intel AVX-512 instructions support 8 opmask registers (k0-k7). The width of each opmask register is architecturally defined as size MAX_KL (64 bits). Seven of the eight opmask registers (k1-k7) can be used in conjunction with EVEX-encoded AVX-512 Foundation instructions to provide conditional execution and efficient merging of data elements in the destination operand. The encoding of opmask register k0 is typically used when all data elements (unconditional processing) are desired. Additionally, the opmask registers are also used as vector flags/element-level vector sources to introduce novel SIMD functionality as seen in new instructions such as VCOMPRESSPS.

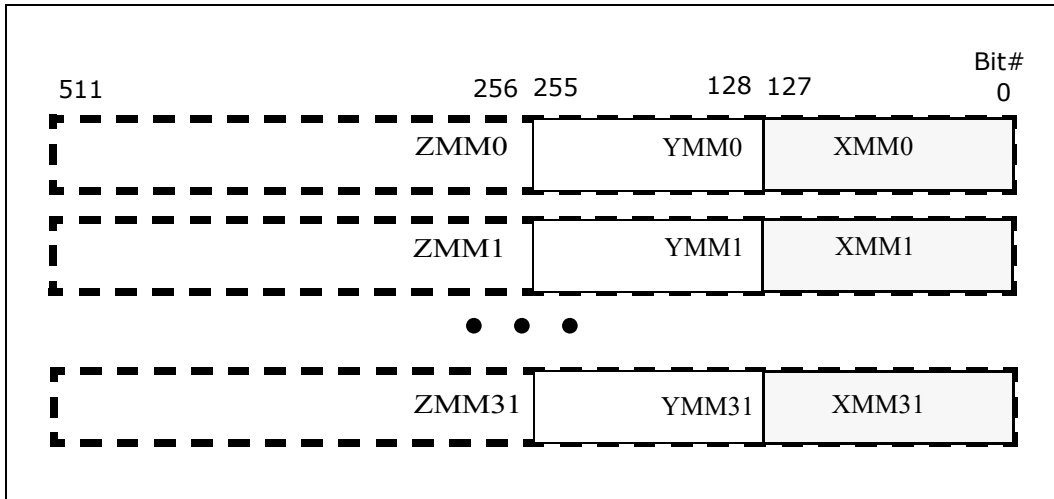


Figure 15-1. 512-Bit Wide Vectors and SIMD Register Set

15.1.4 Instruction Syntax Enhancement

The architecture of EVEX encoding enhances the vector instruction encoding scheme in the following way:

- 512-bit vector-length, up to 32 ZMM registers, and enhanced vector programming environment are supported using the enhanced VEX (EVEX).

The EVEX prefix provides more encodable bit fields than the VEX prefix. In addition to encoding 32 ZMM registers in 64-bit mode, instruction encoding using the EVEX prefix can directly encode 7 (out of 8) opmask register operands to provide conditional processing in vector instruction programming. The enhanced vector programming environment can be explicitly expressed in the instruction syntax to include the following elements:

- An opmask operand: the opmask registers are expressed using the notation "k1" through "k7". An EVEX-encoded instruction supporting conditional vector operation using the opmask register k1 is expressed by attaching the notation {k1} next to the destination operand. The use of this feature is optional for most instructions. There are two types of masking (merging and zeroing) differentiated using the EVEX.z bit ({z} in instruction signature).
- Embedded broadcast may be supported for some instructions on the source operand that can be encoded as a memory vector. Data elements of a memory vector may be conditionally fetched or written to.
- For instruction syntax that operates only on floating-point data in SIMD registers with rounding semantics, the EVEX encoding can provide explicit rounding control within the EVEX bit fields at either scalar or 512-bit vector length.

In AVX-512 instructions, vector addition of all elements of the source operands can be expressed in the same syntax as AVX instruction:

```
VADDPS zmm1, zmm2, zmm3
```

Additionally, the EVEX encoding scheme of AVX-512 Foundation can express conditional vector addition as:

```
VADDPS zmm1 {k1}{z}, zmm2, zmm3
```

where:

- Conditional processing and updates to destination are expressed with an opmask register.
- Zeroing behavior of the opmask selected destination element is expressed by the {z} modifier (with merging as the default if no modifier is specified).

Note that some SIMD instructions supporting three-operand syntax but processing only less than or equal to 128-bits of data are considered part of the 512-bit SIMD instruction set extensions, because bits MAXVL-1:128 of the destination register are zeroed by the processor. The same rule applies to instructions operating on 256-bits of data where bits MAXVL-1:256 of the destination register are zeroed.

15.1.5 EVEX Instruction Encoding Support

Intel AVX-512 instructions employ a new encoding prefix, referred to as EVEX, in the Intel 64 and IA-32 instruction encoding format. Instruction encoding using the EVEX prefix provides the following capabilities:

- Direct encoding of a SIMD register operand within EVEX (similar to VEX). This provides instruction syntax support for three source operands.
- Compaction of REX prefix functionality and extended SIMD register encoding: the equivalent REX-prefix compaction functionality offered by the VEX prefix is provided within EVEX. Furthermore, EVEX extends the operand encoding capability to allow direct addressing of up to 32 ZMM registers in 64-bit mode.
- Compaction of SIMD prefix functionality and escape byte encoding: the functionality of a SIMD prefix (66H, F2H, F3H) on opcode is equivalent to an opcode extension field to introduce new processing primitives. This functionality is provided in the VEX prefix encoding scheme and employed within the EVEX prefix. Similarly, the functionality of the escape opcode byte (0FH) and two-byte escape (0F38H, 0F3AH) are also compacted within the EVEX prefix encoding.
- Most EVEX-encoded SIMD numeric and data processing instruction semantics with memory operands have more relaxed memory alignment requirements than instructions encoded using SIMD prefixes (see Section 15.7, “Memory Alignment”).
- Direct encoding of an opmask operand within the EVEX prefix. This provides instruction syntax support for conditional vector-element operation and merging of destination operand using an opmask register (k1-k7).
- Direct encoding of a broadcast attribute for instructions with a memory operand source. This provides instruction syntax support for elements broadcasting the second operand before being used in the actual operation.
- Compressed memory address displacements for a more compact instruction encoding byte sequence.

EVEX encoding applies to SIMD instructions operating on XMM, YMM and ZMM registers. EVEX is not supported for instructions operating on MMX or x87 registers. Details of EVEX instruction encoding are discussed in Section 2.6, “Intel® AVX-512 Encoding” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*.

15.2 DETECTION OF AVX-512 FOUNDATION INSTRUCTIONS

The majority of AVX-512 Foundation instructions are encoded using the EVEX encoding scheme. EVEX-encoded instructions can operate on the 512-bit ZMM register state plus 8 opmask registers. The opmask instructions in AVX-512 Foundation instructions operate only on opmask registers or with a general purpose register. System software requirements to support the ZMM state and opmask instructions are described in Section 15.5, “Accessing XMM, YMM AND ZMM Registers”.

Processor support of AVX-512 Foundation instructions is indicated by CPUID.(EAX=07H, ECX=0):EBX.AVX512F[bit 16] = 1. Detection of AVX-512 Foundation instructions operating on ZMM states and opmask registers needs to follow the general procedural flow in Figure 15-2.

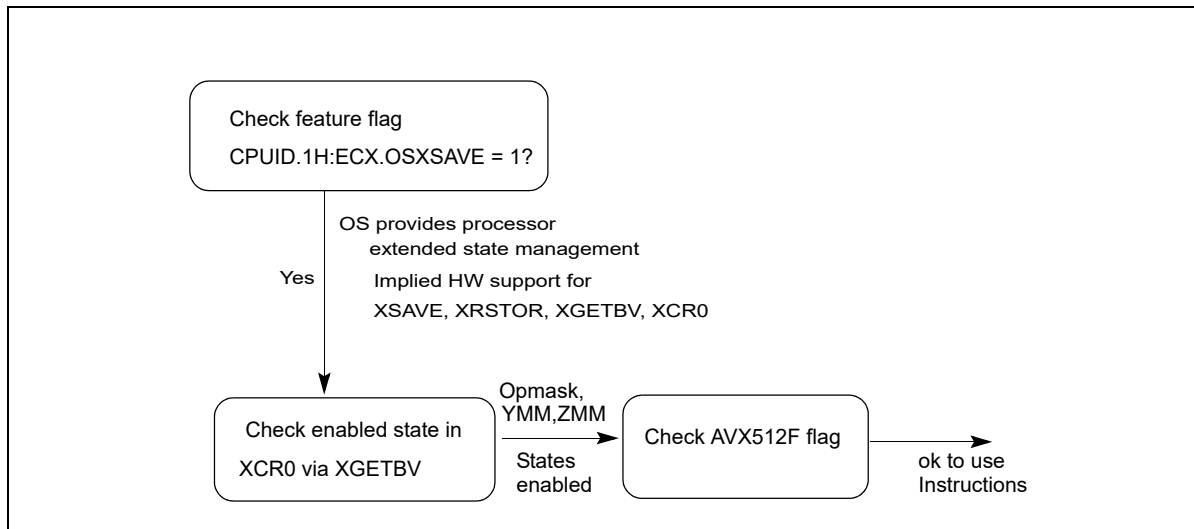


Figure 15-2. Procedural Flow for Application Detection of AVX-512 Foundation Instructions

Prior to using AVX-512 Foundation instructions, the application must identify that the operating system supports the XGETBV instruction and the ZMM register state, in addition to confirming the processor’s support for ZMM state management using XSAVE/XRSTOR and AVX-512 Foundation instructions. The following simplified sequence accomplishes both and is strongly recommended.

1. Detect CPUID.1:ECX.OSXSAVE[bit 27] = 1 (XGETBV enabled for application use¹).
2. Execute XGETBV and verify that XCR0[7:5] = ‘111b’ (OPMASK state, upper 256-bit of ZMM0-ZMM15 and ZMM16-ZMM31 state are enabled by OS) and that XCR0[2:1] = ‘11b’ (XMM state and YMM state are enabled by OS).
3. Detect CPUID.0x7.0:EBX.AVX512F[bit 16] = 1.

15.2.1 Additional 512-bit Instruction Extensions of the Intel AVX-512 Family

Processor support of the Intel AVX-512 Exponential and Reciprocal instructions are indicated by querying the feature flag:

- If CPUID.(EAX=07H, ECX=0):EBX.AVX512ER[bit 27] = 1, the collection of VEXP2PD/VEXP2PS/VRCP28xx/VRSQRT28xx instructions are supported.

Processor support of the Intel AVX-512 Prefetch instructions are indicated by querying the feature flag:

- If CPUID.(EAX=07H, ECX=0):EBX.AVX512PF[bit 26] = 1, a collection of VGATHERPF0xxx/VGATHERPF1xxx/VSCATTERPF0xxx/VSCATTERPF1xxx instructions are supported.

Detection of 512-bit instructions operating on ZMM states and opmask registers, outside of AVX-512 Foundation, needs to follow the general procedural flow in Figure 15-3.

1. If CPUID.01H:ECX.OSXSAVE reports 1, it also indirectly implies the processor supports XSAVE, XRSTOR, XGETBV, processor extended state bit vector XCR0 register. Thus an application may streamline the checking of CPUID feature flags for XSAVE and OSXSAVE. XSETBV is a privileged instruction.

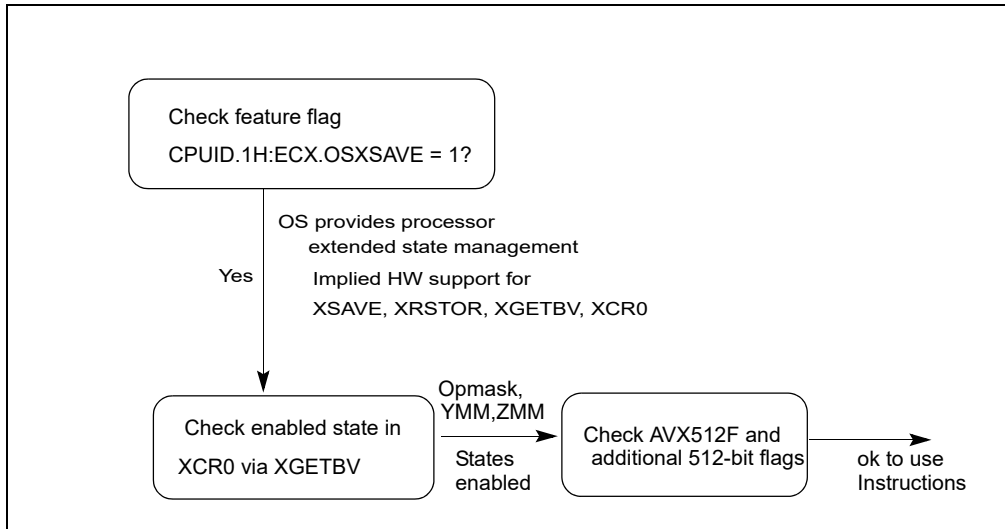


Figure 15-3. Procedural Flow for Application Detection of 512-bit Instructions

PREFETCH1W does not require OS support for XMM/YMM/ZMM/k-reg, SIMD FP exception support.

Procedural Flow of Application Detection of other 512-bit extensions:

Prior to using the Intel AVX-512 Exponential and Reciprocal instructions, the application must identify that the operating system supports the XGETBV instruction and the ZMM register state, in addition to confirming the processor's support for ZMM state management using XSAVE/XRSTOR and AVX-512 Foundation instructions. The following simplified sequence accomplishes both and is strongly recommended.

1. Detect $\text{CPUID.1:ECX.OSXSAVE}[\text{bit } 27] = 1$ (XGETBV enabled for application use).
2. Execute XGETBV and verify that $\text{XCR0}[7:5] = '111b'$ (OPMASK state, upper 256-bit of ZMM0-ZMM15 and ZMM16-ZMM31 state are enabled by OS) and that $\text{XCR0}[2:1] = '11b'$ (XMM state and YMM state are enabled by OS).
3. Verify both $\text{CPUID.0x7.0:EBX.AVX512F}[\text{bit } 16] = 1$, and $\text{CPUID.0x7.0:EBX.AVX512ER}[\text{bit } 27] = 1$.

Prior to using the Intel AVX-512 Prefetch instructions, the application must identify that the operating system supports the XGETBV instruction and the ZMM register state, in addition to confirming the processor's support for ZMM state management using XSAVE/XRSTOR and AVX-512 Foundation instructions. The following simplified sequence accomplishes both and is strongly recommended.

1. Detect $\text{CPUID.1:ECX.OSXSAVE}[\text{bit } 27] = 1$ (XGETBV enabled for application use).
2. Execute XGETBV and verify that $\text{XCR0}[7:5] = '111b'$ (OPMASK state, upper 256-bit of ZMM0-ZMM15 and ZMM16-ZMM31 state are enabled by OS) and that $\text{XCR0}[2:1] = '11b'$ (XMM state and YMM state are enabled by OS).
3. Verify both $\text{CPUID.0x7.0:EBX.AVX512F}[\text{bit } 16] = 1$, and $\text{CPUID.0x7.0:EBX.AVX512PF}[\text{bit } 26] = 1$.

15.3 DETECTION OF 512-BIT INSTRUCTION GROUPS OF INTEL® AVX-512 FAMILY

In addition to the Intel AVX-512 Foundation instructions, Intel AVX-512 family provides several groups of instruction extensions that can operate in vector lengths of 512/256/128 bits. Each group is enumerated by a CPUID leaf 7 feature flag and can be encoded via the EVEX.L'L field to support operation at vector lengths smaller than 512 bits. These instruction groups are listed in Table 15-1.

Table 15-1. 512-bit Instruction Groups in the Intel AVX-512 Family

CPUID Leaf 7 Feature Flag Bit	Feature Flag abbreviation of 512-bit Instruction Group	SW Detection Flow
CPUID.(EAX=07H, ECX=0):EBX[bit 16]	AVX512F (AVX-512 Foundation)	Figure 15-2
CPUID.(EAX=07H, ECX=0):EBX[bit 28]	AVX512CD	Figure 15-4
CPUID.(EAX=07H, ECX=0):EBX[bit 17]	AVX512DQ	Figure 15-4
CPUID.(EAX=07H, ECX=0):EBX[bit 30]	AVX512BW	Figure 15-4

Software must follow the detection procedure for the 512-bit AVX-512 Foundation instructions as described in Section 15.2.

Detection of other 512-bit sibling instruction groups listed in Table 15-1 (excluding AVX512F) follows the procedure described in Figure 15-4:

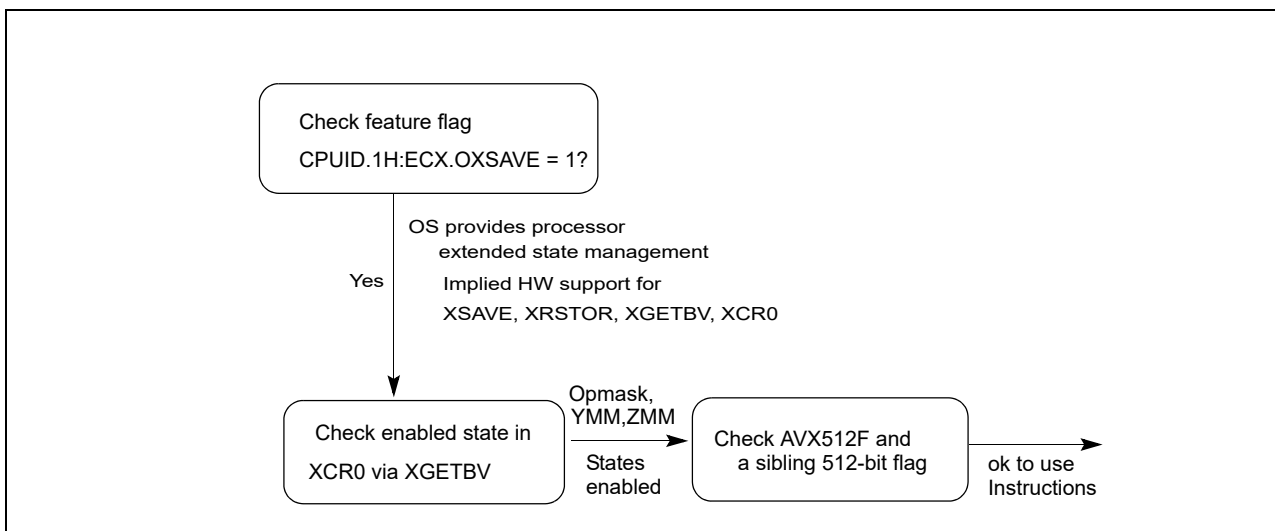


Figure 15-4. Procedural Flow for Application Detection of 512-bit Instruction Groups

To detect 512-bit instructions enumerated by AVX512CD, the following sequence is strongly recommended.

1. Detect CPUID.1:ECX.OSXSAVE[bit 27] = 1 (XGETBV enabled for application use).
2. Execute XGETBV and verify that XCR0[7:5] = '111b' (OPMASK state, upper 256-bit of ZMM0-ZMM15 and ZMM16-ZMM31 state are enabled by OS) and that XCR0[2:1] = '11b' (XMM state and YMM state are enabled by OS).
3. Verify both CPUID.0x7.0:EBX.AVX512F[bit 16] = 1, CPUID.0x7.0:EBX.AVX512CD[bit 28] = 1.

Similarly, the detection procedure for enumerating 512-bit instructions reported by AVX512DW follows the same flow.

15.4 DETECTION OF INTEL AVX-512 INSTRUCTION GROUPS OPERATING AT 256 AND 128-BIT VECTOR LENGTHS

For each of the 512-bit instruction groups in the Intel AVX-512 family listed in Table 15-1, the EVEX encoding scheme may support a vast majority of these instructions operating at 256-bit or 128-bit (if applicable) vector lengths. Encoding support for vector lengths smaller than 512-bits is indicated by CPUID.(EAX=07H, ECX=0):EBX[bit 31], abbreviated as AVX512VL.

The AVX512VL flag alone is never sufficient to determine a given Intel AVX-512 instruction may be encoded at vector lengths smaller than 512 bits. Software must use the procedure described in Figure 15-5 and Table 15-2.

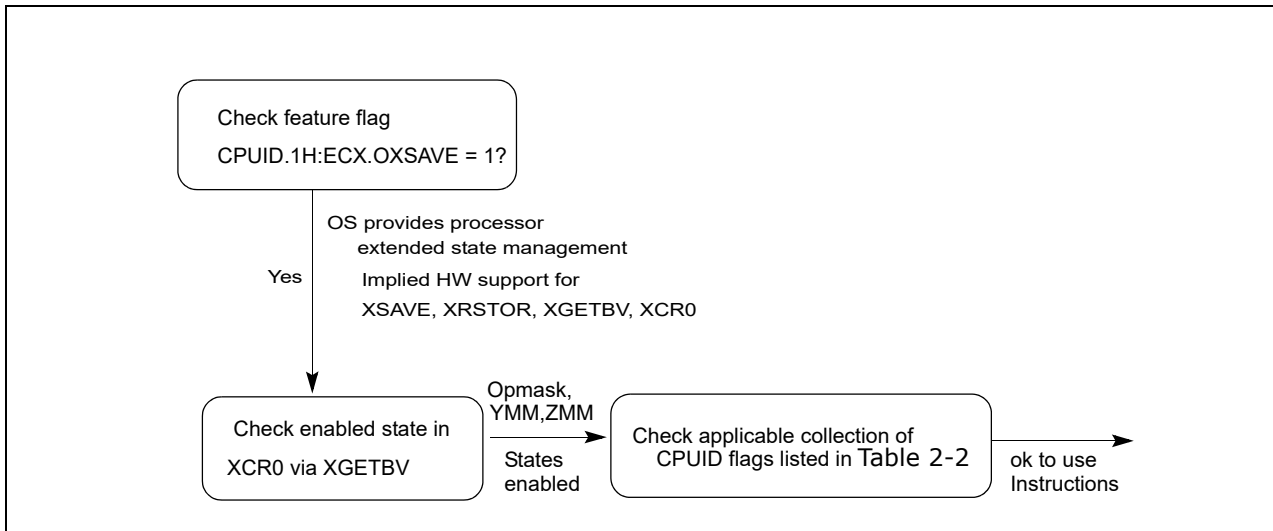


Figure 15-5. Procedural Flow for Detection of Intel AVX-512 Instructions Operating at Vector Lengths < 512

To illustrate the procedure described in Figure 15-5 and Table 15-2 for software to use EVEX.256 encoded VPCONFLICT, the following sequence is provided. It is strongly recommended this sequence is followed.

- 1) Detect CPUID.1:ECX.OSXSAVE[bit 27] = 1 (XGETBV enabled for application use).
- 2) Execute XGETBV and verify that XCR0[7:5] = '111b' (OPMASK state, upper 256-bit of ZMM0-ZMM15 and ZMM16-ZMM31 state are enabled by OS) and that XCR0[2:1] = '11b' (XMM state and YMM state are enabled by OS).
- 3) Verify CPUID.0x7.0:EBX.AVX512F[bit 16] = 1, CPUID.0x7.0:EBX.AVX512CD[bit 28] = 1, and CPUID.0x7.0:EBX.AVX512VL[bit 31] = 1.

Table 15-2. Feature flag Collection Required of 256/128 Bit Vector Lengths for Each Instruction Group

Usage of 256/128 Vector Lengths	Feature Flag Collection to Verify
AVX512F	AVX512F & AVX512VL
AVX512CD	AVX512F & AVX512CD & AVX512VL
AVX512DQ	AVX512F & AVX512DQ & AVX512VL
AVX512BW	AVX512F & AVX512BW & AVX512VL

In some specific cases, AVX512VL may only support EVEX.256 encoding but not EVEX.128. These cases are listed in Table 15-3.

Table 15-3. Instruction Mnemonics That Do Not Support EVEX.128 Encoding

Instruction Group	Instruction Mnemonics Supporting EVEX.256 Only Using AVX512VL
AVX512F	VBROADCASTSD, VBROADCASTF32X4, VEXTRACTI32X4, VINSERTF32X4, VINSERTI32X4, VPERMD, VPERMPD, VPERMPS, VPERMQ, VSHUFF32X4, VSHUFF64X2, VSHUFI32X4, VSHUFI64X2
AVX512CD	
AVX512DQ	VBROADCASTF32X2, VBROADCASTF64X2, VBROADCASTI32X4, VBROADCASTI64X2, VEXTRACTI64X2, VINSERTF64X2, VINSERTI64X2,
AVX512BW	

15.5 ACCESSING XMM, YMM AND ZMM REGISTERS

The lower 128 bits of a YMM register is aliased to the corresponding XMM register. Legacy SSE instructions (i.e., SIMD instructions operating on XMM state but not using the VEX prefix, also referred to non-VEX encoded SIMD instructions) will not access the upper bits (MAXVL-1:128) of the YMM registers. AVX and FMA instructions with a VEX prefix and vector length of 128-bits zeroes the upper 128 bits of the YMM register.

Upper bits of YMM registers (255:128) can be read and written to by many instructions with a VEX.256 prefix. XSAVE and XRSTOR may be used to save and restore the upper bits of the YMM registers.

The lower 256 bits of a ZMM register are aliased to the corresponding YMM register. Legacy SSE instructions (i.e., SIMD instructions operating on XMM state but not using the VEX prefix, also referred to non-VEX encoded SIMD instructions) will not access the upper bits (MAXVL-1:128) of the ZMM registers, where MAXVL is maximum vector length (currently 512 bits). AVX and FMA instructions with a VEX prefix and vector length of 128-bits zero the upper 384 bits of the ZMM register, while the VEX prefix and vector length of 256-bits zeroes the upper 256 bits of the ZMM register.

Upper bits of ZMM registers (511:256) can be read and written to by instructions with an EVEX.512 prefix.

15.6 ENHANCED VECTOR PROGRAMMING ENVIRONMENT USING EVEX ENCODING

EVEX-encoded AVX-512 instructions support an enhanced vector programming environment. The enhanced vector programming environment uses the combination of EVEX bit-field encodings and a set of eight opmask registers to provide the following capabilities:

- Conditional vector processing of an EVEX-encoded instruction. Opmask registers k1 through k7 can be used to conditionally govern the per-data-element computational operation and the per-element updates to the destination operand of an AVX-512 Foundation instruction. Each bit of the opmask register governs one vector element operation (a vector element can be 8 bits, 16 bits, 32 bits or 64 bits).
- In addition to providing predication control on vector instructions via EVEX bit-field encoding, the opmask registers can also be used similarly on general-purpose registers as source/destination operands using modR/M encoding for non-mask-related instructions. In this case, an opmask register k0 through k7 can be selected.
- In 64-bit mode, 32 vector registers can be encoded using the EVEX prefix.
- Broadcast may be supported for some instructions on the operand that can be encoded as a memory vector. The data elements of a memory vector may be conditionally fetched or written to, and the vector size is dependent on the data transformation function.
- Flexible rounding control for the register-to-register flavor of EVEX encoded 512-bit and scalar instructions. Four rounding modes are supported by direct encoding within the EVEX prefix, overriding MXCSR settings.
- Broadcast of one element to the rest of the destination vector register.
- Compressed 8-bit displacement encoding scheme to increase the instruction encoding density for instructions that normally require disp32 syntax.

15.6.1 OPMASK Register to Predicate Vector Data Processing

AVX-512 instructions using EVEX encode a predicate operand to conditionally control per-element computational operation and updating of the result to the destination operand. The predicate operand is known as the opmask register. The opmask is a set of eight architectural registers of size MAX_KL (64-bit). Note that from this set of eight architectural registers, only k1 through k7 can be addressed as a predicate operand. k0 can be used as a regular source or destination but cannot be encoded as a predicate operand. Note also that a predicate operand can be used to enable memory fault-suppression for some instructions with a memory operand (source or destination).

As a predicate operand, the opmask registers contain one bit to govern the operation/update to each data element of a vector register. In general, opmask registers can support instructions with all element sizes: byte (int8), word (int16), single-precision floating-point (float32), integer doubleword(int32), double-precision floating-point (float64), integer quadword (int64). Therefore, a ZMM vector register can hold 8, 16, 32, or 64 elements in principle. The length of an opmask register, MAX_KL, is sufficient to handle up to 64 elements with one bit per element, i.e., 64 bits. Masking is supported in most of the AVX-512 instructions. For a given vector length, each instruction accesses only the number of least significant mask bits that are needed based on its data type. For example, AVX-512 Foundation instructions operating on 64-bit data elements with a 512-bit vector length, only use the 8 least significant bits of the opmask register.

An opmask register affects an AVX-512 instruction at per-element granularity. Any numeric or non-numeric operation of each data element and per-element updates of intermediate results to the destination operand are predicated on the corresponding bit of the opmask register.

An opmask serving as a predicate operand in AVX-512 obeys the following properties:

- The instruction's operation is not performed for an element if the corresponding opmask bit is not set. This implies that no exception or violation can be caused by an operation on a masked-off element. Consequently, no MXCSR exception flag is updated as a result of a masked-off operation.
- A destination element is not updated with the result of the operation if the corresponding writemask bit is not set. Instead, the destination element value must be preserved (merging-masking) or it must be zeroed out (zeroing-masking).
- For some instructions with a memory operand, memory faults are suppressed for elements with a mask bit of 0.

Note that this feature provides a versatile construct to implement control-flow predication as the mask in effect provides a merging behavior for AVX-512 vector register destinations. As an alternative the masking can be used for zeroing instead of merging, so that the masked out elements are updated with 0 instead of preserving the old value. The zeroing behavior is provided to remove the implicit dependency on the old value when it is not needed.

Most instructions with masking enabled accept both forms of masking. Instructions that must have EVEX.aaa bits different than 0 (gather and scatter) and instructions that write to memory only accept merging-masking.

It's important to note that the per-element destination update rule also applies when the destination operand is a memory location. Vectors are written on a per element basis, based on the opmask register used as a predicate operand.

The value of an opmask register can be:

- Generated as a result of a vector instruction (e.g., CMP, FPCLASS, etc.).
- Loaded from memory.
- Loaded from a GPR register.
- Modified by mask-to-mask operations.

Opmask registers can be used for purposes outside of predication. For example, they can be used to manipulate sparse sets of elements from a vector, or used to set the EFLAGS based on the 0/0xFFFFFFFFFFFFFFFF/other status of the OR of two opmask registers.

15.6.1.1 Opmask Register K0

The only exception to the opmask rules described above is that opmask k0 can not be used as a predicate operand. Opmask k0 cannot be encoded as a predicate operand for a vector operation; the encoding value that would select opmask k0 will instead select an implicit opmask value of 0xFFFFFFFFFFFFFFFF, thereby effectively disabling

masking. Opmask register k0 can still be used for any instruction that takes opmask register(s) as operand(s) (either source or destination).

Note that certain instructions implicitly use the opmask as an extra destination operand. In such cases, trying to use the “no mask” feature will translate into a #UD fault being raised.

15.6.1.2 Example of Opmask Usages

The example below illustrates the predicated vector add operation and predicated updates of added results into the destination operand. The initial state of vector registers zmm0, zmm1, and zmm2 and k3 are:

```

MSB.....LSB

zmm0 =
[ 0x00000003 0x00000002 0x00000001 0x00000000 ] (bytes 15 through 0)
[ 0x00000007 0x00000006 0x00000005 0x00000004 ] (bytes 31 through 16)
[ 0x0000000B 0x0000000A 0x00000009 0x00000008 ] (bytes 47 through 32)
[ 0x0000000F 0x0000000E 0x0000000D 0x0000000C ] (bytes 63 through 48)

zmm1 =
[ 0x0000000F 0x0000000F 0x0000000F 0x0000000F ] (bytes 15 through 0)
[ 0x0000000F 0x0000000F 0x0000000F 0x0000000F ] (bytes 31 through 16)
[ 0x0000000F 0x0000000F 0x0000000F 0x0000000F ] (bytes 47 through 32)
[ 0x0000000F 0x0000000F 0x0000000F 0x0000000F ] (bytes 63 through 48)

zmm2 =
[ 0xAAAAAAAA 0xAAAAAAAA 0xAAAAAAAA 0xAAAAAAAA ] (bytes 15 through 0)
[ 0xBBBBBBBB 0xBBBBBBBB 0xBBBBBBBB 0xBBBBBBBB ] (bytes 31 through 16)
[ 0xCCCCCCCC 0xCCCCCCCC 0xCCCCCCCC 0xCCCCCCCC ] (bytes 47 through 32)
[ 0xDDDDDDDD 0xDDDDDDDD 0xDDDDDDDD 0xDDDDDDDD ] (bytes 63 through 48)

k3 = 0x8F03 (1000 1111 0000 0011)
    
```

An opmask register serving as a predicate operand is expressed as a curly-braces-enclosed decorator following the first operand in the Intel assembly syntax. Given this state, we will execute the following instruction:

```
vpaddq zmm2 {k3}, zmm0, zmm1
```

The vpaddq instruction performs 32-bit integer additions on each data element conditionally based on the corresponding bit value in the predicate operand k3. Since per-element operations are not operated if the corresponding bit of the predicate mask is not set, the intermediate result is:

```

[ ***** ***** 0x00000010 0x0000000F ] (bytes 15 through 0)
[ ***** ***** ***** ***** ] (bytes 31 through 16)
[ 0x0000001A 0x00000019 0x00000018 0x00000017 ] (bytes 47 through 32)
[ 0x0000001E ***** ***** ***** ] (bytes 63 through 48)
    
```

where “*****” indicates that no operation is performed.

This intermediate result is then written into the destination vector register, zmm2, using the opmask register k3 as the writemask, producing the following final result:


```

zmm2 =
[ 0xAAAAAAAA 0xAAAAAAAA 0x00000010 0x0000000F ] (bytes 15 through 0)
[ 0xBBBBBBBBB 0xBBBBBBBBB 0xBBBBBBBBB 0xBBBBBBBBB ] (bytes 31 through 16)
[ 0x0000001A 0x00000019 0x00000018 0x00000017 ] (bytes 47 through 32)
[ 0x0000001E 0xDDDDDDDD 0xDDDDDDDD 0xDDDDDDDD ] (bytes 63 through 48)

```

Note that for a 64-bit instruction (for example, `vaddpd`), only the 8 LSB of mask `k3` (`0x03`) would be used to identify the predicate operation on each one of the 8 elements of the source/destination vectors.

15.6.2 OpMask Instructions

AVX-512 Foundation instructions provide a collection of opmask instructions that allow programmers to set, copy, or operate on the contents of a given opmask register. There are three types of opmask instructions:

- **Mask read/write instructions:** These instructions move data between a general-purpose integer register or memory and an opmask mask register, or between two opmask registers. For example:
 - `kmovw k1, ebx`; move lower 16 bits of `ebx` to `k1`.
- **Flag instructions:** This category consists of instructions that modify EFLAGS based on the content of opmask registers.
 - `kortestw k1, k2`; OR registers `k1` and `k2` and updated EFLAGS accordingly.
- **Mask logical instructions:** These instructions perform standard bitwise logical operations between opmask registers.
 - `kandw k1, k2, k3`; AND lowest 16 bits of registers `k2` and `k3`, leaving the result in `k1`.

15.6.3 Broadcast

EVEX encoding provides a bit-field to encode data broadcast for some load-op instructions, i.e., instructions that load data from memory and perform some computational or data movement operation. A source element from memory can be broadcasted (repeated) across all the elements of the effective source operand (up to 16 times for a 32-bit data element, up to 8 times for a 64-bit data element). This is useful when **reusing** the same scalar operand for all the operations in a vector instruction. **Note that some processors may perform multiple loads of the source element and thus software should not rely on atomicity of the data being broadcast (e.g., when the source element is simultaneously modified by another logical processor).**

Broadcast is only enabled on instructions with an element size of 32 bits or 64 bits. Byte and word instructions do not support embedded broadcast.

The functionality of data broadcast is expressed as a curly-braces-enclosed decorator following the last register/memory operand in the Intel assembly syntax.

For instance:

```
vmulps zmm1, zmm2, [rax] {1to16}
```

The `{1to16}` primitive loads one float32 (single precision) element from memory, replicates it 16 times to form a vector of 16 32-bit floating-point elements, multiplies the 16 float32 elements with the corresponding elements in the first source operand vector, and puts each of the 16 results into the destination operand.

AVX-512 instructions with store semantics and pure load instructions do not support broadcast primitives.

```
vmovaps [rax] {k3}, zmm19
```

In contrast, the k3 opmask register is used as the predicate operand in the above example. Only the store operation on data elements corresponding to the non-zero bits in k3 will be performed.

15.6.4 Static Rounding Mode and Suppress All Exceptions

In previous SIMD instruction extensions (up to AVX and AVX2), rounding control is generally specified in MXCSR, with a handful of instructions providing per-instruction rounding override via encoding fields within the imm8 operand. AVX-512 offers a more flexible encoding attribute to override MXCSR-based rounding control for floating-pointing instructions with rounding semantics. This rounding attribute embedded in the EVEX prefix is called Static (per instruction) Rounding Mode or Rounding Mode override. This attribute allows programmers to statically apply a specific arithmetic rounding mode irrespective of the value of RM bits in MXCSR. It is available only to register-to-register flavors of EVEX-encoded floating-point instructions with rounding semantic. The differences between these three rounding control interfaces are summarized in Table 15-4.

Table 15-4. Characteristics of Three Rounding Control Interfaces

Rounding Interface	Static Rounding Override	Imm8 Embedded Rounding Override	MXCSR Rounding Control
Semantic Requirement	FP rounding	FP rounding	FP rounding
Prefix Requirement	EVEX.B = 1	NA	NA
Rounding Control	EVEX.L'L	IMM8[1:0] or MXCSR.RC (depending on IMM8[2])	MXCSR.RC
Suppress All Exceptions (SAE)	Implied	no	no
SIMD FP Exception #XM	All suppressed	Can raise #I, #P (unless SPE is set)	MXCSR masking controls
MXCSR flag update	No	yes (except PE if SPE is set)	Yes
Precedence	Above MXCSR.RC	Above EVEX.L'L	Default
Scope	512-bit, reg-reg, Scalar reg-reg	ROUNDPx, ROUNDSx, VCVTSP2PH, VRNDSCALExx	All SIMD operands, vector lengths

The static rounding-mode override in AVX-512 also implies the “suppress-all-exceptions” (SAE) attribute. The SAE effect is as if all the MXCSR mask bits are set, and none of the MXCSR flags will be updated. Using static rounding-mode via EVEX without SAE is not supported.

Static Rounding Mode and SAE control can be enabled in the encoding of the instruction by setting the EVEX.b bit to 1 in a register-register vector instruction. In such a case, vector length is assumed to be MAXVL (512-bit in case of AVX-512 packed vector instructions) or 128-bit for scalar instructions. Table 15-5 summarizes the possible static rounding-mode assignments in AVX-512 instructions.

Note that some instructions already allow specifying the rounding mode statically via immediate bits. In such cases, the immediate bits take precedence over the embedded rounding mode (in the same vein that they take precedence over whatever MXCSR.RM says).

Table 15-5. Static Rounding Mode

Function	Description
{rn-sae}	Round to nearest (even) + SAE
{rd-sae}	Round down (toward -inf) + SAE
{ru-sae}	Round up (toward +inf) + SAE
{rz-sae}	Round toward zero (Truncate) + SAE

An example of use would be as follows:

```
vaddps zmm7 {k6}, zmm2, zmm4, {rd-sae}
```

This would perform the single-precision floating-point addition of vectors zmm2 and zmm4 with round-towards-minus-infinity, leaving the result in vector zmm7 using k6 as conditional writemask.

Note that MXCSR.RM bits are ignored and unaffected by the outcome of this instruction.

Examples of instruction instances where the static rounding-mode is not allowed are shown below:

```
; rounding-mode already specified in the instruction immediate
vrndscaleps zmm7 {k6}, zmm2, 0x00

; instructions with memory operands
vmulps zmm7 {k6}, zmm2, [rax], {rd-sae}

; instructions with vector length different than MAXVL (512-bit)
vaddps ymm7 {k6}, ymm2, ymm4, {rd-sae}
```

15.6.5 Compressed Disp8*N Encoding

EVEX encoding supports a new displacement representation that allows for a more compact encoding of memory addressing commonly used in unrolled code, where an 8-bit displacement can address a range exceeding the dynamic range of an 8-bit value. This compressed displacement encoding is referred to as disp8*N, where N is a constant implied by the memory operation characteristic of each instruction.

The compressed displacement is based on the assumption that the effective displacement (of a memory operand occurring in a loop) is a multiple of the granularity of the memory access of each iteration. Since the base register in memory addressing already provides byte-granular resolution, the lower bits of the traditional disp8 operand become redundant, and can be implied from the memory operation characteristic.

The memory operation characteristics depend on the following:

- The destination operand is updated as a full vector, a single element, or multi-element tuples.
- The memory source operand (or vector source operand if the destination operand is memory) is fetched (or treated) as a full vector, a single element, or multi-element tuples.

For example:

```
vaddps zmm7, zmm2, disp8[membase + index*8]
```

The destination zmm7 is updated as a full 512-bit vector, and 64-bytes of data are fetched from memory as a full vector; the next unrolled iteration may fetch from memory in 64-byte granularity per iteration. There are 6 bits of lowest address that can be compressed, hence $N = 2^6 = 64$. The contribution of "disp8" to effective address calculation is $64 * \text{disp8}$.

```
vbroadcastf32x4 zmm7, disp8[membase + index*8]
```

In VBROADCASTF32x4, memory is fetched as a 4tuple of 4 32-bit entities. Hence the common lowest address bits that can be compressed are 4, corresponding to the 4tuple width of $2^4 = 16$ bytes (4x32 bits). Therefore, $N = 2^4$.

For EVEX encoded instructions that update only one element in the destination, or the source element is fetched individually, the number of lowest address bits that can be compressed is generally the width in bytes of the data element, hence $N = 2^{(\text{width})}$.

15.7 MEMORY ALIGNMENT

Memory alignment requirements on EVEX-encoded SIMD instructions are similar to VEX-encoded SIMD instructions. Memory alignment applies to EVEX-encoded SIMD instructions in three categories:

- Explicitly-aligned SIMD load and store instructions accessing 64 bytes of memory with EVEX prefix encoded vector length of 512 bits (e.g., VMOVAPD, VMOVAPS, VMOVDQA, etc.). These instructions always require the memory address to be aligned on a 64-byte boundary.
- Explicitly-unaligned SIMD load and store instructions accessing 64 bytes or less of data from memory (e.g., VMOVUPD, VMOVUPS, VMOVDQU, VMOVQ, VMOVD, etc.). These instructions do not require the memory address to be aligned on a natural vector-length byte boundary.
- Most arithmetic and data processing instructions encoded using EVEX support memory access semantics. When these instructions access from memory, there are no alignment restrictions.

Software may see performance penalties when unaligned accesses cross cacheline boundaries or vector-length naturally-aligned boundaries, so reasonable attempts to align commonly used data sets should continue to be pursued.

Atomic memory operation in Intel 64 and IA-32 architecture is guaranteed only for a subset of memory operand sizes and alignment scenarios. The guaranteed atomic operations are described in Section 8.1.1, “Guaranteed Atomic Operations” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. AVX and FMA instructions do not introduce any new guaranteed atomic memory operations.

AVX-512 instructions may generate an #AC(0) fault on misaligned 4 or 8-byte memory references in Ring-3 when CR0.AM=1. 16, 32 and 64-byte memory references will not generate an #AC(0) fault. See Table 15-7 for details.

Certain AVX-512 Foundation instructions always require 64-byte alignment (see the complete list of VEX and EVEX encoded instructions in Table 15-6). These instructions will #GP(0) if not aligned to 64-byte boundaries.

Table 15-6. SIMD Instructions Requiring Explicitly Aligned Memory

Require 16-byte alignment	Require 32-byte alignment	Require 64-byte alignment*
(V)MOVDQA xmm, m128	VMOVDQA ymm, m256	VMOVDQA zmm, m512
(V)MOVDQA m128, xmm	VMOVDQA m256, ymm	VMOVDQA m512, zmm
(V)MOVAPS xmm, m128	VMOVAPS ymm, m256	VMOVAPS zmm, m512
(V)MOVAPS m128, xmm	VMOVAPS m256, ymm	VMOVAPS m512, zmm
(V)MOVAPD xmm, m128	VMOVAPD ymm, m256	VMOVAPD zmm, m512
(V)MOVAPD m128, xmm	VMOVAPD m256, ymm	VMOVAPD m512, zmm
(V)MOVNTDQA xmm, m128	VMOVNTPS m256, ymm	VMOVNTPS m512, zmm
(V)MOVNTPS m128, xmm	VMOVNTPD m256, ymm	VMOVNTPD m512, zmm
(V)MOVNTPD m128, xmm	VMOVNTDQ m256, ymm	VMOVNTDQ m512, zmm
(V)MOVNTDQ m128, xmm	VMOVNTDQA ymm, m256	VMOVNTDQA zmm, m512

Table 15-7. Instructions Not Requiring Explicit Memory Alignment

(V)MOVDQU xmm, m128	VMOVDQU ymm, m256	VMOVDQU zmm, m512
(V)MOVDQU m128, m128	VMOVDQU m256, ymm	VMOVDQU m512, zmm
(V)MOVUPS xmm, m128	VMOVUPS ymm, m256	VMOVUPS zmm, m512
(V)MOVUPS m128, xmm	VMOVUPS m256, ymm	VMOVUPS m512, zmm
(V)MOVUPD xmm, m128	VMOVUPD ymm, m256	VMOVUPD zmm, m512
(V)MOVUPD m128, xmm	VMOVUPD m256, ymm	VMOVUPD m512, zmm

15.8 SIMD FLOATING-POINT EXCEPTIONS

AVX-512 instructions can generate SIMD floating-point exceptions (#XM) if embedded “suppress all exceptions” (SAE) in EVEX is not set. When SAE is not set, these instructions will respond to exception masks of MXCSR in the

same way as VEX-encoded AVX instructions. When CR4.OSXMMEXCPT=0, any unmasked FP exceptions generate an Undefined Opcode exception (#UD).

15.9 INSTRUCTION EXCEPTION SPECIFICATION

Exception behavior of VEX-encoded AVX / AVX2 instructions are described in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*. Exception behavior of AVX-512 Foundation instructions and additional 512-bit extensions are described in Section 2.7, "Exception Classifications of EVEX-Encoded instructions" and Section 2.8, "Exception Classifications of Opmask instructions".

15.10 EMULATION

Setting the CR0.EM bit to 1 provides a technique to emulate legacy SSE floating-point instruction sets in software. This technique is not supported with AVX instructions, nor FMA instructions.

If an operating system wishes to emulate AVX instructions, set XCR0[2:1] to zero. This will cause AVX instructions to #UD. Emulation of FMA by the operating system can be done similarly as with emulating AVX instructions.

15.11 WRITING FLOATING-POINT EXCEPTION HANDLERS

AVX-512, AVX and FMA floating-point exceptions are handled in an entirely analogous way to legacy SSE floating-point exceptions. To handle unmasked SIMD floating-point exceptions, the operating system or executive must provide an exception handler. Section 11.5.1, "SIMD Floating-Point Exceptions", describes the SIMD floating-point exception classes and gives suggestions for writing an exception handler to handle them.

To indicate that the operating system provides a handler for SIMD floating-point exceptions (#XM), the CR4.OSXMMEXCPT flag (bit 10) must be set.

3. Updates to Chapter 18, Volume 1

Change bars and green 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: Update to section 18.3.7.2, “#CP(ENDBRANCH) Priority w.r.t #NM and #UD”, to accurately reflect that #CP(ENDBRANCH) is in a higher priority class than #NM and #UD.

18.1 INTRODUCTION

Return-oriented programming (ROP), and similarly CALL/JMP-oriented programming (COP/JOP), have been the prevalent attack methodologies for stealth exploit writers targeting vulnerabilities in programs. These attack methodologies have the common elements:

- A code module with execution privilege and contain small snippets of code sequence with the characteristic: at least one instruction in the sequence being a control transfer instruction that depends on data either in the return stack or in a register for the target address.
- Diverting the control flow instruction (e.g., RET, CALL, JMP) from its original target address to a new target (via modification in the data stack or in the register).

Control-Flow Enforcement Technology (CET) provides the following capabilities to defend against ROP/COP/JOP style control-flow subversion attacks:

- Shadow stack: Return address protection to defend against ROP.
- Indirect branch tracking: Free branch protection to defend against COP/JOP.

Both capabilities introduce new instruction set extensions, and are described in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*.

Control-Flow Enforcement Technology introduces a new exception (#CP) with interrupt vector 21.

18.1.1 Shadow Stack

A shadow stack is a second stack for the program that is used exclusively for control transfer operations. This stack is separate from the data stack and can be enabled for operation individually in user mode or supervisor mode. When shadow stacks are enabled, the CALL instruction pushes the return address on both the data and shadow stack. The RET instruction pops the return address from both stacks and compares them. If the return addresses from the two stacks do not match, the processor signals a control protection exception (#CP). Note that the shadow stack only holds the return addresses and not parameters passed to the call instruction.

The shadow stack is protected from tamper through the page table protections such that regular store instructions cannot modify the contents of the shadow stack. To provide this protection the page table protections are extended to support an additional attribute for pages to mark them as "Shadow Stack" pages. When shadow stacks are enabled, control transfer instructions/flows like near call, far call, call to interrupt/exception handlers, etc. store return addresses to the shadow stack and the access will fault if the underlying page is not marked as a "Shadow Stack" page. However stores from instructions like MOV, XSAVE, etc. will not be allowed. Likewise control transfer instructions like near RET, far RET, IRET, etc. when they attempt to read from the shadow stack the access will fault if the underlying page is not marked as a "Shadow Stack" page. This paging protection detects and prevents conditions that cause an overflow or underflow of the shadow stack when the shadow stack is delimited by non-shadow stack guard pages, or any malicious attempts to redirect the processor to consume data from addresses that are not shadow stack addresses.

18.1.2 Indirect Branch Tracking

The ENDBRANCH instruction is a new instruction that is used to mark valid jump target addresses of indirect calls and jumps in the program. This instruction opcode is selected to be one that is a NOP on legacy machines such that programs compiled with ENDBRANCH new instruction continue to function on old machines without the CET enforcement. On processors that support CET the ENDBRANCH is still a NOP and is primarily used as a marker instruction by the processor pipeline to detect control flow violations. The CPU implements a state machine that tracks indirect JMP and CALL instructions. When one of these instructions is executed, the state machine moves from IDLE to WAIT_FOR_ENDBRANCH state. In WAIT_FOR_ENDBRANCH state the next instruction in the program

stream must be an ENDBRANCH. If the next instruction is not an ENDBRANCH, the processor causes a control protection exception (#CP); otherwise, the state machine moves back to IDLE state.

18.1.3 Speculative Behavior when CET is Enabled

Speculative execution of near indirect JMP/CALL/RET indirect branches may be able to create an active side channel vulnerability that reveals the contents of data.

There are two basic methods that an attacker may be able to use to control indirect branch speculation in order to speculatively execute code that causes a side channel:

1. Attacker controlled prediction.
2. Attacker controlled jump redirection.

With attacker controlled prediction, the attacker trains indirect branch predictors such that the desired victim indirect branch goes to the attacker desired location. Examples include Branch Target Injection (also called "Variant 2" and "Spectre") and RSB wrap on underflow (also called "ret2spec").

With attacker controlled jump redirection, the attacker controls a speculative-only value used as input to the indirect branch so that the branch mispredicts to the attacker desired location. Examples of this include Bound Check Bypass Store (where a speculative store containing an attacker controlled value may overwrite the indirect branch target before the load of the target) and Speculative Store Bypass (where a load of the indirect branch target may bypass the most recent store of the target value and thus speculatively read an older attacker controlled value at the same memory location).

In addition to the existing mitigation features like IBRS, STIBP and IBPB, processors supporting CET will have a variety of additional features to constrain control flow speculation in order to mitigate such attacks. For details on these features, see Section 18.2.6, "Constraining Execution at Targets of RET" and Section 18.3.8, "Constraining Speculation after Missing ENDBRANCH".

18.2 SHADOW STACKS

A shadow stack is a second expand down stack used exclusively for control transfer operations. This stack is separate from the data stack. The shadow stack is not used to store data and hence is not explicitly writeable by software. Writes to the shadow stack are restricted to control transfer instructions and shadow stack management instructions. The shadow stack feature can be enabled separately in user mode (CPL == 3) or supervisor mode (CPL < 3).

Shadow stacks operate only in protected mode. Shadow stacks cannot be enabled in virtual 8086 mode.

It is recommended to not configure the shadow stack in the linear address range 0 to 64 KB or adjacent to the canonical address boundary.

18.2.1 Shadow Stack Pointer and its Operand and Address Size Attributes

When CET is enabled the processor supports a new architectural register, shadow stack pointer (SSP), when the processor supports the shadow stack feature. The SSP cannot be directly encoded as a source, destination or memory operand in instructions. The SSP points to the current top of the shadow stack.

The width of the shadow stack is 32-bit in 32-bit/compatibility mode and is 64-bit in 64-bit mode. The address-size attribute of the shadow stack is likewise 32-bit in 32-bit/compatibility mode and 64-bit in 64-bit mode.

18.2.2 Terminology

When shadow stacks are enabled, certain control transfer instructions/flows and shadow stack management instructions do loads/stores to the shadow stack. Such load/stores from control transfer instructions and shadow stack management instructions are termed as shadow_stack_load and shadow_stack_store to distinguish them from a load/store performed by other instructions like MOV, XSAVES, etc.

The pseudocode for the instruction operations use the notation `ShadowStackEnabled(CPL)` as a test of whether shadow stacks are enabled at the CPL. This term returns a TRUE or FALSE indication as follows.

```
ShadowStackEnabled(CPL):
  IF CR4.CET = 1 AND CR0.PE = 1 AND EFLAGS.VM = 0
    IF CPL = 3
      THEN
        (* Obtain the shadow stack enable from IA32_U_CET MSR (MSR address 6A0H) used to enable
           feature for CPL = 3 *)
        SHADOW_STACK_ENABLED = IA32_U_CET.SH_STK_EN;
      ELSE
        (* Obtain the shadow stack enable from IA32_S_CET MSR (MSR address 6A2H) used to enable
           feature for CPL < 3 *)
        SHADOW_STACK_ENABLED = IA32_S_CET.SH_STK_EN;
    FI;
    IF SHADOW_STACK_ENABLED = 1
      THEN
        return TRUE;
      ELSE
        return FALSE;
    FI;
  ELSE
    (* Shadow stacks not enabled in real mode and virtual-8086 mode or if the master CET feature
       enable in CR4 is disabled *)
    return FALSE;
  ENDIF
```

Additionally, the following terms are used.

- `ShadowStackPush4B`: Decrements the shadow stack pointer (SSP) by 4 bytes and copies the 4 byte source operand to the top of the shadow stack.
- `ShadowStackPush8B`: Decrements the shadow stack pointer (SSP) by 8 bytes and copies the 8 byte source operand to the top of the shadow stack.
- `ShadowStackPop4B`: Copies 4 bytes at the current top of stack (indicated by the SSP register) to the location specified with the destination operand. It then increments the SSP register by 4 bytes to point to the new top of stack.
- `ShadowStackPop8B`: Copies 8 bytes at the current top of stack (indicated by the SSP register) to the location specified with the destination operand. It then increments the SSP register by 8 bytes to point to the new top of stack.
- `shadow_stack_lock_cmpxchg8B(address, new_value, expected_value)`: this function executes atomically and compares the `expected_value` to the 8 byte read from memory specified by the `address` operand using a locked `shadow_stack_load`. If the two values are equal, the `new_value` is written to memory specified by the `address` operand using a locked `shadow_stack_store`. If the two values are not equal, then the value read by the locked `shadow_stack_load` is written back. The memory specified by the `address` operand receives a write cycle without regard to the result of the comparison. The function returns the value read from the memory specified by the `address` operand.

18.2.3 Supervisor Shadow Stack Token

On an inter-privilege far `CALL` or when calling an interrupt/exception handler at a higher privilege level, a stack switch occurs; if shadow stacks are enabled at the new privilege level, then a shadow stack switch occurs. Shadow stacks that can be switched to by hardware as part of a privilege change are required to have a supervisor shadow stack token set up by the supervisor to provide the address of the new SSP register. The supervisor shadow stack tokens also serve the purpose of enforcing that a shadow stack can be made active on only one logical processor when switched to by the processor. The supervisor shadow stack token must be set up only on shadow stacks

intended to be used on these transfers. The address of the supervisor shadow stack token is programmed into the IA32_PLx_SSP MSR (where $0 \leq x \leq 2$). The WRMSR and XRSTORS instructions require the address specified in the IA32_PLx_SSP MSR (where $0 \leq x \leq 2$) to be 4 byte aligned; otherwise, the instruction causes a general protection exception (#GP(0)).

The supervisor shadow stack token is a 64-bit value formulated as follows.

- Bit 63:3: Bits 63:3 of the linear address of the supervisor shadow stack token.
- Bit 2: Reserved. Must be zero.
- Bit 1: Reserved. Must be zero.
- Bit 0: Busy bit. If 0, indicates this shadow stack is not active on any logical processor. If 1, indicates this shadow stack is currently active on one of the logical processors.

The following figure illustrates a supervisor shadow stack with a supervisor shadow stack token located at its base.

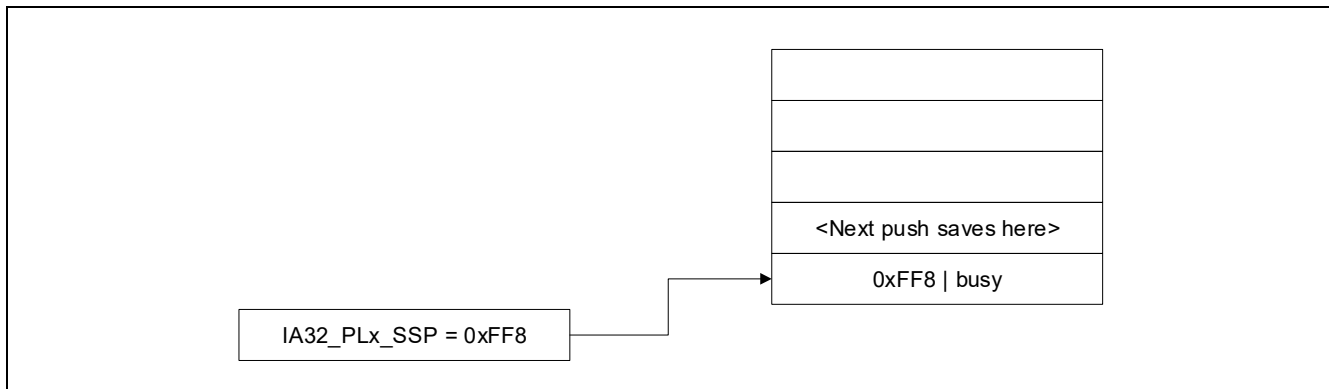


Figure 18-1. Supervisor Shadow Stack with a Supervisor Shadow Stack Token

The 8-byte supervisor shadow stack token and the 24-byte stack frame pushed on the shadow stack after the token is acquired must be fully contained within a 32-byte region that is aligned to 32-bytes on the shadow stack. If they are not, a general-protection exception (#GP(0)) occurs. While the accesses to the token may cause faults, VM exits, or data breakpoints, the subsequent pushes of the remaining 24 bytes should not.

The processor does the following checks prior to switching to a supervisor shadow stack programmed into the IA32_PLx_SSP MSR. These steps are performed atomically.

1. Load the supervisor shadow stack token from the address specified in the IA32_PLx_SSP MSR using a shadow_stack_load.
2. Check if the busy bit in the token is 0; reserved bits must be 0.
3. Check if the address programmed in the MSR matches the address in the supervisor shadow stack token; reserved bits must be 0.
4. If checks 2 and 3 are successful, then set the busy bit in the token using a shadow_stack_store and switch the SSP to the value specified in the IA32_PLx_SSP MSR.
5. If checks 2 or 3 fail, then the busy bit is not set and a #GP(0) exception is raised.

On a far RET to a lesser privilege level or on an IRET that switches shadow stack, the instruction clears the busy bit in the shadow stack token as follows. These steps are also performed atomically.

1. Load the supervisor shadow stack token from the SSP using a shadow_stack_load.
2. Check if the busy bit in the token is 1; reserved bits must be 0.
3. Check if the address programmed in supervisor shadow stack token matches SSP; reserved bits must be 0.
4. If checks 2 and 3 are successful, then clear the busy bit in the token using a shadow_stack_store; else continue without modifying the contents of the shadow stack pointed to by SSP.

18.2.4 Shadow Stack Usage on Task Switch

A task switch (see Chapter 7, “Task Management” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*) may be invoked by:

- JMP or CALL instruction to a TSS descriptor in the GDT.
- JMP or CALL instruction to a task-gate descriptor in the GDT or the current LDT.
- An interrupt or exception vector points to a task-gate descriptor in the IDT.

With shadow stack enabled, the new task must be associated with a 32-bit TSS and must not be in virtual-8086 mode. The 32-bit SSP for the new task is located at offset 104 in the 32-bit TSS. Thus the TSS of the new task must be at least 108 bytes. This SSP is required to be 8 byte aligned, and required to point to a “supervisor shadow stack” token (though the task may be at CPL3).

On a task switch initiated by a CALL instruction, an interrupt, or exception, the SSP of the old task is pushed onto the shadow stack of the new task along with the CS and LIP of the old task. This is true even for a nested task switch initiated by a CALL instruction. Likewise, on a task switch initiated by IRET, the SSP of the new task is restored from the shadow stack of old task. The CS and LIP on the shadow stack of the old task are matched against the return address determined by the CS and EIP of the new task. If the match fails, a #CP(FAR-RET/IRET) exception is reported.

18.2.5 Switching Shadow Stacks

The architecture provides a mechanism to switch shadow stacks using a pair of instructions; RSTORSSP and SAVEPREVSSP. The RSTORSSP instruction verifies a shadow-stack-restore token located at the top of the new shadow stack and referenced by the memory operand of this instruction. After RSTORSSP determines the validity of the restore point on the new shadow stack, it switches the SSP to point to the token. The shadow-stack-restore token is a 64-bit value formatted as follows.

- Bit 63:2: Value of shadow stack pointer when this restore point was created.
- Bit 1: Reserved. Must be zero.
- Bit 0: Mode bit. If 0, the token is a compatibility/legacy mode shadow-stack-restore token. If 1, then this shadow stack restore token can be used with a RSTORSSP instruction in 64-bit mode.

The shadow-stack-restore token is created by the SAVEPREVSSP instruction. The operating system may also create a restore point on a shadow stack by creating a shadow-stack-restore token.

Once the shadow stack has been switched to a new shadow stack by the RSTORSSP instruction, software can create a restore point on the old shadow stack by executing the SAVEPREVSSP instruction. In order to allow the SAVEPREVSSP instruction to determine the address where to save the shadow-stack-restore token, the RSTORSSP instruction replaces the shadow-stack-restore token with a previous-ssp token that holds the value of the SSP at the time the RSTORSSP instruction was invoked. The previous-ssp token is formatted as follows.

- Bit 63:2: Shadow stack pointer when the RSTORSSP instruction was invoked, i.e., the SSP of the old shadow stack.
- Bit 1: Set to 1.
- Bit 0: Mode bit. If 0, then this previous-ssp token can be used with a SAVEPREVSSP instruction in compatibility/legacy mode. If 1, then this previous-ssp token can be used with a SAVEPREVSSP instruction in 64-bit mode.

The following figure illustrates the RSTORSSP instruction operation during a shadow stack switching sequence.

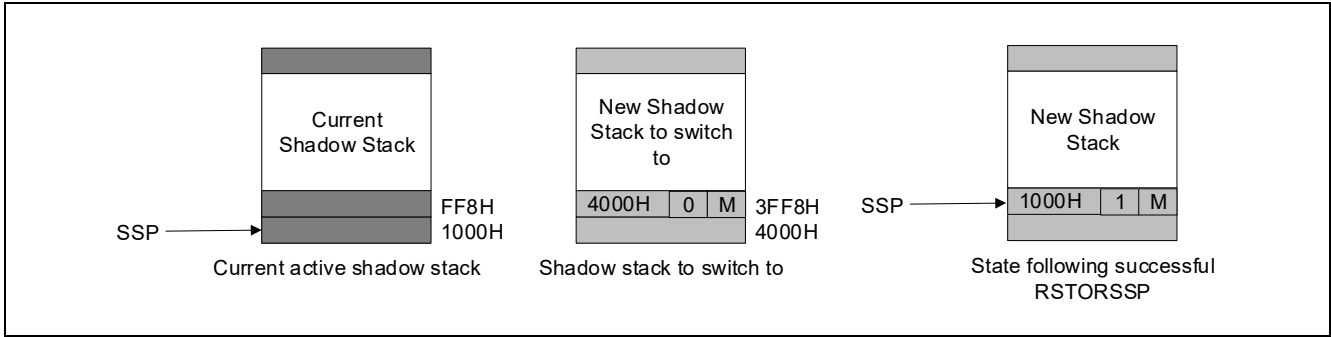


Figure 18-2. RSTORSSP to Switch to New Shadow Stack

In this example, the initial SSP is 1000H and the shadow-stack-restore token is on a new shadow stack at address 3FF8H. The token at address 3FF8H holds the SSP when this restore point was created; in this example it is 4000H.

In order to switch to the new shadow stack, the RSTORSSP instruction is invoked with the memory operand pointing set to 3FF8H. When the RSTORSSP instruction completes, the SSP is set to 3FF8H and the shadow-stack-restore token at 3FF8H is replaced by a previous-ssp token that holds the address 1000H, i.e., the old SSP.

The following figure illustrates the SAVEPREVSSP instruction operation during a shadow stack switching sequence.

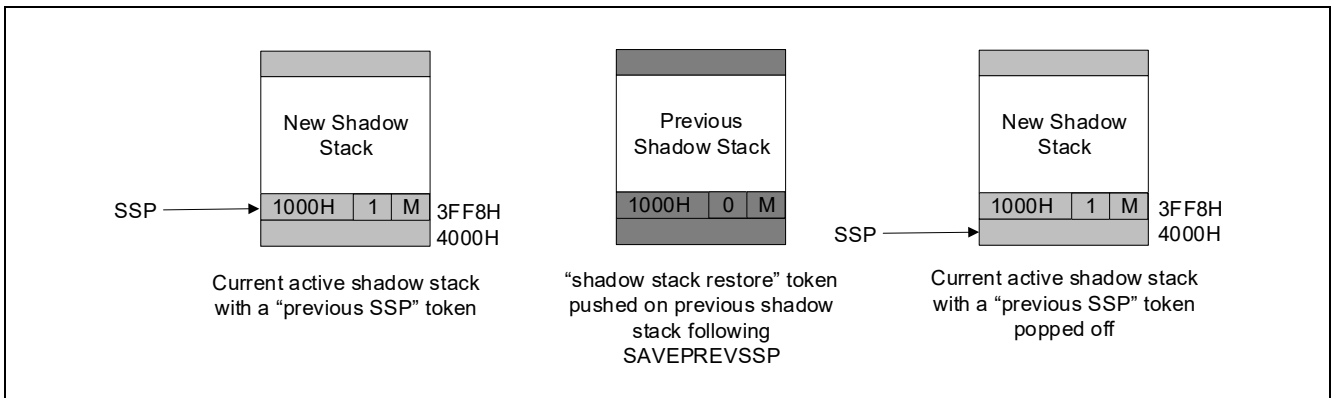


Figure 18-3. SAVEPREVSSP to Save a Restore Point

To allow switching back to this old shadow stack, a SAVEPREVSSP instruction is now invoked. The SAVEPREVSSP instruction does not take any memory operand and expects to find a previous-ssp token at the top of the shadow stack, i.e., at address 3FF8H. The SAVEPREVSSP instruction then saves a shadow-stack-restore token on the old shadow stack at address FF8H, and the token itself holds the address 1000H which is the address recorded in the previous-ssp token. The SAVEPREVSSP instruction also pops the previous-ssp token off the current shadow stack and thus the SSP following SAVEPREVSSP is 4000H.

Subsequently to switch back to the old shadow stack, a RSTORSSP instruction may be invoked with memory operand set to FF8H.

If, following a switch to a new shadow stack, it is not required to create a restore point on the old shadow stack, then the previous-ssp token created by the RSTORSSP instruction can be popped off the shadow stack by using the INCSSP instruction.

See the SAVEPREVSSP and RSTORSSP instruction operations for the detailed algorithm.

18.2.6 Constraining Execution at Targets of RET

Instructions at the target of a RET instruction will not execute, even speculatively, if the RET addresses (either from normal stack or shadow stack) are speculative-only or do not match, unless the target of the RET is also predicted (e.g., by a Return Stack Buffer prediction), when CET shadow stack is enabled. A RET address would be speculative-only if it was modified by an older speculative-only store, or was an older value than the most recent value stored to that address on the logical processor.

18.3 INDIRECT BRANCH TRACKING

When the indirect branch tracking feature is active, the indirect JMP/CALL instruction behavior changes as follows.

- **JMP:** If the next instruction retired after an indirect JMP is not an ENDBR32 instruction in legacy and compatibility mode, or ENDBR64 instruction in 64-bit mode, then a #CP fault is generated. Below JMP instructions are tracked to enforce an ENDBRANCH. Note that Jcc, RIP relative, and far direct JMP are not included as these have an offset encoded into the instruction and are not exploitable to create unintended control transfers.
 - JMP r/m16, JMP r/m32, JMP r/m64
 - JMP m16:16, JMP m16:32, JMP m16:64
- **CALL:** If the next instruction retired after an indirect CALL is not an ENDBR32 instruction in legacy and compatibility mode, or ENDBR64 in 64-bit mode, then a #CP fault is generated. Below CALL instructions are tracked to enforce an ENDBRANCH. Note that relative and zero displacement forms of CALL instructions are not included as these have an offset encoded into the instruction and are not exploitable to create unintended control transfers.
 - CALL r/m16, CALL r/m32, CALL r/m64
 - CALL m16:16, CALL m16:32, CALL m16:64

The ENDBR32 and ENDBR64 instructions will have the same effect as the NOP instruction on Intel 64 processors that do not support CET. On processors supporting CET, these instructions do not change register or flag state. This allows CET instrumented programs to execute on processors that do not support CET. Even when CET is supported and enabled, these NOP-like instructions do not affect the execution state of the program, do not cause any additional register pressure, and are minimally intrusive from power and performance perspectives.

The processor implements two dual-state machines to track indirect CALL/JMP for terminations. One state machine is maintained for user mode and one for supervisor mode. At reset the user and supervisor mode state machines are in IDLE state.

On instructions other than indirect CALL/JMP, the state machine stays in the IDLE state.

On an indirect CALL or JMP instruction, the state machine transitions to the WAIT_FOR_ENDBRANCH state.

In the WAIT_FOR_ENDBRANCH state, the indirect branch tracking state machine verifies the next instruction is an ENDBR32 instruction in legacy and compatibility mode, or ENDBR64 instruction in 64-bit mode, and either:

- Causes a #CP fault, or
- Allows the next instruction if legacy compatibility configuration allows (see Section 18.3.6).

The priority of the #CP(ENDBRANCH) exception relative to other events is as follows.

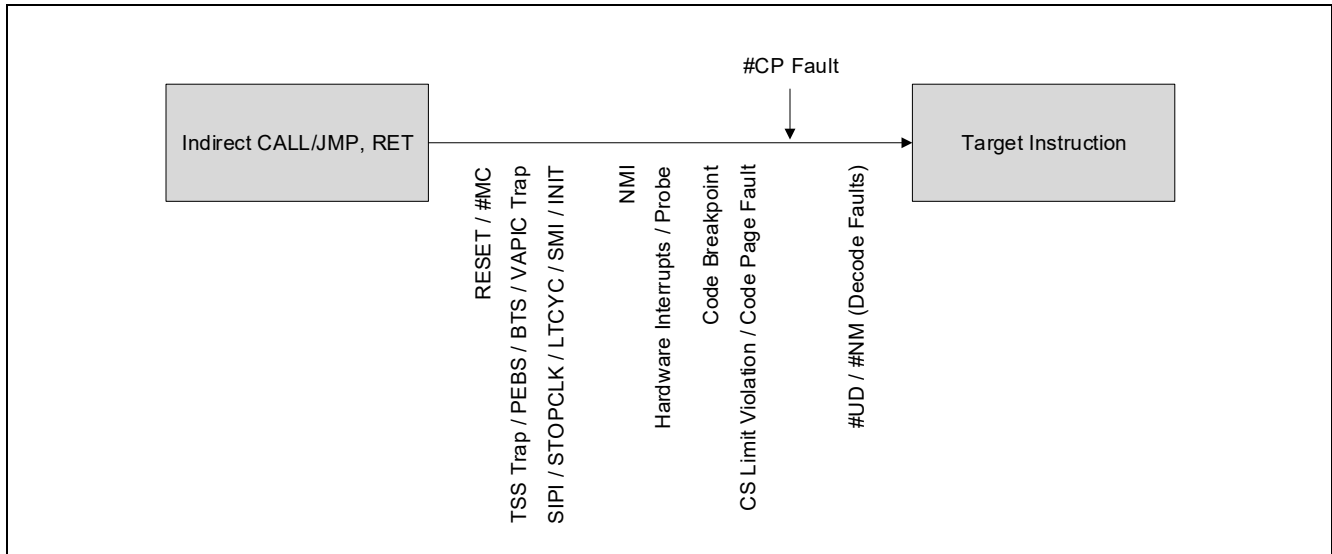


Figure 18-4. Priority of Control Protection Exception on Missing ENDBRANCH

Higher priority faults/traps/events that occur at the end of an indirect CALL/JMP are delivered ahead of any #CP(ENDBRANCH) fault. The CET state machine at the privilege level where the higher priority fault/trap/event occurred retains its state when the control transfers to the fault/trap/event handler. The instruction pointer pushed on the stack for a #CP(ENDBRANCH) fault is the address of the instruction at the target of the indirect CALL/JMP that caused the fault.

18.3.1 No-track Prefix for Near Indirect CALL/JMP

CET allows software to designate certain indirect CALL and JMP instructions as “non-tracked indirect control transfer instructions”. Software (e.g., compiler generated code for switch statements, jump tables, etc.) should use the no-track prefix only if they have generated code to validate the possible targets of this CALL/JMP to be legal targets. Software (e.g., compilers), when using the no-track prefix with CALL/JMP where an absolute offset is specified indirectly in a memory location, should ensure that such memory locations cannot be tampered. When enabled by setting the NO_TRACK_EN control in the IA32_U_CET/IA32_S_CET MSR, near indirect CALL and JMP instructions when prefixed with 3EH do not modify the CET indirect branch tracker. Far CALL and JMP instructions are always tracked and ignore the 3EH prefix. When this control is 0, near indirect CALL and JMP instructions are always tracked irrespective of the presence of the 3EH prefix.

In 64-bit mode, the 3EH prefix on an indirect CALL or JMP is recognized as a no-track prefix if there isn’t a 64H/65H prefix on the instruction.

In legacy/compatibility mode, the 3EH prefix on an indirect CALL or JMP is recognized as a no-track prefix when it is the last group 2 prefix on the instruction.

18.3.2 Terminology

The pseudocode for the instruction operations use a notation `EndbranchEnabled(CPL)` as a test of whether indirect branch tracking is enabled at the CPL. This term returns a TRUE or FALSE indication as follows.

`EndbranchEnabled(CPL):`

```

IF CR4.CET = 1 AND CRO.PE = 1 AND EFLAGS.VM = 0
  IF CPL = 3
    THEN
      (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL = 3 *)
      ENDBR_ENABLED = IA32_U_CET.ENDBR_EN;
    ELSE
      (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL < 3 *)
      ENDBR_ENABLED = IA32_S_CET.ENDBR_EN;
  FI;
  IF ENDBR_ENABLED = 1
    THEN
      return TRUE;
    ELSE
      return FALSE;
  FI;
ELSE
  (* Indirect branch tracking is not enabled in real mode and virtual-8086 mode or if the master CET feature
  enable in CR4 is disabled *)
  return FALSE;
ENDIF

```

Likewise the notation `EndbranchEnabledAndNotSuppressed` is defined as follows:

`EndbranchEnabledAndNotSuppressed(CPL):`

```

IF CR4.CET = 1 AND CRO.PE = 1 AND EFLAGS.VM = 0
  IF CPL = 3
    THEN
      (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL = 3 *)
      ENDBR_ENABLED = IA32_U_CET.ENDBR_EN;
      SUPPRESSED = IA32_U_CET.SUPPRESS;
    ELSE
      (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL < 3 *)
      ENDBR_ENABLED = IA32_S_CET.ENDBR_EN;
      SUPPRESSED = IA32_S_CET.SUPPRESS;
  FI;
  IF ENDBR_ENABLED = 1 AND SUPPRESSED = 0
    THEN
      return TRUE;
    ELSE
      return FALSE;
  FI;
ELSE
  (* Indirect branch tracking is not enabled in real mode and virtual-8086 mode or if the master CET feature
  enable in CR4 is disabled *)
  return FALSE;
ENDIF

```


18.3.3 Indirect Branch Tracking

The hardware implements two CET indirect branch tracker state machines, one for user mode (CPL == 3) and one for supervisor mode (CPL < 3). At any time, which of the CET indirect branch trackers is in the active state depends on the CPL of the machine. When a user space program is executing, the CPL 3 CET indirect branch tracker is active. When supervisor mode software is executing, the CPL < 3 tracker is active. This section describes the various control transfer conditions and the tracker state on those transfers.

18.3.3.1 Control Transfers between CPL 3 and CPL < 3

Some events and instructions can cause control transfer to occur from CPL 3 to CPL < 3, and vice versa. As part of the CPL change the hardware also switches the active CET indirect branch tracker. For example, when an interrupt occurs during execution of a user mode (CPL == 3) program and it causes the CPL to switch to supervisor mode (CPL < 3) then, as part of the CPL change, the user mode CET indirect branch tracker becomes inactive and the supervisor mode CET indirect branch tracker becomes active. A subsequent IRET is used by the interrupt handler to return to the interrupted user mode program. This IRET causes the processor to switch the CPL to user mode (CPL == 3) and, as part of the CPL change, the supervisor mode CET indirect branch tracker becomes inactive and the user mode CET indirect branch tracker becomes active.

The CPL where the event or instruction that caused the control transfer occurs is termed the source CPL, and the CET indirect branch tracker state at that CPL is referred here as the source CET indirect branch tracker state. The CPL reached at the end of the control transfer is termed the destination CPL, and the CET indirect branch tracker state at that CPL is referred to as the destination CET indirect branch tracker state.

This section describes various cases of control transfers that occur between user mode (CPL 3) and supervisor mode (CPL < 3).

In all these cases the source CET indirect branch tracker state becomes not active and retains its state (IDLE, WAIT_FOR_ENDBRANCH), and the target CET indirect branch tracker state becomes active if there was no fault during the transfer.

- Case 1: Far CALL/JMP, SYSCALL/SYSENTER
 - If indirect branch tracking is enabled, the target indirect branch tracker state becomes active and is unsuppressed and goes to WAIT_FOR_ENDBRANCH. This enforces that the subroutine invoked by a far CALL/JMP must begin with an ENDBRANCH.
- Case 2: Hardware interrupt/trap/exception/NMI/Software interrupt/Machine Checks
 - If indirect branch tracking is enabled, the target indirect branch tracker state becomes active and is unsuppressed and goes to WAIT_FOR_ENDBRANCH.
- Case 3: IRET/Far RET
 - If indirect branch tracking enabled, the target indirect branch tracker becomes active and keeps its state. If the user mode was interrupted by a higher priority event, like an interrupt at the end of the indirect CALL/JMP, then when an IRET or Far RET is used to return to the interrupted user mode program, the user mode indirect branch tracker retains its state and a #CP fault will occur if the next instruction decoded is not an ENDBR32/64 according to mode of machine.

18.3.3.2 Control Transfers within CPL 3 or CPL < 3

Some events and instructions can cause control transfer to occur within CPL 3 or CPL < 3. For such transfers since the CPL class does not change, the same indirect branch tracker is used at the beginning and end of the control transfer.

- Case 1: Far CALL/JMP, Near indirect CALL/JMPCALL/JMP
 - Far CALL/JMP: If indirect branch tracking is enabled, active indirect branch tracker is unsuppressed and goes to WAIT_FOR_ENDBRANCH.
 - Near indirect CALL/JMPCALL/JMP: If indirect branch tracking is enabled and not suppressed, active indirect branch tracker goes to WAIT_FOR_ENDBRANCH.
- Case 2: Hardware interrupt/trap/exception/NMI/Software interrupt/Machine Checks

- If indirect branch tracking is enabled, the active indirect branch tracker is unsuppressed and goes to WAIT_FOR_ENDBRANCH.
- Case 3: IRET
 - If indirect branch tracking is enabled, the active indirect branch tracker keeps its state.

18.3.4 Indirect Branch Tracking State Machine

The state machine is described by Table 18-1.

Table 18-1. Indirect Branch Tracking State Machine

Current State	Trigger	Next State
TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1	Instructions other than indirect CALL/JMP or 3EH prefixed near indirect CALL/JMP and NO_TRACK_EN=1	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	Indirect CALL/JMP without 3EH prefix Indirect CALL/JMP with 3EH prefix and NO_TRACK_EN=0 Far CALL/JMP	TRACKER=WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1
TRACKER= WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1	INT3/INT1	TRACKER= WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1
	ENDBRANCH instruction	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	Successful ENCLU[ERESUME]	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	Instructions other than ENDBRANCH, successful ENCLU[ERESUME] or INT3 or INT1	If legacy compatibility treatment is not enabled or if not allowed by legacy code page bitmap: <ul style="list-style-type: none"> ▪ No state change and deliver #CP (ENDBRANCH) If legacy compatibility treatment is enabled and transfer allowed by legacy code page bitmap: <ul style="list-style-type: none"> ▪ TRACKER=IDLE, SUPPRESS=ISUPPRESS_DIS, ENDBR_EN=1
TRACKER=x, SUPPRESS=x, ENDBR_EN=0	All instructions	TRACKER=x, SUPPRESS=x, ENDBR_EN=0
TRACKER=IDLE, SUPPRESS=1, ENDBR_EN=1	Far CALL/JMP, INTn/INT3/INTO	TRACKER=WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1
	ENDBRANCH instruction Successful ENCLU[ERESUME]	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	All other instructions including indirect CALL/JMP	TRACKER=IDLE, SUPPRESS=1, ENDBR_EN=1
TRACKER=1, SUPPRESS=1, ENDBR_EN=1 (This state cannot be reached by hardware and is disallowed as a valid state by WRMSR/XRSTORS/VM entry/VM exit)	NA	NA

18.3.5 INT3 Treatment

INT3 are treated special in the WAIT_FOR_ENDBRANCH state. Occurrence of INT3 do not move the tracker to IDLE but instead the #BP trap from the INT3 instructions respectively is delivered as a higher priority event than the #CP exception due to missing ENDBRANCH.

Inside an enclave, INT3 delivers a fault-class exception and thus does not require the CPL to be less than DPL in the IDT gate 3. Following opt-out entry, the instruction delivers #UD. Following opt-in entry, INT3 delivers #BP. The special treatment of INT3 in WAIT_FOR_ENDBRANCH state does not apply in enclave mode following opt-out entry.

18.3.6 Legacy Compatibility Treatment

ENDBRANCH legacy compatibility treatment allows a CET enabled program to be used with legacy software that was not compiled / instrumented with ENDBRANCH. A CET enabled program enters legacy compatibility treatment when all of the below conditions are met.

1. Legacy compatibility configuration is enabled in this CPL class by setting the LEG_IW_EN bit in IA32_U_CET/IA32_S_CET.
2. Control transfer is performed using an indirect CALL/JMP without no-track prefix to an instruction other than ENDBRANCH.
3. The legacy code page bitmap is setup to indicate that the target of the control transfer is a legacy code page.

The legacy code page bitmap is a data structure in program memory that is used by the hardware to determine if the code page to which a legacy transfer is being performed is allowed. The access rights for accessing the legacy code page bitmap is determined by the current privilege level (CPL). The legacy code page bitmap is expected to be setup as a read-only data structure.

When a matching ENDBRANCH instruction is not decoded at the target of an indirect CALL/JMP when required, the processor performs the below actions.

CET indirect branch tracking state machine violation event handler:

If LEG_IW_EN == 1

 LA = LIP;

 IF ENCLAVE_MODE == 1

 LA = LA - SECS.BASEADDR;

 ENDIF

(* Load byte from bitmap. Address-size attribute for this load is 64 bits if IA32_EFER.LMA is 1 and is 32 bits when IA32_EFER.LMA is 0 *)

 IF (IA32_EFER.LMA & CS.L) == 0

 BITMAP_BYTE = load 1 byte from address (BITMAP_BASE + LA[31:15])

 ELSE IF (CR4.LA57 == 0)

 BITMAP_BYTE = load 1 byte from address (BITMAP_BASE + LA[47:15])

 ELSE

 BITMAP_BYTE = load 1 byte from address (BITMAP_BASE + LA[56:15])

 FI;

 IF BITMAP_BYTE & (1 << LA[14:12]) == 0 then Deliver #CP(ENDBRANCH) fault

 IF CPL = 3

 IA32_U_CET.TRACKER = IDLE

 IA32_U_CET.SUPPRESS = IA32_U_CET.SUPPRESS_DIS == 0 ? 1 : 0

 ELSE

 IA32_S_CET.TRACKER = IDLE

 IA32_S_CET.SUPPRESS = IA32_S_CET.SUPPRESS_DIS == 0 ? 1 : 0

 ENDIF

 Restart the instruction (handle all arch. consistency around MOV SS state machines, STI etc.) without opening up interrupt/trap window.

ELSE

 Deliver #CP(ENDBRANCH) Fault

ENDIF

Faults/traps in pseudocode are delivered normally (e.g., #PF, EPT violation). On a fault, the active tracker holds the last value (WAIT_FOR_ENDBRANCH) and the address saved on the stack is the current IP (instruction that wasn't the ENDBRANCH).

The CET indirect branch tracking state machine is suppressed in legacy compatibility mode if the SUPPRESS_DIS control bit is 0.

Once the CET indirect branch tracking state machine has been suppressed, subsequent indirect CALL/JMP are not tracked for termination instruction.

Once CET indirect branch tracking has been suppressed, subsequent execution of ENDBRANCH instructions will do the following (see the ENDBR32 and ENDBR64 instructions in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A* for details).

```
IF EndbranchEnabled(CPL) == 0
    NOP
ELSE
    SUPPRESS = 0
    TRACKER = IDLE
ENDIF
```

18.3.6.1 Legacy Code Page Bitmap Format

The legacy code page bitmap is a flat bitmap whose linear address is pointed to by the EB_LEG_BITMAP_BASE. Each bit in the bitmap represents a 4K page in linear memory. If the bit is 1 it indicates that the corresponding code page is a legacy code page; else it is a CET-enabled code page.

The processor uses the linear address of the instruction to which legacy transfer was attempted to lookup the bitmap. Bits of the linear address used as index in the bitmap are as follows.

- In legacy and compatibility mode: Bits 31:12.
- In 64-bit mode (EFER.LMA=1 and CS.L=1): Bits 47:12.

18.3.7 Other Considerations

18.3.7.1 Intel® Transactional Synchronization Extensions (Intel® TSX) Interactions

The XBEGIN instruction encodes the relative offset to the abort handler and hence the fallback to the abort handler can be considered as a "direct" branch and the abort handler does not need to have an ENDBRANCH.

CET continues to enforce indirect CALL/JMP tracking within a transaction. Legacy compatibility treatment inside a transaction functions normally. If a transaction abort occurs then the processor sets the state of the indirect branch tracker to IDLE and not-suppressed.

18.3.7.2 #CP(ENDBRANCH) Priority w.r.t #NM and #UD

#NM, #UD and #CP(ENDBRANCH) are opcode based faults. However, #CP(ENDBRANCH) is in a higher priority class than #NM and #UD as CET architecturally requires an ENDBRANCH at target of indirect CALL/JMP.

18.3.7.3 #CP(ENDBRANCH) Priority w.r.t #BP and #DB

Debug Exceptions priority is as follows.

- Traps delivered before any #CP(ENDBRANCH) fault: Data breakpoint trap, IO breakpoint trap single step trap, task switch trap.
- Code Breakpoint fault detected before instruction decode and delivered before #CP(ENDBRANCH).
- General-detect (GD) exception condition fault: Lower priority than #CP(ENDBRANCH).

- On IRET back from #DB/#BP, the source indirect branch tracker becomes active if enabled and not suppressed. INT3 does not cause #CP(ENDBRANCH) to support debugger usage of replacing bytes of ENDBRANCH with INT3 to set breakpoints. INT3 at target of a CALL-JMP(indirect) cause #BP(INT3) instead of #CP(ENDBRANCH), #CP(ENDBRANCH) fault is delayed. #BP caused by INT3 treated like other events that are higher priority than CET fault. On IRET back from #BP the source indirect tracker becomes active if enabled and not suppressed.

18.3.8 Constraining Speculation after Missing ENDBRANCH

When the CET tracker is in the WAIT_FOR_ENDBRANCH state, instruction execution will be limited or blocked, even speculatively, if the next instruction is not an ENDBRANCH.

This means that when indirect branch tracking is enabled and not suppressed, the instructions at the target of a near indirect JMP/CALL without the no-track prefix will only speculatively execute if there is an ENDBRANCH at the target. This can constrain both attacker controlled prediction as well as attacker controlled jump redirection attacks on near indirect JMPs/CALLs by reducing the gadgets available to an attacker using these techniques. Early implementations of CET may limit the speculative execution to a small number of instructions (less than 8, with no more than 5 loads) past a missing ENDBRANCH, while later implementations will completely block the speculative execution of instructions after a missing ENDBRANCH.

This mechanism also limits or blocks speculation of the next sequential instructions after an indirect JMP or CALL, presuming the JMP/CALL puts the CET tracker into the WAIT_FOR_ENDBRANCH state and the next sequential instruction is not an ENDBRANCH.

18.4 INTEL® TRUSTED EXECUTION TECHNOLOGY (INTEL® TXT) INTERACTIONS

GETSEC[ENTERACCS] and GETSEC[SENDER] clear CR4.CET, and it is not restored when these instructions complete.

GETSEC[EXITAC] will cause #GP(0) fault if CR4.CET is set.

4. Updates to Chapter 2, Volume 2A

Change bars and green 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 include :

- Update to Table 2-41, “#UD Conditions Dependent on EVEX.b Context”.
- Update to exception tables 2-44, 2-45, 2-46, 2-47, 2-48, 2-49, 2-50, 2-51, 2-52, 2-53, 2-54, 2-55, 2-56, 2-57, 2-58, 2-59, 2-60, 2-61, and 2-62.

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).

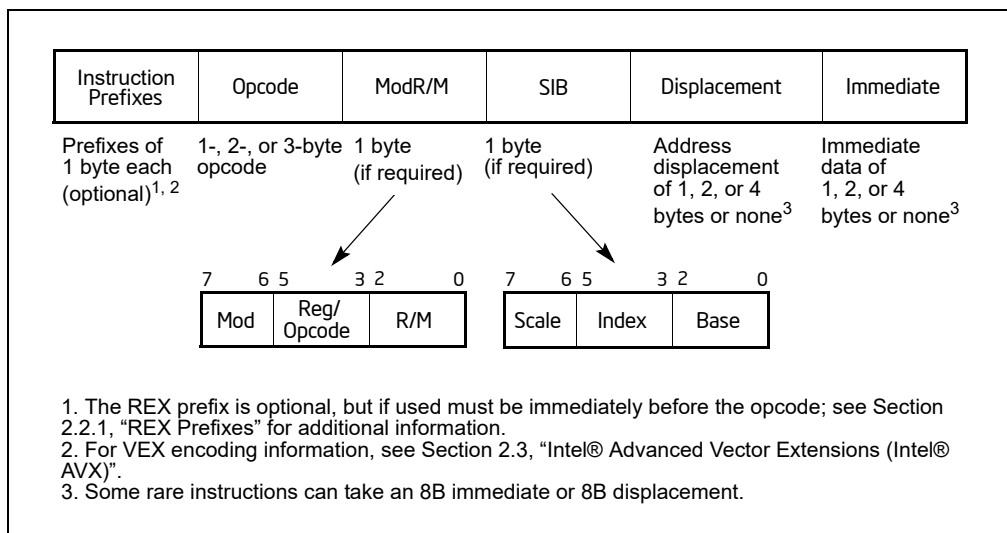


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

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/REPZ 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 POPCNT, LZCNT and ADOX instructions.

INSTRUCTION FORMAT

- 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 Chapter 17, “Intel® MPX,” 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 hints¹:
 - 2EH—Branch not taken (used only with Jcc instructions).
 - 3EH—Branch taken (used only with Jcc 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,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. Use these prefixes only with conditional branch instructions (Jcc). Other use of branch hint prefixes and/or other undefined opcodes with Intel 64 or IA-32 instructions is reserved; such use may cause unpredictable behavior.

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. Some earlier microarchitectures used these as branch hints, but recent generations have not and they are reserved for future hint usage.

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, CVTQ2PD 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.

	Mod	11	
	RM	000	
/digit (Opcode);	REG =	001	
	C8H	11001000	

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

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

			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	CL CX ECX	DL DX EDX	BL BX EBX	AH SP ESP	CH BP EBP	DH SI ESI	BH DI EDI		
	MM0 XMM0	MM1 XMM1	MM2 XMM2	MM3 XMM3	MM4 XMM4	MM5 XMM5	MM6 XMM6	MM7 XMM7		
	0 000	1 001	2 010	3 011	4 100	5 101	6 110	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:

- 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 prefix is necessary only if an instruction references one of the extended registers or uses a 64-bit operand. If a REX prefix is used when it has no meaning, it is ignored.

Only one REX prefix is allowed per instruction. If used, the REX prefix byte must immediately precede the opcode byte or the escape opcode byte (0FH). 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 be immediately preceding 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).

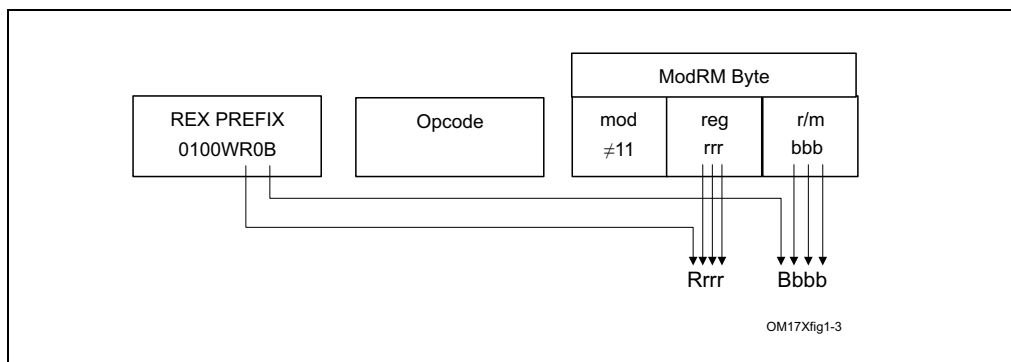
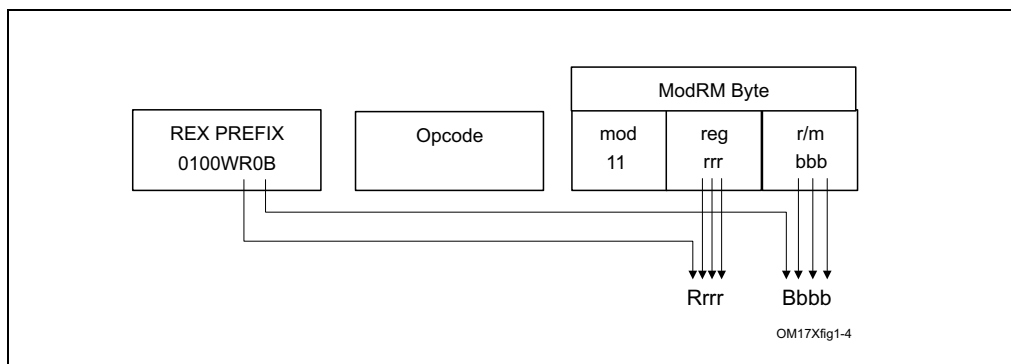
See Table 2-4 for a summary of the REX prefix format. Figure 2-4 through Figure 2-7 show examples of REX prefix fields in use. Some combinations of REX prefix fields are invalid. In such cases, the prefix is ignored. Some additional information follows:

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

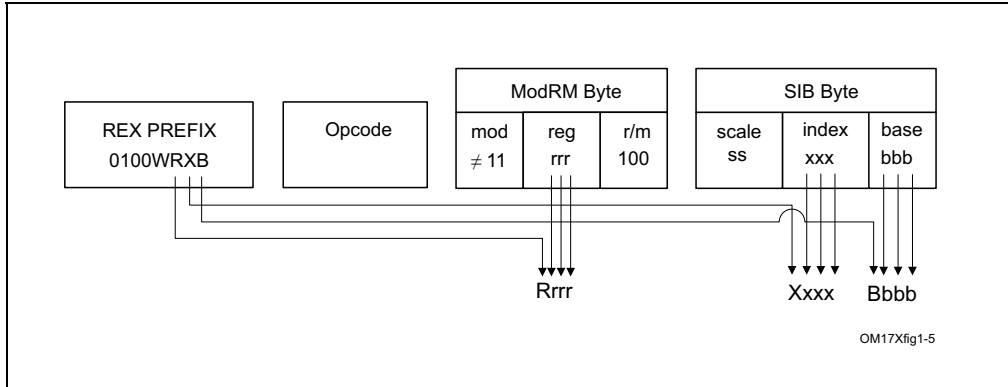


Figure 2-6. Memory Addressing With a SIB Byte

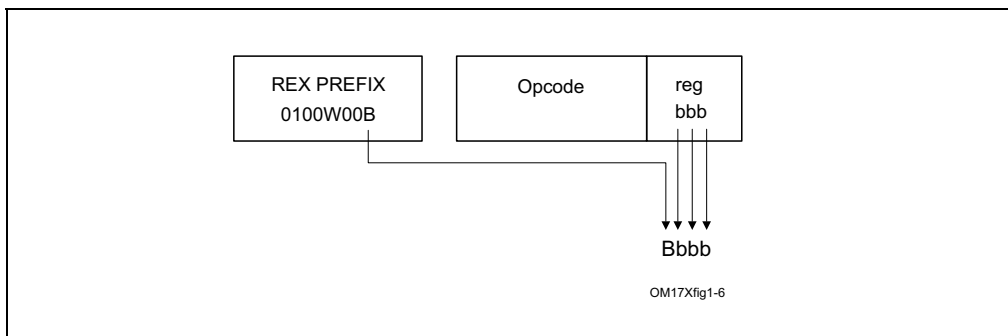


Figure 2-7. Register Operand Coded in Opcode Byte; REX.X & REX.R Not Used

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.
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 $\pm 2\text{GB}$ 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 MODRM.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.

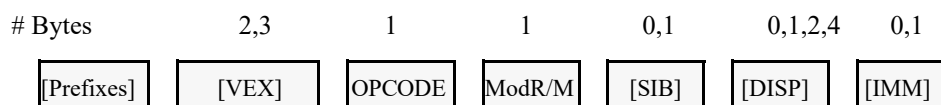


Figure 2-8. Instruction Encoding Format with VEX Prefix

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 preceding 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.

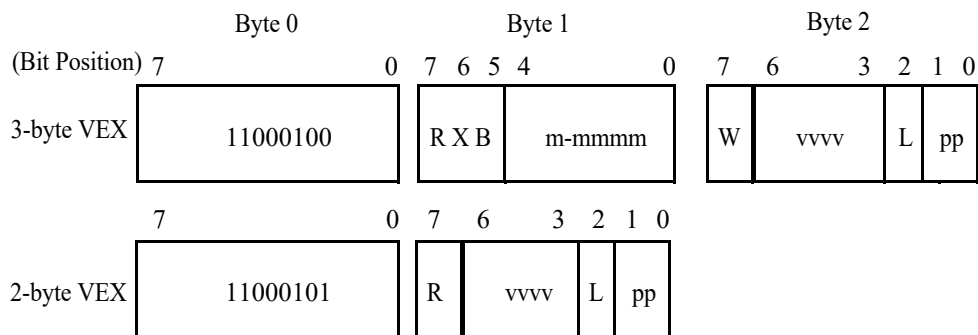
The VEX prefix is required to be the last prefix and immediately precedes the opcode bytes. It must follow any other prefixes. If VEX prefix is present a REX prefix is not supported.

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 2-8 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 2.5, “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
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

Opcode	Instruction mnemonic
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 the following table.

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 the following table.

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 MODRM, 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 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 AVX Instruction Length

The 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 64-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 2-3 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 2-3). 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 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
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 AVX AND SSE INSTRUCTION EXCEPTION SPECIFICATION

To look up the exceptions of legacy 128-bit SIMD instruction, 128-bit VEX-encoded instructions, and 256-bit VEX-encoded instruction, Table 2-14 summarizes the exception behavior into separate classes, with detailed exception conditions defined in sub-sections 2.4.1 through 2.5.1. For example, ADDPS contains the entry:

“See Exceptions Type 2”

In this entry, "Type2" can be looked up in Table 2-14.

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 22.25.3, "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)
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)MOVVDQA, (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*, (V)FMADD132PD, (V)FMADD213PD, (V)FMADD231PD, (V)FMADD132PS, (V)FMADD213PS, (V)FMADD231PS, (V)FMADDSUB132PD, (V)FMADDSUB213PD, (V)FMADDSUB231PD, (V)FMADDSUB132PS, (V)FMADDSUB213PS, (V)FMADDSUB231PS, (V)FMSUBADD132PD, (V)FMSUBADD213PD, (V)FMSUBADD231PD, (V)FMSUBADD132PS, (V)FMSUBADD213PS, (V)FMSUBADD231PS, (V)FMSUB132PD, (V)FMSUB213PD, (V)FMSUB231PD, (V)FMSUB132PS, (V)FMSUB213PS, (V)FMSUB231PS, (V)FNMADD132PD, (V)FNMADD213PD, (V)FNMADD231PD, (V)FNMADD132PS, (V)FNMADD213PS, (V)FNMADD231PS, (V)FNMMSUB132PD, (V)FNMMSUB213PD, (V)FNMMSUB231PD, (V)FNMMSUB132PS, (V)FNMMSUB213PS, (V)FNMMSUB231PS, (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)ADDS, (V)ADSS, (V)CMPD, (V)CMPS, (V)COMSD, (V)COMSS, (V)CVTPS2PD, (V)CVTSD2SI, (V)CVTSD2SS, (V)CVTSI2SD, (V)CVTSI2SS, (V)CVTSS2SD, (V)CVTSS2SI, (V)CVTSS2SI, (V)CVTTSD2SI, (V)CVTTSS2SI, (V)DIVSD, (V)DIVSS, (V)FMADD132SD, (V)FMADD213SD, (V)FMADD231SD, (V)FMADD132SS, (V)FMADD213SS, (V)FMADD231SS, (V)FMSUB132SD, (V)FMSUB213SD, (V)FMSUB231SD, (V)FMSUB132SS, (V)FMSUB213SS, (V)FMSUB231SS, (V)FNMADD132SD, (V)FNMADD213SD, (V)FNMADD231SD, (V)FNMADD132SS, (V)FNMADD213SS, (V)FNMADD231SS, (V)FNMMSUB132SD, (V)FNMMSUB213SD, (V)FNMMSUB231SD, (V)FNMMSUB132SS, (V)FNMMSUB213SS, (V)FNMMSUB231SS, (V)MAXSD, (V)MAXSS, (V)MINS, (V)MINS, (V)MULSD, (V)MULSS, (V)ROUNDSD, (V)ROUNDSS, (V)SQRTSD, (V)SQRTSS, (V)SUBSD, (V)SUBSS, (V)UCOMSD, (V)UCOMSS
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, (V)BLENDVPD, (V)BLENDVPS, (V)LDDQU***, (V)MASKMOVDQU, (V)PTEST, (V)PTESTPS, (V)PTESTPD, (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)PCMPQB, (V)PCMPQW, (V)PCMPQD, (V)PCMPQQ, (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)PMASB, (V)PMASW, (V)PMASD, (V)PMASUB, (V)PMASUW, (V)PMASUD, (V)PMINSB, (V)PMINSW, (V)PMINSD, (V)PMINUB, (V)PMINUW, (V)PMINUD, (V)PMULHUW, (V)PMULHRW, (V)PMULHW, (V)PMULLW, (V)PMULLD, (V)PMULUDQ, (V)PMULDQ, (V)POR, (V)PSADBW, (V)PSHUFB, (V)PSHUFD, (V)PSHUFW, (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, (V)VBLEND, (V)VPERMD, (V)VPERMPS, (V)VPERMPD, (V)VPERMQ, (V)VPSLLVD, (V)VPSLLVQ, (V)VPSRAVD, (V)VPSRLVD, (V)VPSRLVQ, (V)VPERMILPD, (V)VPERMILPS, (V)VPERM2F128
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, (V)PMOVSXBW, (V)RCPPS, (V)RSQRTSS, (V)PMOVSX/ZX, (V)LDMXCSR*, (V)STMXCSR
Type 6	VEXTRACTF128/VEXTRACTFxxx, VBROADCASTSS, VBROADCASTSD, VBROADCASTF128, VINSERTF128, VMASKMOVPS**, VMASKMOVPD**, VPMASKMOVD, VPMASKMOVQ, VBROADCASTI128, VPBROADCASTB, VPBROADCASTD, VPBROADCASTW, VPBROADCASTQ, VEXTRACTI128, VINSERTI128, VPERM21128
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 FORMAT

- (**) - 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.
- (***) - PCMPSTRM, PCMPSTRM, PCMPSTRM 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 AVX instructions. Within each class of exception conditions that are listed in Table 2-18 through Table 2-27, certain subsets of AVX instructions may be subject to #UD exception depending on the encoded value of the VEX.L field. Table 2-17 provides supplemental information of 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, VPBLEND, 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, VPMULHRW, VPMULHW/LW, VPMULLD, VPMULLDQ, 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/W/D/DQ, VPUNPCKHQDQ, VPUNPCKLBW/W/D/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, VPMOVMASKB, VPSLLDQ, VPSRLDQ, VPSLLW, VPSLLD, VPSLLQ, VPSRAW, VPSRAD, VPSRLW, VPSRLD, VPSRLQ	VMOVLHPS, VMOVHLPS
Type 8			
Type 11			
Type 12			

2.4.1 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 XCRO[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 (FOH).
			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.

2.4.2 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 XCRO[2:1] ≠ '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CRO.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 CRO.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.

2.4.3 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 XCRO[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 (FOH).
			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.

2.4.4 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 XCRO[2:1] ≠ '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CRO.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 CRO.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, PCMPSTRI, PCMPSTRM, PCMPISTRI, and PCMPISTRM instructions do not cause #GP if the memory operand is not aligned to 16-Byte boundary.

2.4.5 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 XCRO[2:1] ≠ '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	Legacy SSE instruction: If CRO.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 CRO.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.

2.4.6 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 XCRO[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 CRO.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.

2.4.7 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 XCRO[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.

2.4.8 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 XCRO[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.

2.4.9 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 XCRO[2:1] ≠ '11b'. If CR4.OSXSAVE[bit 18]=0.
	X	X	X	X	If preceded by a LOCK prefix (FOH).
			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.

2.4.10 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 XCRO[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'.
Device Not Available, #NM	X	X	X	X	If CR0.TS[bit 3]=1.
			X		For an illegal address in the SS segment.
Stack, #SS(0)				X	If a memory address referencing the SS segment is in a non-canonical form.
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
General Protection, #GP(0)				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 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.

2.5.1 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 GPR 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.

2.6 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.

INSTRUCTION FORMAT

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).

2.6.1 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.

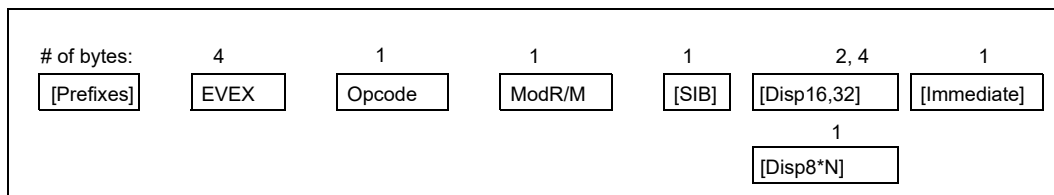


Figure 2-10. AVX-512 Instruction Format and the EVEX Prefix

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).

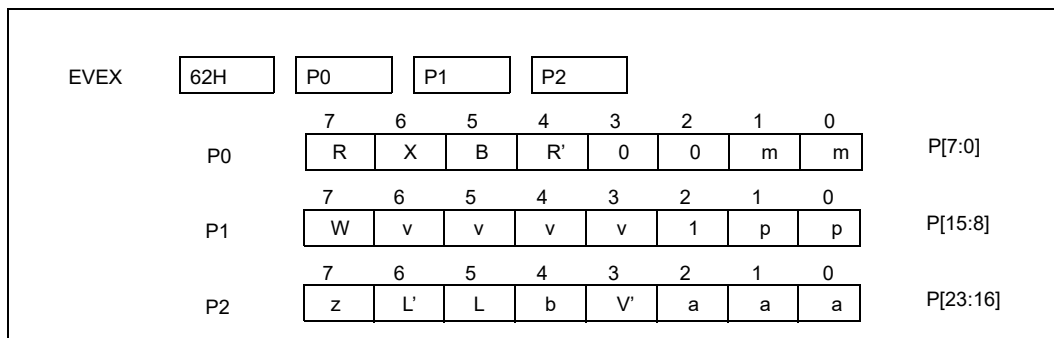


Figure 2-11. Bit Field Layout of the EVEX Prefix¹

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
--	Reserved	P[3 : 2]	Must be 0.
--	Fixed Value	P[10]	Must be 1.
EVEX.mm	Compressed legacy escape	P[1 : 0]	Identical to low two bits of VEX.mmmmm.
EVEX.pp	Compressed legacy prefix	P[9 : 8]	Identical to VEX.pp.
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.R'	High-16 register specifier modifier	P[4]	Combine with EVEX.R and ModR/M.reg. This bit is stored in inverted format.
EVEX.X	High-16 register specifier modifier	P[6]	Combine with EVEX.B and ModR/M.rm, when SIB/VSIB absent.
EVEX.vvvv	VVVV register specifier	P[14 : 11]	Same as VEX.vvvv. This field is encoded in bit inverted format.
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.aaa	Embedded opmask register specifier	P[18 : 16]	
EVEX.W	Osize promotion/Opcode extension	P[15]	
EVEX.z	Zeroing/Merging	P[23]	
EVEX.b	Broadcast/RC/SAE Context	P[20]	
EVEX.L'L	Vector length/RC	P[22 : 21]	

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

- Reserved bits: P[3:2] 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.
- 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.

2.6.2 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 2.6.3.

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	2nd Source 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

2.6.3 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 2.6.4).
- 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.

2.6.4 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.

2.6.5 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 2.6.11).

EVEX-encoded instruction that are pure load/store, and "Load+op" instruction semantic that operate on data element size less than 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	1 Tuple
	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
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

2.6.6 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.

2.6.7 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.

2.6.8 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.

2.6.9 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.

2.6.10 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

2.6.11 #UD Equations for EVEX

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

2.6.11.1 State Dependent #UD

In general, attempts to 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

2.6.11.2 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 : 2]	--	if > 0	if > 0
P[10]	--	if 0	if 0
P[1: 0]	EVEX.mm	if 00b	if 00b
P[7 : 6]	EVEX.RX	None (valid)	None (BOUND if EVEX.RX != 11b)

2.6.11.3 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
		Instruction supporting conditional processing mask 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 = 0xxx	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 KO.

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 specifies vector length, see Table 2-36, supports {sae} syntax.
3. L'L specifies vector length, see Table 2-36, supports embedded broadcast syntax
4. L'L specifies either vector length or ignored.

2.6.12 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.

2.6.13 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.

2.7 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

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

Exception Class	Instruction set	Mem arg	(#XM)
Type E5NF	Legacy-like Promotion	Varies, No fault suppression	No
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, VADDPs, VCMPPD, VCMPPS, VCVTDQ2PS, VCVTPD2DQ, VCVTPD2PS, VCVTPD2QQ, VCVTPD2UQQ, VCVTPD2UDQ, VCVTPS2DQ, VCVTPS2UDQs, VCVTQ2PD, VCVTQ2PS, VCVTTPD2DQ, VCVTTPD2QQ, VCVTTPD2UDQ, VCVTTPD2UQQ, VCVTTPS2DQ, VCVTTPS2UDQ, VCVTUDQ2PS, VCVTUQQ2PD, VCVTUQQ2PS, VDIVPD, VDIVPS, VEXP2PD, VEXP2PS, VFIXUPIMMPD, VFIXUPIMMPS, VFMADDxxxPD, VFMADDxxxPS, VFMADDSUBxxxPD, VFMADDSUBxxxPS, VFMSUBADDxxxPD, VFMSUBADDxxxPS, VFMSUBxxxPD, VFMSUBxxxPS, VFNMADDxxxPD, VFNMADDxxxPS, VFNMSUBxxxPD, VFNMSUBxxxPS, VGETEXPPD, VGETEXPPS, VGETMANTPD, VGETMANTPS, VMAXPD, VMAXPS, VMINPD, VMINPS, VMULPD, VMULPS, VRANGEPD, VRANGEPS, VREDUCEPD, VREDUCEPS, VRNDSCALEPD, VRNDSCALEPS, VRCP28PD, VRCP28PS, VRSQRT28PD, VRSQRT28PS, VSCALEFPD, VSCALEFPS, VSQRTPD, VSQRTPS, VSUBPD, VSUBPS
Type E3	VADDSd, VADDSs, VCMPSD, VCMPSs, VCVTPS2QQ, VCVTPS2UQQ, VCVTPS2PD, VCVTSD2SS, VCVTSS2SD, VCVTTPS2QQ, VCVTTPS2UQQ, VDIVSD, VDIVSS, VFMADDxxxSD, VFMADDxxxSS, VFMSUBxxxSD, VFMSUBxxxSS, VFNMADDxxxSD, VFNMADDxxxSS, VFNMSUBxxxSD, VFNMSUBxxxSS, VFIXUPIMMSD, VFIXUPIMMSS, VGETEXPSD, VGETEXPSS, VGETMANTSD, VGETMANTSS, VMAXSD, VMAXSS, VMINSD, VMINSS, VMULSD, VMULSS, VRANGESD, VRANGESS, VREDUCESD, VREDUCESS, VRNDSCALESD, VRNDSCALESS, VSCALEFSD, VSCALEFSS, VRCP28SD, VRCP28SS, VRSQRT28SD, VRSQRT28SS, VSQRSD, VSQRSS, VSUBSD, VSUBSS
Type E3NF	VCOMISD, VCOMISS, VCVTSD2SI, VCVTSD2USI, VCVTSI2SD, VCVTSI2SS, VCVTSS2SI, VCVTSS2USI, VCVTTSD2SI, VCVTTSD2USI, VCVTTSS2SI, VCVTTSS2USI, VCVTUSI2SD, VCVTUSI2SS, VUCOMISD, VUCOMISS
Type E4	VANDPD, VANDPS, VANDNPD, VANDNPS, VBLENDMPD, VBLENDMPS, VFPCLASSPD, VFPCLASSPS, VORPD, VORPS, VPABSD, VPABSQ, VPADD, 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, VRSQRT14PD, VRSQRT14PS, VXORPD, VXORPS

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

Exception Class	Instruction
E4.nb ²	VCOMPRESSPD, VCOMPRESSPS, VEXPANDPD, VEXPANDPS, VMOVDQU8, VMOVDQU16, VMOVDQU32, VMOVDQU64, VMOVUPD, VMOVUPS, VPABSB, VPABSW, VPADDB, VPADDW, VPADDSB, VPADDSW, VPADDUSB, VPADDUSW, VPAVGB, VPAVGW, VPCMPB, VPCMPQB, VPCMPQW, 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, VSHUFFD, VSHUFFPS, 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, PMOVZXBW, PMOVZXBQ, PMOVZXWD, PMOVZXWQ, PMOVZXDQ, VCVTDQ2PD, VCVTUDQ2PD, VPMOVsx _{xx} , VPMOVz _{xx}
Type E5NF	VMOVDDUP
Type E6	VBROADCASTF32X2, VBROADCASTF32X4, VBROADCASTF64X2, VBROADCASTF32X8, VBROADCASTF64X4, VBROADCASTI32X2, VBROADCASTI32X4, VBROADCASTI64X2, VBROADCASTI32X8, VBROADCASTI64X4, VBROADCASTSD, VBROADCASTSS, VFPCCLASSSD, VFPCCLASSSS, VPBROADCASTB, VPBROADCASTD, VPBROADCASTW, VPBROADCASTQ, VPMOVQB, VPMOVSQB, VPMOVUSQB, VPMOVQW, VPMOVSQW, VPMOVUSQW, VPMOVQD, VPMOVSQD, VPMOVUSQD, VPMOVDB, VPMOVSD, 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	VMOVHLP, 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, VPEXTRB, VPEXTRD, VPEXTRW, VPEXTRQ, VPINSRB, VPINSRD, VPINSRW, VPINSRQ
Type E10	VMOVSD, VMOVSS, VRCP14SD, VRCP14SS, VRSQRT14SD, VRSQRT14SS
Type E10NF	(VCVTISI2SD, VCVTUSI2SD) ⁶
Type E11	VCVTPH2PS, VCVTPS2PH
Type E12	VGATHERDPS, VGATHERDPD, VGATHERQPS, VGATHERQPD, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ, VPSCATTERDD, VPSCATTERDQ, VPSCATTERQD, VPSCATTERQQ, VSCATTERDPD, VSCATTERDPS, VSCATTERQPD, VSCATTERQPS
Type E12NP	VGATHERPFODPD, VGATHERPFODPS, VGATHERPFOQPD, VGATHERPFOQPS, VGATHERPF1DPD, VGATHERPF1DPS, VGATHERPF1QPD, VGATHERPF1QPS, VSCATTERPFODPD, VSCATTERPFODPS, VSCATTERPFOQPD, VSCATTERPFOQPS, 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.

2.7.1 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: <ul style="list-style-type: none"> ▪ 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: <ul style="list-style-type: none"> ▪ 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.

2.7.2 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: <ul style="list-style-type: none"> ▪ 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.

2.7.3 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: <ul style="list-style-type: none"> 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.OSXMMEXCPT[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: <ul style="list-style-type: none"> ▪ 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.

2.7.4 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: <ul style="list-style-type: none"> 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 (FOH).
			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: <ul style="list-style-type: none"> ▪ 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.

2.7.5 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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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.

2.7.6 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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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.

2.7.7 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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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.

2.7.8 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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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.

2.7.9 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: <ul style="list-style-type: none"> 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 (FOH).
			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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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.

2.7.10 Exception 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: <ul style="list-style-type: none"> 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 (FOH).
			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.

2.7.11 Exception 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: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> ▪ 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 (FOH).
			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.

2.8 EXCEPTION CLASSIFICATIONS OF OPMASK INSTRUCTIONS

The exception behavior of VEX-encoded opmask instructions are listed below.

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: <ul style="list-style-type: none"> ▪ 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.

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: <ul style="list-style-type: none"> ▪ 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.

5. Updates to Chapter 3, Volume 2A

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

Changes to this chapter include:

- Update to CALL description.
- Update to INCSSPD/INCSSPQ pseudocode.
- Update to JMP description.
- Update to KADDW/KADDB/KADDQ/KADDD to add intrinsics and a flags affected line.
- Update to LOOP/LOOPcc pseudocode.
- Typo corrections in LDS/LES/LFS/LGS/LSS pseudocode.
- Typo corrections in LSL pseudocode.

CALL—Call Procedure

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
E8 <i>cw</i>	CALL <i>rel16</i>	D	N.S.	Valid	Call near, relative, displacement relative to next instruction.
E8 <i>cd</i>	CALL <i>rel32</i>	D	Valid	Valid	Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.
FF <i>12</i>	CALL <i>r/m16</i>	M	N.E.	Valid	Call near, absolute indirect, address given in <i>r/m16</i> .
FF <i>12</i>	CALL <i>r/m32</i>	M	N.E.	Valid	Call near, absolute indirect, address given in <i>r/m32</i> .
FF <i>12</i>	CALL <i>r/m64</i>	M	Valid	N.E.	Call near, absolute indirect, address given in <i>r/m64</i> .
9A <i>cd</i>	CALL <i>ptr16:16</i>	D	Invalid	Valid	Call far, absolute, address given in operand.
9A <i>cp</i>	CALL <i>ptr16:32</i>	D	Invalid	Valid	Call far, absolute, address given in operand.
FF <i>13</i>	CALL <i>m16:16</i>	M	Valid	Valid	Call far, absolute indirect address given in <i>m16:16</i> . In 32-bit mode: if selector points to a gate, then RIP = 32-bit zero extended displacement taken from gate; else RIP = zero extended 16-bit offset from far pointer referenced in the instruction.
FF <i>13</i>	CALL <i>m16:32</i>	M	Valid	Valid	In 64-bit mode: If selector points to a gate, then RIP = 64-bit displacement taken from gate; else RIP = zero extended 32-bit offset from far pointer referenced in the instruction.
REX.W FF <i>13</i>	CALL <i>m16:64</i>	M	Valid	N.E.	In 64-bit mode: If selector points to a gate, then RIP = 64-bit displacement taken from gate; else RIP = 64-bit offset from far pointer referenced in the instruction.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
D	Offset	NA	NA	NA
M	ModRM:r/m (<i>r</i>)	NA	NA	NA

Description

Saves procedure linking information on the stack and branches to the called procedure specified using the target operand. The target operand specifies the address of the first instruction in the called procedure. The operand can be an immediate value, a general-purpose register, or a memory location.

This instruction can be used to execute four types of calls:

- **Near Call** — A call to a procedure in the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intra-segment call.
- **Far Call** — A call to a procedure located in a different segment than the current code segment, sometimes referred to as an inter-segment call.
- **Inter-privilege-level far call** — A far call to a procedure in a segment at a different privilege level than that of the currently executing program or procedure.
- **Task switch** — A call to a procedure located in a different task.

The latter two call types (inter-privilege-level call and task switch) can only be executed in protected mode. See “Calling Procedures Using Call and RET” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*, for additional information on near, far, and inter-privilege-level calls. See Chapter 7, “Task Management,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*, for information on performing task switches with the CALL instruction.

Near Call. When executing a near call, the processor pushes the value of the EIP register (which contains the offset of the instruction following the CALL instruction) on the stack (for use later as a return-instruction pointer). The processor then branches to the address in the current code segment specified by the target operand. The target operand specifies either an absolute offset in the code segment (an offset from the base of the code segment) or a relative offset (a signed displacement relative to the current value of the instruction pointer in the EIP register; this value points to the instruction following the CALL instruction). The CS register is not changed on near calls.

For a near call absolute, an absolute offset is specified indirectly in a general-purpose register or a memory location (*r/m16*, *r/m32*, or *r/m64*). The operand-size attribute determines the size of the target operand (16, 32 or 64 bits). When in 64-bit mode, the operand size for near call (and all near branches) is forced to 64-bits. Absolute offsets are loaded directly into the EIP(RIP) register. If the operand size attribute is 16, the upper two bytes of the EIP register are cleared, resulting in a maximum instruction pointer size of 16 bits. When accessing an absolute offset indirectly using the stack pointer [ESP] as the base register, the base value used is the value of the ESP before the instruction executes.

A relative offset (*rel16* or *rel32*) is generally specified as a label in assembly code. But at the machine code level, it is encoded as a signed, 16- or 32-bit immediate value. This value is added to the value in the EIP(RIP) register. In 64-bit mode the relative offset is always a 32-bit immediate value which is sign extended to 64-bits before it is added to the value in the RIP register for the target calculation. As with absolute offsets, the operand-size attribute determines the size of the target operand (16, 32, or 64 bits). In 64-bit mode the target operand will always be 64-bits because the operand size is forced to 64-bits for near branches.

Far Calls in Real-Address or Virtual-8086 Mode. When executing a far call in real-address or virtual-8086 mode, the processor pushes the current value of both the CS and EIP registers on the stack for use as a return-instruction pointer. The processor then performs a “far branch” to the code segment and offset specified with the target operand for the called procedure. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). With the pointer method, the segment and offset of the called procedure is encoded in the instruction using a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address immediate. With the indirect method, the target operand specifies a memory location that contains a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address. The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The far address is loaded directly into the CS and EIP registers. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared.

Far Calls in Protected Mode. When the processor is operating in protected mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level
- Far call to a different privilege level (inter-privilege level call)
- Task switch (far call to another task)

In protected mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate, task gate, or TSS) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in protected mode is very similar to one carried out in real-address or virtual-8086 mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register; the offset from the instruction is loaded into the EIP register.

A call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. Using this mechanism provides an extra level of indirection and is the preferred method of making calls between 16-bit and 32-bit code segments.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a call gate. The segment selector specified by the target operand identifies the call gate. The target operand can specify the call gate segment selector either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch. (Note that when using a call gate to perform a far call to a segment at the same privilege level, no stack switch occurs.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack, an optional set of parameters from the calling procedure's stack, and the segment selector and instruction pointer for the calling procedure's code segment. (A value in the call gate descriptor determines how many parameters to copy to the new stack.) Finally, the processor branches to the address of the procedure being called within the new code segment.

Executing a task switch with the CALL instruction is similar to executing a call through a call gate. The target operand specifies the segment selector of the task gate for the new task activated by the switch (the offset in the target operand is ignored). The task gate in turn points to the TSS for the new task, which contains the segment selectors for the task's code and stack segments. Note that the TSS also contains the EIP value for the next instruction that was to be executed before the calling task was suspended. This instruction pointer value is loaded into the EIP register to re-start the calling task.

The CALL instruction can also specify the segment selector of the TSS directly, which eliminates the indirection of the task gate. See Chapter 7, "Task Management," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for information on the mechanics of a task switch.

When you execute a task switch with a CALL instruction, the nested task flag (NT) is set in the EFLAGS register and the new TSS's previous task link field is loaded with the old task's TSS selector. Code is expected to suspend this nested task by executing an IRET instruction which, because the NT flag is set, automatically uses the previous task link to return to the calling task. (See "Task Linking" in Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for information on nested tasks.) Switching tasks with the CALL instruction differs in this regard from JMP instruction. JMP does not set the NT flag and therefore does not expect an IRET instruction to suspend the task.

Mixing 16-Bit and 32-Bit Calls. When making far calls between 16-bit and 32-bit code segments, use a call gate. If the far call is from a 32-bit code segment to a 16-bit code segment, the call should be made from the first 64 KBytes of the 32-bit code segment. This is because the operand-size attribute of the instruction is set to 16, so only a 16-bit return address offset can be saved. Also, the call should be made using a 16-bit call gate so that 16-bit values can be pushed on the stack. See Chapter 21, "Mixing 16-Bit and 32-Bit Code," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, for more information.

Far Calls in Compatibility Mode. When the processor is operating in compatibility mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level, remaining in compatibility mode
- Far call to the same privilege level, transitioning to 64-bit mode
- Far call to a different privilege level (inter-privilege level call), transitioning to 64-bit mode

Note that a CALL instruction can not be used to cause a task switch in compatibility mode since task switches are not supported in IA-32e mode.

In compatibility mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in compatibility mode is very similar to one carried out in protected mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register and the offset from the instruction is loaded into the EIP register. The difference is that 64-bit mode may be entered. This is specified by the L bit in the new code segment descriptor.

Note that a 64-bit call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. However, using this mechanism requires that the target code segment descriptor have the L bit set, causing an entry to 64-bit mode.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a 64-bit call gate. The segment selector specified by the target operand identifies the call gate. The target

operand can specify the call gate segment selector either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the 16-byte call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is set to NULL. The new stack pointer is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch. (Note that when using a call gate to perform a far call to a segment at the same privilege level, an implicit stack switch occurs as a result of entering 64-bit mode. The SS selector is unchanged, but stack segment accesses use a segment base of 0x0, the limit is ignored, and the default stack size is 64-bits. The full value of RSP is used for the offset, of which the upper 32-bits are undefined.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack and the segment selector and instruction pointer for the calling procedure's code segment. (Parameter copy is not supported in IA-32e mode.) Finally, the processor branches to the address of the procedure being called within the new code segment.

Near(Far) Calls in 64-bit Mode. When the processor is operating in 64-bit mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level, transitioning to compatibility mode
- Far call to the same privilege level, remaining in 64-bit mode
- Far call to a different privilege level (inter-privilege level call), remaining in 64-bit mode

Note that in this mode the CALL instruction can not be used to cause a task switch in 64-bit mode since task switches are not supported in IA-32e mode.

In 64-bit mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in 64-bit mode is very similar to one carried out in compatibility mode. The target operand specifies an absolute far address indirectly with a memory location (*m16:16*, *m16:32* or *m16:64*). The form of CALL with a direct specification of absolute far address is not defined in 64-bit mode. The operand-size attribute determines the size of the offset (16, 32, or 64 bits) in the far address. The new code segment selector and its descriptor are loaded into the CS register; the offset from the instruction is loaded into the EIP register. The new code segment may specify entry either into compatibility or 64-bit mode, based on the L bit value.

A 64-bit call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. However, using this mechanism requires that the target code segment descriptor have the L bit set.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a 64-bit call gate. The segment selector specified by the target operand identifies the call gate. The target operand can only specify the call gate segment selector indirectly with a memory location (*m16:16*, *m16:32* or *m16:64*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the 16-byte call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is set to NULL. The new stack pointer is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch.

Note that when using a call gate to perform a far call to a segment at the same privilege level, an implicit stack switch occurs as a result of entering 64-bit mode. The SS selector is unchanged, but stack segment accesses use a segment base of 0x0, the limit is ignored, and the default stack size is 64-bits. (The full value of RSP is used for the offset.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack and the segment selector and instruction pointer for the calling procedure's code segment. (Parameter copy is not supported in IA-32e mode.) Finally, the processor branches to the address of the procedure being called within the new code segment.

Refer to Chapter 6, “Procedure Calls, Interrupts, and Exceptions” and Chapter 18, “Control-Flow Enforcement Technology (CET)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* for CET details.

Instruction ordering. Instructions following a far call may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the far call have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Instructions sequentially following a near indirect CALL instruction (i.e., those not at the target) may be executed speculatively. If software needs to prevent this (e.g., in order to prevent a speculative execution side channel), then an LFENCE instruction opcode can be placed after the near indirect CALL in order to block speculative execution.

Operation

```

IF near call
  THEN IF near relative call
    THEN
      IF OperandSize = 64
        THEN
          tempDEST := SignExtend(DEST); (* DEST is rel32 *)
          tempRIP := RIP + tempDEST;
          IF stack not large enough for a 8-byte return address
            THEN #SS(0); FI;
          Push(RIP);
          IF ShadowStackEnabled(CPL) AND DEST != 0
            ShadowStackPush8B(RIP);
          FI;
          RIP := tempRIP;
        FI;
      IF OperandSize = 32
        THEN
          tempEIP := EIP + DEST; (* DEST is rel32 *)
          IF tempEIP is not within code segment limit THEN #GP(0); FI;
          IF stack not large enough for a 4-byte return address
            THEN #SS(0); FI;
          Push(EIP);
          IF ShadowStackEnabled(CPL) AND DEST != 0
            ShadowStackPush4B(EIP);
          FI;
          EIP := tempEIP;
        FI;
      IF OperandSize = 16
        THEN
          tempEIP := (EIP + DEST) AND 0000FFFFH; (* DEST is rel16 *)
          IF tempEIP is not within code segment limit THEN #GP(0); FI;
          IF stack not large enough for a 2-byte return address
            THEN #SS(0); FI;
          Push(IP);
          IF ShadowStackEnabled(CPL) AND DEST != 0
            (* IP is zero extended and pushed as a 32 bit value on shadow stack *)
            ShadowStackPush4B(IP);
          FI;
          EIP := tempEIP;
        FI;
      ELSE (* Near absolute call *)

```



```

IF OperandSize = 64
  THEN
    tempRIP := DEST; (* DEST is r/m64 *)
    IF stack not large enough for a 8-byte return address
      THEN #SS(0); FI;
    Push(RIP);
    IF ShadowStackEnabled(CPL)
      ShadowStackPush8B(RIP);
    FI;
    RIP := tempRIP;
FI;
IF OperandSize = 32
  THEN
    tempEIP := DEST; (* DEST is r/m32 *)
    IF tempEIP is not within code segment limit THEN #GP(0); FI;
    IF stack not large enough for a 4-byte return address
      THEN #SS(0); FI;
    Push(EIP);
    IF ShadowStackEnabled(CPL)
      ShadowStackPush4B(EIP);
    FI;
    EIP := tempEIP;
FI;
IF OperandSize = 16
  THEN
    tempEIP := DEST AND 0000FFFFH; (* DEST is r/m16 *)
    IF tempEIP is not within code segment limit THEN #GP(0); FI;
    IF stack not large enough for a 2-byte return address
      THEN #SS(0); FI;
    Push(IP);
    IF ShadowStackEnabled(CPL)
      (* IP is zero extended and pushed as a 32 bit value on shadow stack *)
      ShadowStackPush4B(IP);
    FI;
    EIP := tempEIP;
FI;
FI;rel/abs
IF (Call near indirect, absolute indirect)
  IF EndbranchEnabledAndNotSuppressed(CPL)
    IF CPL = 3
      THEN
        IF ( no 3EH prefix OR IA32_U_CET.NO_TRACK_EN == 0 )
          THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
          FI;
        ELSE
          IF ( no 3EH prefix OR IA32_S_CET.NO_TRACK_EN == 0 )
            THEN
              IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            FI;
          FI;
        FI;
      FI;
    FI;
  FI;
FI; near

```

IF far call and (PE = 0 or (PE = 1 and VM = 1)) (* Real-address or virtual-8086 mode *)

THEN

IF OperandSize = 32

THEN

IF stack not large enough for a 6-byte return address

THEN #SS(0); FI;

IF DEST[31:16] is not zero THEN #GP(0); FI;

Push(CS); (* Padded with 16 high-order bits *)

Push(EIP);

CS := DEST[47:32]; (* DEST is *ptr16:32* or [*m16:32*] *)

EIP := DEST[31:0]; (* DEST is *ptr16:32* or [*m16:32*] *)

ELSE (* OperandSize = 16 *)

IF stack not large enough for a 4-byte return address

THEN #SS(0); FI;

Push(CS);

Push(IP);

CS := DEST[31:16]; (* DEST is *ptr16:16* or [*m16:16*] *)

EIP := DEST[15:0]; (* DEST is *ptr16:16* or [*m16:16*]; clear upper 16 bits *)

FI;

FI;

IF far call and (PE = 1 and VM = 0) (* Protected mode or IA-32e Mode, not virtual-8086 mode*)

THEN

IF segment selector in target operand NULL

THEN #GP(0); FI;

IF segment selector index not within descriptor table limits

THEN #GP(new code segment); FI;

Read type and access rights of selected segment descriptor;

IF IA32_EFER.LMA = 0

THEN

IF segment type is not a conforming or nonconforming code segment, call gate, task gate, or TSS

THEN #GP(segment selector); FI;

ELSE

IF segment type is not a conforming or nonconforming code segment or 64-bit call gate,

THEN #GP(segment selector); FI;

FI;

Depending on type and access rights:

GO TO CONFORMING-CODE-SEGMENT;

GO TO NONCONFORMING-CODE-SEGMENT;

GO TO CALL-GATE;

GO TO TASK-GATE;

GO TO TASK-STATE-SEGMENT;

FI;

CONFORMING-CODE-SEGMENT:

IF L bit = 1 and D bit = 1 and IA32_EFER.LMA = 1

THEN GP(new code segment); FI;

IF DPL > CPL

THEN #GP(new code segment); FI;

IF segment not present

THEN #NP(new code segment); FI;

```

IF stack not large enough for return address
    THEN #SS(0); FI;
tempEIP := DEST(Offset);
IF target mode = Compatibility mode
    THEN tempEIP := tempEIP AND 00000000_FFFFFFFFH; FI;
IF OperandSize = 16
    THEN
        tempEIP := tempEIP AND 0000FFFFH; FI; (* Clear upper 16 bits *)
IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode) and (tempEIP outside new code segment limit)
    THEN #GP(0); FI;
IF tempEIP is non-canonical
    THEN #GP(0); FI;
IF ShadowStackEnabled(CPL)
    IF OperandSize = 32
        THEN
            tempPushLIP = CSBASE + EIP;
        ELSE
            IF OperandSize = 16
                THEN
                    tempPushLIP = CSBASE + IP;
                ELSE (* OperandSize = 64 *)
                    tempPushLIP = RIP;
            FI;
        FI;
    tempPushCS = CS;
FI;
IF OperandSize = 32
    THEN
        Push(CS); (* Padded with 16 high-order bits *)
        Push(EIP);
        CS := DEST(CodeSegmentSelector);
        (* Segment descriptor information also loaded *)
        CS(RPL) := CPL;
        EIP := tempEIP;
    ELSE
        IF OperandSize = 16
            THEN
                Push(CS);
                Push(IP);
                CS := DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) := CPL;
                EIP := tempEIP;
            ELSE (* OperandSize = 64 *)
                Push(CS); (* Padded with 48 high-order bits *)
                Push(RIP);
                CS := DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) := CPL;
                RIP := tempEIP;
            FI;
        FI;
IF ShadowStackEnabled(CPL)
    IF (IA32_EFER.LMA and DEST(CodeSegmentSelector).L) = 0

```

```

    (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
    IF (SSP & 0xFFFFFFFF00000000 != 0)
        THEN #GP(0); FI;
    FI;
    (* align to 8 byte boundary if not already aligned *)
    tempSSP = SSP;
    Shadow_stack_store 4 bytes of 0 to (SSP - 4)
    SSP = SSP & 0xFFFFFFFFFFFFFFF8H
    ShadowStackPush8B(tempPushCS); (* Padded with 48 high-order bits of 0 *)
    ShadowStackPush8B(tempPushLIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
    ShadowStackPush8B(tempSSP);
    FI;
    IF EndbranchEnabled(CPL)
        IF CPL = 3
            THEN
                IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
                IA32_U_CET.SUPPRESS = 0
            ELSE
                IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
                IA32_S_CET.SUPPRESS = 0
            FI;
    FI;
    END;

NONCONFORMING-CODE-SEGMENT:
    IF L-Bit = 1 and D-BIT = 1 and IA32_EFER.LMA = 1
        THEN GP(new code segment selector); FI;
    IF (RPL > CPL) or (DPL ≠ CPL)
        THEN #GP(new code segment selector); FI;
    IF segment not present
        THEN #NP(new code segment selector); FI;
    IF stack not large enough for return address
        THEN #SS(0); FI;
    tempEIP := DEST(Offset);
    IF target mode = Compatibility mode
        THEN tempEIP := tempEIP AND 00000000_FFFFFFFFH; FI;
    IF OperandSize = 16
        THEN tempEIP := tempEIP AND 0000FFFFH; FI; (* Clear upper 16 bits *)
    IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode) and (tempEIP outside new code segment limit)
        THEN #GP(0); FI;
    IF tempEIP is non-canonical
        THEN #GP(0); FI;
    IF ShadowStackEnabled(CPL)
        IF IA32_EFER.LMA & CS.L
            tempPushLIP = RIP
        ELSE
            tempPushLIP = CSBASE + EIP;
        FI;
        tempPushCS = CS;
    FI;
    IF OperandSize = 32
        THEN
            Push(CS); (* Padded with 16 high-order bits *)
            Push(EIP);

```

```

    CS := DEST(CodeSegmentSelector);
    (* Segment descriptor information also loaded *)
    CS(RPL) := CPL;
    EIP := tempEIP;
ELSE
    IF OperandSize = 16
        THEN
            Push(CS);
            Push(IP);
            CS := DEST(CodeSegmentSelector);
            (* Segment descriptor information also loaded *)
            CS(RPL) := CPL;
            EIP := tempEIP;
        ELSE (* OperandSize = 64 *)
            Push(CS); (* Padded with 48 high-order bits *)
            Push(RIP);
            CS := DEST(CodeSegmentSelector);
            (* Segment descriptor information also loaded *)
            CS(RPL) := CPL;
            RIP := tempEIP;

    FI;
FI;
IF ShadowStackEnabled(CPL)
    IF (IA32_EFER.LMA and DEST(CodeSegmentSelector).L) = 0
        (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
        IF (SSP & 0xFFFFFFFF00000000 != 0)
            THEN #GP(0); FI;
        FI;
    (* align to 8 byte boundary if not already aligned *)
    tempSSP = SSP;
    Shadow_stack_store 4 bytes of 0 to (SSP - 4)
    SSP = SSP & 0xFFFFFFFFFFFFFFF8H
    ShadowStackPush8B(tempPushCS); (* Padded with 48 high-order 0 bits *)
    ShadowStackPush8B(tempPushLIP); (* Padded 32 high-order bits of 0 for 32 bit LIP*)
    ShadowStackPush8B(tempSSP);
    FI;
IF EndbranchEnabled(CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
END;

CALL-GATE:
    IF call gate (DPL < CPL) or (RPL > DPL)
        THEN #GP(call-gate selector); FI;
    IF call gate not present
        THEN #NP(call-gate selector); FI;
    IF call-gate code-segment selector is NULL

```

```

    THEN #GP(0); FI;
IF call-gate code-segment selector index is outside descriptor table limits
    THEN #GP(call-gate code-segment selector); FI;
Read call-gate code-segment descriptor;
IF call-gate code-segment descriptor does not indicate a code segment
or call-gate code-segment descriptor DPL > CPL
    THEN #GP(call-gate code-segment selector); FI;
IF IA32_EFER.LMA = 1 AND (call-gate code-segment descriptor is
not a 64-bit code segment or call-gate code-segment descriptor has both L-bit and D-bit set)
    THEN #GP(call-gate code-segment selector); FI;
IF call-gate code segment not present
    THEN #NP(call-gate code-segment selector); FI;
IF call-gate code segment is non-conforming and DPL < CPL
    THEN go to MORE-PRIVILEGE;
    ELSE go to SAME-PRIVILEGE;
FI;
END;

```

MORE-PRIVILEGE:

```

IF current TSS is 32-bit
    THEN
        TSSstackAddress := (new code-segment DPL * 8) + 4;
        IF (TSSstackAddress + 5) > current TSS limit
            THEN #TS(current TSS selector); FI;
        NewSS := 2 bytes loaded from (TSS base + TSSstackAddress + 4);
        NewESP := 4 bytes loaded from (TSS base + TSSstackAddress);
    ELSE
        IF current TSS is 16-bit
            THEN
                TSSstackAddress := (new code-segment DPL * 4) + 2
                IF (TSSstackAddress + 3) > current TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS := 2 bytes loaded from (TSS base + TSSstackAddress + 2);
                NewESP := 2 bytes loaded from (TSS base + TSSstackAddress);
            ELSE (* current TSS is 64-bit *)
                TSSstackAddress := (new code-segment DPL * 8) + 4;
                IF (TSSstackAddress + 7) > current TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS := new code-segment DPL; (* NULL selector with RPL = new CPL *)
                NewRSP := 8 bytes loaded from (current TSS base + TSSstackAddress);
        FI;
    FI;
IF IA32_EFER.LMA = 0 and NewSS is NULL
    THEN #TS(NewSS); FI;
Read new stack-segment descriptor;
IF IA32_EFER.LMA = 0 and (NewSS RPL ≠ new code-segment DPL
or new stack-segment DPL ≠ new code-segment DPL or new stack segment is not a
writable data segment)
    THEN #TS(NewSS); FI;
IF IA32_EFER.LMA = 0 and new stack segment not present
    THEN #SS(NewSS); FI;
IF CallGateSize = 32
    THEN
        IF new stack does not have room for parameters plus 16 bytes

```

```

        THEN #SS(NewSS); FI;
    IF CallGate(InstructionPointer) not within new code-segment limit
        THEN #GP(0); FI;
    SS := newSS; (* Segment descriptor information also loaded *)
    ESP := newESP;
    CS:EIP := CallGate(CS:InstructionPointer);
    (* Segment descriptor information also loaded *)
    Push(oldSS:oldESP); (* From calling procedure *)
    temp := parameter count from call gate, masked to 5 bits;
    Push(parameters from calling procedure's stack, temp)
    Push(oldCS:oldEIP); (* Return address to calling procedure *)
ELSE
    IF CallGateSize = 16
        THEN
            IF new stack does not have room for parameters plus 8 bytes
                THEN #SS(NewSS); FI;
            IF (CallGate(InstructionPointer) AND FFFFH) not in new code-segment limit
                THEN #GP(0); FI;
            SS := newSS; (* Segment descriptor information also loaded *)
            ESP := newESP;
            CS:IP := CallGate(CS:InstructionPointer);
            (* Segment descriptor information also loaded *)
            Push(oldSS:oldESP); (* From calling procedure *)
            temp := parameter count from call gate, masked to 5 bits;
            Push(parameters from calling procedure's stack, temp)
            Push(oldCS:oldEIP); (* Return address to calling procedure *)
        ELSE (* CallGateSize = 64 *)
            IF pushing 32 bytes on the stack would use a non-canonical address
                THEN #SS(NewSS); FI;
            IF (CallGate(InstructionPointer) is non-canonical)
                THEN #GP(0); FI;
            SS := NewSS; (* NewSS is NULL)
            RSP := NewESP;
            CS:IP := CallGate(CS:InstructionPointer);
            (* Segment descriptor information also loaded *)
            Push(oldSS:oldESP); (* From calling procedure *)
            Push(oldCS:oldEIP); (* Return address to calling procedure *)
        FI;
    FI;
IF ShadowStackEnabled(CPL) AND CPL = 3
    THEN
        IF IA32_EFER.LMA = 0
            THEN IA32_PL3_SSP := SSP;
            ELSE (* adjust so bits 63:N get the value of bit N-1, where N is the CPU's maximum linear-address width *)
                IA32_PL3_SSP := LA_adjust(SSP);
        FI;
    FI;
CPL := CodeSegment(DPL)
CS(RPL) := CPL
IF ShadowStackEnabled(CPL)
    oldSSP := SSP
    SSP := IA32_PLi_SSP; (* where i is the CPL *)
    IF SSP & 0x07 != 0 (* if SSP not aligned to 8 bytes then #GP *)
        THEN #GP(0); FI;

```

```

(* Token and CS:LIP:oldSSP pushed on shadow stack must be contained in a naturally aligned 32-byte region*)
IF (SSP & ~0x1F) != ((SSP - 24) & ~0x1F)
    #GP(0); FI;
IF ((IA32_EFER.LMA and CS.L) = 0 AND SSP[63:32] != 0)
    THEN #GP(0); FI;
expected_token_value = SSP          (* busy bit - bit position 0 - must be clear *)
new_token_value = SSP | BUSY_BIT    (* Set the busy bit *)
IF shadow_stack_lock_cmpxchg8b(SSP, new_token_value, expected_token_value) != expected_token_value
    THEN #GP(0); FI;
IF oldSS.DPL != 3
    (* These stack pushes should not cause faults, VM exits, or data breakpoints *)
    (* Such events will apply to the earlier accesses to the token, which is in the same naturally aligned 32-byte region *)
    ShadowStackPush8B(oldCS); (* Padded with 48 high-order bits of 0 *)
    ShadowStackPush8B(oldCSBASE+oldRIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
    ShadowStackPush8B(oldSSP);
FI;
FI;
IF EndbranchEnabled (CPL)
    IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
    IA32_S_CET.SUPPRESS = 0
FI;
END;

SAME-PRIVILEGE:
IF CallGateSize = 32
    THEN
        IF stack does not have room for 8 bytes
            THEN #SS(0); FI;
        IF CallGate(InstructionPointer) not within code segment limit
            THEN #GP(0); FI;
        CS:EIP := CallGate(CS:EIP) (* Segment descriptor information also loaded *)
        Push(oldCS:oldEIP); (* Return address to calling procedure *)
    ELSE
        If CallGateSize = 16
            THEN
                IF stack does not have room for 4 bytes
                    THEN #SS(0); FI;
                IF CallGate(InstructionPointer) not within code segment limit
                    THEN #GP(0); FI;
                CS:IP := CallGate(CS:instruction pointer);
                (* Segment descriptor information also loaded *)
                Push(oldCS:oldIP); (* Return address to calling procedure *)
            ELSE (* CallGateSize = 64)
                IF pushing 16 bytes on the stack touches non-canonical addresses
                    THEN #SS(0); FI;
                IF RIP non-canonical
                    THEN #GP(0); FI;
                CS:IP := CallGate(CS:instruction pointer);
                (* Segment descriptor information also loaded *)
                Push(oldCS:oldIP); (* Return address to calling procedure *)
        FI;
    FI;
CS(RPL) := CPL
IF ShadowStackEnabled(CPL)

```



```

(* Align to next 8 byte boundary *)
tempSSP = SSP;
Shadow_stack_store 4 bytes of 0 to (SSP - 4)
SSP = SSP & 0xFFFFFFFFFFFFF8H;
(* push cs:lip:ssp on shadow stack *)
ShadowStackPush8B(oldCS); (* Padded with 48 high-order bits of 0 *)
ShadowStackPush8B(oldCSBASE + oldRIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
ShadowStackPush8B(tempSSP);
FI;
IF EndbranchEnabled (CPL)
  IF CPL = 3
    THEN
      IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH;
      IA32_U_CET.SUPPRESS = 0
    ELSE
      IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
      IA32_S_CET.SUPPRESS = 0
  FI;
FI;
END;

```

TASK-GATE:

```

IF task gate DPL < CPL or RPL
  THEN #GP(task gate selector); FI;
IF task gate not present
  THEN #NP(task gate selector); FI;
Read the TSS segment selector in the task-gate descriptor;
IF TSS segment selector local/global bit is set to local
or index not within GDT limits
  THEN #GP(TSS selector); FI;
Access TSS descriptor in GDT;
IF descriptor is not a TSS segment
  THEN #GP(TSS selector); FI;
IF TSS descriptor specifies that the TSS is busy
  THEN #GP(TSS selector); FI;
IF TSS not present
  THEN #NP(TSS selector); FI;
SWITCH-TASKS (with nesting) to TSS;
IF EIP not within code segment limit
  THEN #GP(0); FI;
END;

```

TASK-STATE-SEGMENT:

```

IF TSS DPL < CPL or RPL
or TSS descriptor indicates TSS not available
  THEN #GP(TSS selector); FI;
IF TSS is not present
  THEN #NP(TSS selector); FI;
SWITCH-TASKS (with nesting) to TSS;
IF EIP not within code segment limit
  THEN #GP(0); FI;
END;

```

Flags Affected

All flags are affected if a task switch occurs; no flags are affected if a task switch does not occur.

Protected Mode Exceptions

#GP(0)	<p>If the target offset in destination operand is beyond the new code segment limit.</p> <p>If the segment selector in the destination operand is NULL.</p> <p>If the code segment selector in the gate is NULL.</p> <p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.</p> <p>If target mode is compatibility mode and SSP is not in low 4GB.</p> <p>If SSP in IA32_PLi_SSP (where i is the new CPL) is not 8 byte aligned.</p> <p>If the token and the stack frame to be pushed on shadow stack are not contained in a naturally aligned 32-byte region of the shadow stack.</p> <p>If "supervisor Shadow Stack" token on new shadow stack is marked busy.</p> <p>If destination mode is 32-bit or compatibility mode, but SSP address in "supervisor shadow stack" token is beyond 4GB.</p> <p>If SSP address in "supervisor shadow stack" token does not match SSP address in IA32_PLi_SSP (where i is the new CPL).</p>
#GP(selector)	<p>If a code segment or gate or TSS selector index is outside descriptor table limits.</p> <p>If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment.</p> <p>If the DPL for a nonconforming-code segment is not equal to the CPL or the RPL for the segment's segment selector is greater than the CPL.</p> <p>If the DPL for a conforming-code segment is greater than the CPL.</p> <p>If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS's segment selector.</p> <p>If the segment descriptor for a segment selector from a call gate does not indicate it is a code segment.</p> <p>If the segment selector from a call gate is beyond the descriptor table limits.</p> <p>If the DPL for a code-segment obtained from a call gate is greater than the CPL.</p> <p>If the segment selector for a TSS has its local/global bit set for local.</p> <p>If a TSS segment descriptor specifies that the TSS is busy or not available.</p>
#SS(0)	<p>If pushing the return address, parameters, or stack segment pointer onto the stack exceeds the bounds of the stack segment, when no stack switch occurs.</p> <p>If a memory operand effective address is outside the SS segment limit.</p>
#SS(selector)	<p>If pushing the return address, parameters, or stack segment pointer onto the stack exceeds the bounds of the stack segment, when a stack switch occurs.</p> <p>If the SS register is being loaded as part of a stack switch and the segment pointed to is marked not present.</p> <p>If stack segment does not have room for the return address, parameters, or stack segment pointer, when stack switch occurs.</p>
#NP(selector)	<p>If a code segment, data segment, call gate, task gate, or TSS is not present.</p>
#TS(selector)	<p>If the new stack segment selector and ESP are beyond the end of the TSS.</p> <p>If the new stack segment selector is NULL.</p> <p>If the RPL of the new stack segment selector in the TSS is not equal to the DPL of the code segment being accessed.</p> <p>If DPL of the stack segment descriptor for the new stack segment is not equal to the DPL of the code segment descriptor.</p>

	If the new stack segment is not a writable data segment.
	If segment-selector index for stack segment is outside descriptor table limits.
#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. If the target offset is beyond the code 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. If the target offset is beyond the code 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.

#GP(selector)	If a memory address accessed by the selector is in non-canonical space.
#GP(0)	If the target offset in the destination operand is non-canonical.

64-Bit Mode Exceptions

#GP(0)	If a memory address is non-canonical. If target offset in destination operand is non-canonical. If the segment selector in the destination operand is NULL. If the code segment selector in the 64-bit gate is NULL. If target mode is compatibility mode and SSP is not in low 4GB. If SSP in IA32_PLi_SSP (where i is the new CPL) is not 8 byte aligned. If the token and the stack frame to be pushed on shadow stack are not contained in a naturally aligned 32-byte region of the shadow stack. If "supervisor Shadow Stack" token on new shadow stack is marked busy. If destination mode is 32-bit mode or compatibility mode, but SSP address in "super-visor shadow" stack token is beyond 4GB. If SSP address in "supervisor shadow stack" token does not match SSP address in IA32_PLi_SSP (where i is the new CPL).
#GP(selector)	If code segment or 64-bit call gate is outside descriptor table limits. If code segment or 64-bit call gate overlaps non-canonical space. If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, or 64-bit call gate. If the segment descriptor pointed to by the segment selector in the destination operand is a code segment and has both the D-bit and the L-bit set. If the DPL for a nonconforming-code segment is not equal to the CPL, or the RPL for the segment's segment selector is greater than the CPL. If the DPL for a conforming-code segment is greater than the CPL. If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate. If the upper type field of a 64-bit call gate is not 0x0.

	If the segment selector from a 64-bit call gate is beyond the descriptor table limits.
	If the DPL for a code-segment obtained from a 64-bit call gate is greater than the CPL.
	If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear.
	If the segment descriptor for a segment selector from the 64-bit call gate does not indicate it is a code segment.
#SS(0)	If pushing the return offset or CS selector onto the stack exceeds the bounds of the stack segment when no stack switch occurs.
	If a memory operand effective address is outside the SS segment limit.
	If the stack address is in a non-canonical form.
#SS(selector)	If pushing the old values of SS selector, stack pointer, EFLAGS, CS selector, offset, or error code onto the stack violates the canonical boundary when a stack switch occurs.
#NP(selector)	If a code segment or 64-bit call gate is not present.
#TS(selector)	If the load of the new RSP exceeds the limit of the TSS.
#UD	(64-bit mode only) If a far call is direct to an absolute address in memory.
	If the LOCK prefix is used.
#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.

INCSSPD/INCSSPQ—Increment Shadow Stack Pointer

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F AE /05 INCSSPD r32	R	V/V	CET_SS	Increment SSP by 4 * r32[7:0].
F3 REX.W 0F AE /05 INCSSPQ r64	R	V/N.E.	CET_SS	Increment SSP by 8 * r64[7:0].

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
R	NA	ModRM:r/m (r)	NA	NA	NA

Description

This instruction can be used to increment the current shadow stack pointer by the operand size of the instruction times the unsigned 8-bit value specified by bits 7:0 in the source operand. The instruction performs a pop and discard of the first and last element on the shadow stack in the range specified by the unsigned 8-bit value in bits 7:0 of the source operand.

Operation

IF CPL = 3

IF (CR4.CET & IA32_U_CET.SH_STK_EN) = 0
THEN #UD; FI;

ELSE

IF (CR4.CET & IA32_S_CET.SH_STK_EN) = 0
THEN #UD; FI;

FI;

IF (operand size is 64-bit)

THEN

Range := R64[7:0];

shadow_stack_load 8 bytes from SSP;

IF Range > 0

THEN shadow_stack_load 8 bytes from SSP + 8 * (Range - 1);

FI;

SSP := SSP + Range * 8;

ELSE

Range := R32[7:0];

shadow_stack_load 4 bytes from SSP;

IF Range > 0

THEN shadow_stack_load 4 bytes from SSP + 4 * (Range - 1);

FI;

SSP := SSP + Range * 4;

FI;

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

INCSSPD void _incsspd(int);

INCSSPQ void _incsspq(int);

Protected Mode Exceptions

- #UD If the LOCK prefix is used.
 If CR4.CET = 0.
 If CPL = 3 and IA32_U_CET.SH_STK_EN = 0.
 If CPL < 3 and IA32_S_CET.SH_STK_EN = 0.
- #PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

- #UD The INCSSP instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

- #UD The INCSSP instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

JMP—Jump

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
EB <i>cb</i>	JMP <i>rel8</i>	D	Valid	Valid	Jump short, RIP = RIP + 8-bit displacement sign extended to 64-bits
E9 <i>cw</i>	JMP <i>rel16</i>	D	N.S.	Valid	Jump near, relative, displacement relative to next instruction. Not supported in 64-bit mode.
E9 <i>cd</i>	JMP <i>rel32</i>	D	Valid	Valid	Jump near, relative, RIP = RIP + 32-bit displacement sign extended to 64-bits
FF <i>/4</i>	JMP <i>r/m16</i>	M	N.S.	Valid	Jump near, absolute indirect, address = zero-extended <i>r/m16</i> . Not supported in 64-bit mode.
FF <i>/4</i>	JMP <i>r/m32</i>	M	N.S.	Valid	Jump near, absolute indirect, address given in <i>r/m32</i> . Not supported in 64-bit mode.
FF <i>/4</i>	JMP <i>r/m64</i>	M	Valid	N.E.	Jump near, absolute indirect, RIP = 64-Bit offset from register or memory
EA <i>cd</i>	JMP <i>ptr16:16</i>	S	Inv.	Valid	Jump far, absolute, address given in operand
EA <i>cp</i>	JMP <i>ptr16:32</i>	S	Inv.	Valid	Jump far, absolute, address given in operand
FF <i>/5</i>	JMP <i>m16:16</i>	M	Valid	Valid	Jump far, absolute indirect, address given in <i>m16:16</i>
FF <i>/5</i>	JMP <i>m16:32</i>	M	Valid	Valid	Jump far, absolute indirect, address given in <i>m16:32</i> .
REX.W FF <i>/5</i>	JMP <i>m16:64</i>	M	Valid	N.E.	Jump far, absolute indirect, address given in <i>m16:64</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
S	Segment + Absolute Address	NA	NA	NA
D	Offset	NA	NA	NA
M	ModRM:r/m (<i>r</i>)	NA	NA	NA

Description

Transfers program control to a different point in the instruction stream without recording return information. The destination (target) operand specifies the address of the instruction being jumped to. This operand can be an immediate value, a general-purpose register, or a memory location.

This instruction can be used to execute four different types of jumps:

- Near jump—A jump to an instruction within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment jump.
- Short jump—A near jump where the jump range is limited to -128 to $+127$ from the current EIP value.
- Far jump—A jump to an instruction located in a different segment than the current code segment but at the same privilege level, sometimes referred to as an intersegment jump.
- Task switch—A jump to an instruction located in a different task.

A task switch can only be executed in protected mode (see Chapter 7, in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for information on performing task switches with the JMP instruction).

Near and Short Jumps. When executing a near jump, the processor jumps to the address (within the current code segment) that is specified with the target operand. The target operand specifies either an absolute offset (that is an offset from the base of the code segment) or a relative offset (a signed displacement relative to the current

value of the instruction pointer in the EIP register). A near jump to a relative offset of 8-bits (*rel8*) is referred to as a short jump. The CS register is not changed on near and short jumps.

An absolute offset is specified indirectly in a general-purpose register or a memory location (*r/m16* or *r/m32*). The operand-size attribute determines the size of the target operand (16 or 32 bits). Absolute offsets are loaded directly into the EIP register. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared, resulting in a maximum instruction pointer size of 16 bits.

A relative offset (*rel8*, *rel16*, or *rel32*) is generally specified as a label in assembly code, but at the machine code level, it is encoded as a signed 8-, 16-, or 32-bit immediate value. This value is added to the value in the EIP register. (Here, the EIP register contains the address of the instruction following the JMP instruction). When using relative offsets, the opcode (for short vs. near jumps) and the operand-size attribute (for near relative jumps) determines the size of the target operand (8, 16, or 32 bits).

Far Jumps in Real-Address or Virtual-8086 Mode. When executing a far jump in real-address or virtual-8086 mode, the processor jumps to the code segment and offset specified with the target operand. Here the target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). With the pointer method, the segment and address of the called procedure is encoded in the instruction, using a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address immediate. With the indirect method, the target operand specifies a memory location that contains a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address. The far address is loaded directly into the CS and EIP registers. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared.

Far Jumps in Protected Mode. When the processor is operating in protected mode, the JMP instruction can be used to perform the following three types of far jumps:

- A far jump to a conforming or non-conforming code segment.
- A far jump through a call gate.
- A task switch.

(The JMP instruction cannot be used to perform inter-privilege-level far jumps.)

In protected mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate, task gate, or TSS) and access rights determine the type of jump to be performed.

If the selected descriptor is for a code segment, a far jump to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far jump to the same privilege level in protected mode is very similar to one carried out in real-address or virtual-8086 mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register, and the offset from the instruction is loaded into the EIP register. Note that a call gate (described in the next paragraph) can also be used to perform far call to a code segment at the same privilege level. Using this mechanism provides an extra level of indirection and is the preferred method of making jumps between 16-bit and 32-bit code segments.

When executing a far jump through a call gate, the segment selector specified by the target operand identifies the call gate. (The offset part of the target operand is ignored.) The processor then jumps to the code segment specified in the call gate descriptor and begins executing the instruction at the offset specified in the call gate. No stack switch occurs. Here again, the target operand can specify the far address of the call gate either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*).

Executing a task switch with the JMP instruction is somewhat similar to executing a jump through a call gate. Here the target operand specifies the segment selector of the task gate for the task being switched to (and the offset part of the target operand is ignored). The task gate in turn points to the TSS for the task, which contains the segment selectors for the task's code and stack segments. The TSS also contains the EIP value for the next instruction that was to be executed before the task was suspended. This instruction pointer value is loaded into the EIP register so that the task begins executing again at this next instruction.

The JMP instruction can also specify the segment selector of the TSS directly, which eliminates the indirection of the task gate. See Chapter 7 in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for detailed information on the mechanics of a task switch.

Note that when you execute at task switch with a JMP instruction, the nested task flag (NT) is not set in the EFLAGS register and the new TSS's previous task link field is not loaded with the old task's TSS selector. A return to the previous task can thus not be carried out by executing the IRET instruction. Switching tasks with the JMP instruction differs in this regard from the CALL instruction which does set the NT flag and save the previous task link information, allowing a return to the calling task with an IRET instruction.

Refer to Chapter 6, "Procedure Calls, Interrupts, and Exceptions" and Chapter 18, "Control-Flow Enforcement Technology (CET)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for CET details.

In 64-Bit Mode. The instruction's operation size is fixed at 64 bits. If a selector points to a gate, then RIP equals the 64-bit displacement taken from gate; else RIP equals the zero-extended offset from the far pointer referenced in the instruction.

See the summary chart at the beginning of this section for encoding data and limits.

Instruction ordering. Instructions following a far jump may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the far jump have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Instructions sequentially following a near indirect JMP instruction (i.e., those not at the target) may be executed speculatively. If software needs to prevent this (e.g., in order to prevent a speculative execution side channel), then an INT3 or LFENCE instruction opcode can be placed after the near indirect JMP in order to block speculative execution.

Operation

```

IF near jump
  IF 64-bit Mode
    THEN
      IF near relative jump
        THEN
          tempRIP := RIP + DEST; (* RIP is instruction following JMP instruction*)
        ELSE (* Near absolute jump *)
          tempRIP := DEST;
      FI;
    ELSE
      IF near relative jump
        THEN
          tempEIP := EIP + DEST; (* EIP is instruction following JMP instruction*)
        ELSE (* Near absolute jump *)
          tempEIP := DEST;
      FI;
  FI;
  IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode)
  and tempEIP outside code segment limit
    THEN #GP(0); FI
  IF 64-bit mode and tempRIP is not canonical
    THEN #GP(0);
  FI;
  IF OperandSize = 32
    THEN
      EIP := tempEIP;
    ELSE
      IF OperandSize = 16
        THEN (* OperandSize = 16 *)
          EIP := tempEIP AND 0000FFFFH;
        ELSE (* OperandSize = 64)
          RIP := tempRIP;

```

```

    FI;
FI;
IF (JMP near indirect, absolute indirect)
    IF EndbranchEnabledAndNotSuppressed(CPL)
        IF CPL = 3
            THEN
                IF ( no 3EH prefix OR IA32_U_CET.NO_TRACK_EN == 0 )
                    THEN
                        IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
                    FI;
                ELSE
                    IF ( no 3EH prefix OR IA32_S_CET.NO_TRACK_EN == 0 )
                        THEN
                            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
                        FI;
                    FI;
                FI;
            FI;
        FI;
    FI;
FI;
FI;
IF far jump and (PE = 0 or (PE = 1 AND VM = 1)) (* Real-address or virtual-8086 mode *)
    THEN
        tempEIP := DEST(Offset); (* DEST is ptr16:32 or [m16:32] *)
        IF tempEIP is beyond code segment limit
            THEN #GP(0); FI;
        CS := DEST(segment selector); (* DEST is ptr16:32 or [m16:32] *)
        IF OperandSize = 32
            THEN
                EIP := tempEIP; (* DEST is ptr16:32 or [m16:32] *)
            ELSE (* OperandSize = 16 *)
                EIP := tempEIP AND 0000FFFFH; (* Clear upper 16 bits *)
            FI;
        FI;
    FI;
FI;
IF far jump and (PE = 1 and VM = 0)
    (* IA-32e mode or protected mode, not virtual-8086 mode *)
    THEN
        IF effective address in the CS, DS, ES, FS, GS, or SS segment is illegal
            or segment selector in target operand NULL
            THEN #GP(0); FI;
        IF segment selector index not within descriptor table limits
            THEN #GP(new selector); FI;
        Read type and access rights of segment descriptor;
        IF (IA32_EFER.LMA = 0)
            THEN
                IF segment type is not a conforming or nonconforming code
                    segment, call gate, task gate, or TSS
                    THEN #GP(segment selector); FI;
            ELSE
                IF segment type is not a conforming or nonconforming code segment
                    call gate
                    THEN #GP(segment selector); FI;
            FI;
        FI;
        Depending on type and access rights:
        GO TO CONFORMING-CODE-SEGMENT;
        GO TO NONCONFORMING-CODE-SEGMENT;

```

```

        GO TO CALL-GATE;
        GO TO TASK-GATE;
        GO TO TASK-STATE-SEGMENT;
ELSE
    #GP(segment selector);
FI;
CONFORMING-CODE-SEGMENT:
IF L-Bit = 1 and D-BIT = 1 and IA32_EFER.LMA = 1
    THEN GP(new code segment selector); FI;
IF DPL > CPL
    THEN #GP(segment selector); FI;
IF segment not present
    THEN #NP(segment selector); FI;
tempEIP := DEST(Offset);
IF OperandSize = 16
    THEN tempEIP := tempEIP AND 0000FFFFH;
FI;
IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode) and
tempEIP outside code segment limit
    THEN #GP(0); FI;
IF tempEIP is non-canonical
    THEN #GP(0); FI;
IF ShadowStackEnabled(CPL)
    IF (IA32_EFER.LMA and DEST(segment selector).L) = 0
        (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
        IF (SSP & 0xFFFFFFFF00000000 != 0)
            THEN #GP(0); FI;
    FI;
FI;
CS := DEST[segment selector]; (* Segment descriptor information also loaded *)
CS(RPL) := CPL
EIP := tempEIP;
IF EndbranchEnabled(CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
END;
NONCONFORMING-CODE-SEGMENT:
IF L-Bit = 1 and D-BIT = 1 and IA32_EFER.LMA = 1
    THEN GP(new code segment selector); FI;
IF (RPL > CPL) OR (DPL ≠ CPL)
    THEN #GP(code segment selector); FI;
IF segment not present
    THEN #NP(segment selector); FI;
tempEIP := DEST(Offset);
IF OperandSize = 16
    THEN tempEIP := tempEIP AND 0000FFFFH; FI;
IF (IA32_EFER.LMA = 0 OR target mode = Compatibility mode)

```

```

and tempEIP outside code segment limit
    THEN #GP(0); FI
IF tempEIP is non-canonical THEN #GP(0); FI;
IF ShadowStackEnabled(CPL)
    IF (IA32_EFER.LMA and DEST(segment selector).L) = 0
        (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
        IF (SSP & 0xFFFFFFFF00000000 != 0)
            THEN #GP(0); FI;
    FI;
FI;
CS := DEST[segment selector]; (* Segment descriptor information also loaded *)
CS(RPL) := CPL;
EIP := tempEIP;
IF EndbranchEnabled(CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
END;

```

CALL-GATE:

```

IF call gate DPL < CPL
or call gate DPL < call gate segment-selector RPL
    THEN #GP(call gate selector); FI;
IF call gate not present
    THEN #NP(call gate selector); FI;
IF call gate code-segment selector is NULL
    THEN #GP(0); FI;
IF call gate code-segment selector index outside descriptor table limits
    THEN #GP(code segment selector); FI;
Read code segment descriptor;
IF code-segment segment descriptor does not indicate a code segment
or code-segment segment descriptor is conforming and DPL > CPL
or code-segment segment descriptor is non-conforming and DPL ≠ CPL
    THEN #GP(code segment selector); FI;
IF IA32_EFER.LMA = 1 and (code-segment descriptor is not a 64-bit code segment
or code-segment segment descriptor has both L-Bit and D-bit set)
    THEN #GP(code segment selector); FI;
IF code segment is not present
    THEN #NP(code-segment selector); FI;
tempEIP := DEST(Offset);
IF GateSize = 16
    THEN tempEIP := tempEIP AND 0000FFFFH; FI;
IF (IA32_EFER.LMA = 0 OR target mode = Compatibility mode) AND tempEIP
outside code segment limit
    THEN #GP(0); FI
CS := DEST[SegmentSelector]; (* Segment descriptor information also loaded *)
CS(RPL) := CPL;
EIP := tempEIP;

```

```

IF EndbranchEnabled(CPL)
  IF CPL = 3
    THEN
      IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH;
      IA32_U_CET.SUPPRESS = 0
    ELSE
      IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
      IA32_S_CET.SUPPRESS = 0
  FI;
FI;
END;
TASK-GATE:
  IF task gate DPL < CPL
  or task gate DPL < task gate segment-selector RPL
    THEN #GP(task gate selector); FI;
  IF task gate not present
    THEN #NP(gate selector); FI;
  Read the TSS segment selector in the task-gate descriptor;
  IF TSS segment selector local/global bit is set to local
  or index not within GDT limits
  or descriptor is not a TSS segment
  or TSS descriptor specifies that the TSS is busy
    THEN #GP(TSS selector); FI;
  IF TSS not present
    THEN #NP(TSS selector); FI;
  SWITCH-TASKS to TSS;
  IF EIP not within code segment limit
    THEN #GP(0); FI;
END;
TASK-STATE-SEGMENT:
  IF TSS DPL < CPL
  or TSS DPL < TSS segment-selector RPL
  or TSS descriptor indicates TSS not available
    THEN #GP(TSS selector); FI;
  IF TSS is not present
    THEN #NP(TSS selector); FI;
  SWITCH-TASKS to TSS;
  IF EIP not within code segment limit
    THEN #GP(0); FI;
END;

```

Flags Affected

All flags are affected if a task switch occurs; no flags are affected if a task switch does not occur.

Protected Mode Exceptions

#GP(0)	<p>If offset in target operand, call gate, or TSS is beyond the code segment limits.</p> <p>If the segment selector in the destination operand, call gate, task gate, or TSS is NULL.</p> <p>If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.</p> <p>If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.</p> <p>If target mode is compatibility mode and SSP is not in low 4GB.</p>
#GP(selector)	If the segment selector index is outside descriptor table limits.

If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment.

If the DPL for a nonconforming-code segment is not equal to the CPL

(When not using a call gate.) If the RPL for the segment's segment selector is greater than the CPL.

If the DPL for a conforming-code segment is greater than the CPL.

If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS's segment selector.

If the segment descriptor for selector in a call gate does not indicate it is a code segment.

If the segment descriptor for the segment selector in a task gate does not indicate an available TSS.

If the segment selector for a TSS has its local/global bit set for local.

If a TSS segment descriptor specifies that the TSS is busy or not available.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NP (selector) If the code segment being accessed is not present.

If call gate, task gate, or TSS not present.

#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. (Only occurs when fetching target from memory.)

#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.

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 the target operand is beyond the code segment limits.

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. (Only occurs when fetching target from memory.)

#UD If the LOCK prefix is used.

Compatibility Mode Exceptions

Same as 64-bit mode exceptions.

64-Bit Mode Exceptions

#GP(0) If a memory address is non-canonical.

If target offset in destination operand is non-canonical.

If target offset in destination operand is beyond the new code segment limit.

If the segment selector in the destination operand is NULL.

If the code segment selector in the 64-bit gate is NULL.

If transitioning to compatibility mode and the SSP is beyond 4GB.

#GP(selector) If the code segment or 64-bit call gate is outside descriptor table limits.

If the code segment or 64-bit call gate overlaps non-canonical space.

- If the segment descriptor from a 64-bit call gate is in non-canonical space.
- If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, 64-bit call gate.
- If the segment descriptor pointed to by the segment selector in the destination operand is a code segment, and has both the D-bit and the L-bit set.
- If the DPL for a nonconforming-code segment is not equal to the CPL, or the RPL for the segment's segment selector is greater than the CPL.
- If the DPL for a conforming-code segment is greater than the CPL.
- If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate.
- If the upper type field of a 64-bit call gate is not 0x0.
- If the segment selector from a 64-bit call gate is beyond the descriptor table limits.
- If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear.
- If the segment descriptor for a segment selector from the 64-bit call gate does not indicate it is a code segment.
- If the code segment is non-conforming and $CPL \neq DPL$.
- If the code segment is confirming and $CPL < DPL$.
- #NP(selector) If a code segment or 64-bit call gate is not present.
- #UD (64-bit mode only) If a far jump is direct to an absolute address in memory.
- If the LOCK prefix is used.
- #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.

KADDW/KADDB/KADDQ/KADD—ADD Two Masks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.L1.0F.W0 4A /r KADDW k1, k2, k3	RVR	V/V	AVX512DQ	Add 16 bits masks in k2 and k3 and place result in k1.
VEX.L1.66.0F.W0 4A /r KADDB k1, k2, k3	RVR	V/V	AVX512DQ	Add 8 bits masks in k2 and k3 and place result in k1.
VEX.L1.0F.W1 4A /r KADDQ k1, k2, k3	RVR	V/V	AVX512BW	Add 64 bits masks in k2 and k3 and place result in k1.
VEX.L1.66.0F.W1 4A /r KADD k1, k2, k3	RVR	V/V	AVX512BW	Add 32 bits masks in k2 and k3 and place result in k1.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3
RVR	ModRM:reg (w)	VEX.1vsv (r)	ModRM:r/m (r, ModRM:[7:6] must be 11b)

Description

Adds the vector mask k2 and the vector mask k3, and writes the result into vector mask k1.

Operation

KADDW

```
DEST[15:0] := SRC1[15:0] + SRC2[15:0]
DEST[MAX_KL-1:16] := 0
```

KADDB

```
DEST[7:0] := SRC1[7:0] + SRC2[7:0]
DEST[MAX_KL-1:8] := 0
```

KADDQ

```
DEST[63:0] := SRC1[63:0] + SRC2[63:0]
DEST[MAX_KL-1:64] := 0
```

KADD

```
DEST[31:0] := SRC1[31:0] + SRC2[31:0]
DEST[MAX_KL-1:32] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
KADDW __mmask16_kadd_mask16 (__mmask16 a, __mmask16 b);
KADDB __mmask8_kadd_mask8 (__mmask8 a, __mmask8 b);
KADDQ __mmask64_kadd_mask64 (__mmask64 a, __mmask64 b);
KADD __mmask32_kadd_mask32 (__mmask32 a, __mmask32 b);
```

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Table 2-63, "TYPE K20 Exception Definition (VEX-Encoded OpMask Instructions w/o Memory Arg)".

LDS/LES/LFS/LGS/LSS—Load Far Pointer

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
C5 /r	LDS r16,m16:16	RM	Invalid	Valid	Load DS:r16 with far pointer from memory.
C5 /r	LDS r32,m16:32	RM	Invalid	Valid	Load DS:r32 with far pointer from memory.
OF B2 /r	LSS r16,m16:16	RM	Valid	Valid	Load SS:r16 with far pointer from memory.
OF B2 /r	LSS r32,m16:32	RM	Valid	Valid	Load SS:r32 with far pointer from memory.
REX + OF B2 /r	LSS r64,m16:64	RM	Valid	N.E.	Load SS:r64 with far pointer from memory.
C4 /r	LES r16,m16:16	RM	Invalid	Valid	Load ES:r16 with far pointer from memory.
C4 /r	LES r32,m16:32	RM	Invalid	Valid	Load ES:r32 with far pointer from memory.
OF B4 /r	LFS r16,m16:16	RM	Valid	Valid	Load FS:r16 with far pointer from memory.
OF B4 /r	LFS r32,m16:32	RM	Valid	Valid	Load FS:r32 with far pointer from memory.
REX + OF B4 /r	LFS r64,m16:64	RM	Valid	N.E.	Load FS:r64 with far pointer from memory.
OF B5 /r	LGS r16,m16:16	RM	Valid	Valid	Load GS:r16 with far pointer from memory.
OF B5 /r	LGS r32,m16:32	RM	Valid	Valid	Load GS:r32 with far pointer from memory.
REX + OF B5 /r	LGS r64,m16:64	RM	Valid	N.E.	Load GS:r64 with far pointer from memory.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

Description

Loads a far pointer (segment selector and offset) from the second operand (source operand) into a segment register and the first operand (destination operand). The source operand specifies a 48-bit or a 32-bit pointer in memory depending on the current setting of the operand-size attribute (32 bits or 16 bits, respectively). The instruction opcode and the destination operand specify a segment register/general-purpose register pair. The 16-bit segment selector from the source operand is loaded into the segment register specified with the opcode (DS, SS, ES, FS, or GS). The 32-bit or 16-bit offset is loaded into the register specified with the destination operand.

If one of these instructions is executed in protected mode, additional information from the segment descriptor pointed to by the segment selector in the source operand is loaded in the hidden part of the selected segment register.

Also in protected mode, a NULL selector (values 0000 through 0003) can be loaded into DS, ES, FS, or GS registers without causing a protection exception. (Any subsequent reference to a segment whose corresponding segment register is loaded with a NULL selector, causes a general-protection exception (#GP) and no memory reference to the segment occurs.)

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.W promotes operation to specify a source operand referencing an 80-bit pointer (16-bit selector, 64-bit offset) in memory. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). See the summary chart at the beginning of this section for encoding data and limits.

Operation

64-BIT_MODE

IF SS is loaded

THEN

IF SegmentSelector = NULL and ((RPL = 3) or
(RPL ≠ 3 and RPL ≠ CPL))

THEN #GP(0);

ELSE IF descriptor is in non-canonical space

```

    THEN #GP(selector); FI;
ELSE IF Segment selector index is not within descriptor table limits
    or segment selector RPL ≠ CPL
    or access rights indicate nonwritable data segment
    or DPL ≠ CPL
    THEN #GP(selector); FI;
ELSE IF Segment marked not present
    THEN #SS(selector); FI;
FI;
SS := SegmentSelector(SRC);
SS := SegmentDescriptor([SRC]);
ELSE IF attempt to load DS, or ES
    THEN #UD;
ELSE IF FS, or GS is loaded with non-NULL segment selector
    THEN IF Segment selector index is not within descriptor table limits
        or access rights indicate segment neither data nor readable code segment
        or segment is data or nonconforming-code segment
        and ( RPL > DPL or CPL > DPL)
            THEN #GP(selector); FI;
        ELSE IF Segment marked not present
            THEN #NP(selector); FI;
        FI;
        SegmentRegister := SegmentSelector(SRC);
        SegmentRegister := SegmentDescriptor([SRC]);
    FI;
ELSE IF FS, or GS is loaded with a NULL selector:
    THEN
        SegmentRegister := NULLSelector;
        SegmentRegister(DescriptorValidBit) := 0; FI; (* Hidden flag;
            not accessible by software *)
    FI;
DEST := Offset(SRC);

PRETECTED MODE OR COMPATIBILITY MODE;
IF SS is loaded
    THEN
        IF SegementSelector = NULL
            THEN #GP(0);
        ELSE IF Segment selector index is not within descriptor table limits
            or segment selector RPL ≠ CPL
            or access rights indicate nonwritable data segment
            or DPL ≠ CPL
            THEN #GP(selector); FI;
        ELSE IF Segment marked not present
            THEN #SS(selector); FI;
        FI;
        SS := SegmentSelector(SRC);
        SS := SegmentDescriptor([SRC]);
    ELSE IF DS, ES, FS, or GS is loaded with non-NULL segment selector
        THEN IF Segment selector index is not within descriptor table limits
            or access rights indicate segment neither data nor readable code segment
            or segment is data or nonconforming-code segment
            and (RPL > DPL or CPL > DPL)
                THEN #GP(selector); FI;

```

```

    ELSE IF Segment marked not present
        THEN #NP(selector); FI;
    FI;
    SegmentRegister := SegmentSelector(SRC) AND RPL;
    SegmentRegister := SegmentDescriptor([SRC]);
FI;
ELSE IF DS, ES, FS, or GS is loaded with a NULL selector:
    THEN
        SegmentRegister := NULLSelector;
        SegmentRegister(DescriptorValidBit) := 0; FI; (* Hidden flag;
            not accessible by software *)
    FI;
    DEST := Offset(SRC);
Real-Address or Virtual-8086 Mode
    SegmentRegister := SegmentSelector(SRC); FI;
    DEST := Offset(SRC);

```

Flags Affected

None

Protected Mode Exceptions

#UD	If source operand is not a memory location. If the LOCK prefix is used.
#GP(0)	If a NULL selector is loaded into the SS register. 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 segment selector.
#GP(selector)	If the SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the segment selector RPL is not equal to CPL, the segment is a non-writable data segment, or DPL is not equal to CPL. If the DS, ES, FS, or GS register is being loaded with a non-NULL segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL and CPL are greater than DPL.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#SS(selector)	If the SS register is being loaded and the segment is marked not present.
#NP(selector)	If DS, ES, FS, or GS register is being loaded with a non-NULL segment selector and the segment is marked not present.
#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.

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 source operand is not a memory location. If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#UD	If source operand is not a memory location. If the LOCK prefix is used.
#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.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form. If a NULL selector is attempted to be loaded into the SS register in compatibility mode. If a NULL selector is attempted to be loaded into the SS register in CPL3 and 64-bit mode. If a NULL selector is attempted to be loaded into the SS register in non-CPL3 and 64-bit mode where its RPL is not equal to CPL.
#GP(Selector)	If the FS, or GS register is being loaded with a non-NULL segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the memory address of the descriptor is non-canonical, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL and CPL are greater than DPL. If the SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the memory address of the descriptor is non-canonical, the segment selector RPL is not equal to CPL, the segment is a nonwritable data segment, or DPL is not equal to CPL.
#SS(0)	If a memory operand effective address is non-canonical
#SS(Selector)	If the SS register is being loaded and the segment is marked not present.
#NP(selector)	If FS, or GS register is being loaded with a non-NULL segment selector and the segment is marked not present.
#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 source operand is not a memory location. If the LOCK prefix is used.

LOOP/LOOPcc—Loop According to ECX Counter

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
E2 <i>cb</i>	LOOP <i>rel8</i>	D	Valid	Valid	Decrement count; jump short if count \neq 0.
E1 <i>cb</i>	LOOPE <i>rel8</i>	D	Valid	Valid	Decrement count; jump short if count \neq 0 and ZF = 1.
E0 <i>cb</i>	LOOPNE <i>rel8</i>	D	Valid	Valid	Decrement count; jump short if count \neq 0 and ZF = 0.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
D	Offset	NA	NA	NA

Description

Performs a loop operation using the RCX, ECX or CX register as a counter (depending on whether address size is 64 bits, 32 bits, or 16 bits). Note that the LOOP instruction ignores REX.W; but 64-bit address size can be over-ridden using a 67H prefix.

Each time the LOOP instruction is executed, the count register is decremented, then checked for 0. If the count is 0, the loop is terminated and program execution continues with the instruction following the LOOP instruction. If the count is not zero, a near jump is performed to the destination (target) operand, which is presumably the instruction at the beginning of the loop.

The target instruction is specified with a relative offset (a signed offset relative to the current value of the instruction pointer in the IP/EIP/RIP register). This offset is generally specified as a label in assembly code, but at the machine code level, it is encoded as a signed, 8-bit immediate value, which is added to the instruction pointer. Offsets of -128 to $+127$ are allowed with this instruction.

Some forms of the loop instruction (LOOPcc) also accept the ZF flag as a condition for terminating the loop before the count reaches zero. With these forms of the instruction, a condition code (cc) is associated with each instruction to indicate the condition being tested for. Here, the LOOPcc instruction itself does not affect the state of the ZF flag; the ZF flag is changed by other instructions in the loop.

Operation

```

IF (AddressSize = 32)
  THEN Count is ECX;
ELSE IF (AddressSize = 64)
  Count is RCX;
ELSE Count is CX;
FI;

Count := Count - 1;

IF Instruction is not LOOP
  THEN
    IF (Instruction := LOOPE) or (Instruction := LOOPZ)
      THEN IF (ZF = 1) and (Count  $\neq$  0)
        THEN BranchCond := 1;
        ELSE BranchCond := 0;
      FI;
    ELSE (Instruction = LOOPNE) or (Instruction = LOOPNZ)
      IF (ZF = 0) and (Count  $\neq$  0)
        THEN BranchCond := 1;
        ELSE BranchCond := 0;

```

```

        FI;
    FI;
ELSE (* Instruction = LOOP *)
    IF (Count ≠ 0)
        THEN BranchCond := 1;
        ELSE BranchCond := 0;
    FI;
FI;

IF BranchCond = 1
    THEN
        IF in 64-bit mode (* OperandSize = 64 *)
            THEN
                tempRIP := RIP + SignExtend(DEST);
                IF tempRIP is not canonical
                    THEN #GP(0);
                ELSE RIP := tempRIP;
                FI;
            ELSE
                tempEIP := EIP + SignExtend(DEST);
                IF OperandSize = 16
                    THEN tempEIP := tempEIP AND 0000FFFFH;
                FI;
                IF tempEIP is not within code segment limit
                    THEN #GP(0);
                ELSE EIP := tempEIP;
                FI;
            FI;
        ELSE
            Terminate loop and continue program execution at (R/E)IP;
        FI;

```

Flags Affected

None

Protected Mode Exceptions

#GP(0) If the offset being jumped to is beyond the limits of the CS segment.
 #UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP If the offset being jumped to is beyond the limits of the CS segment or is outside of the effective address space from 0 to FFFFH. This condition can occur if a 32-bit address size override prefix is used.
 #UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the offset being jumped to is in a non-canonical form.
#UD	If the LOCK prefix is used.

LSL—Load Segment Limit

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 03 /r	LSL <i>r16, r16/m16</i>	RM	Valid	Valid	Load: <i>r16</i> := segment limit, selector <i>r16/m16</i> .
OF 03 /r	LSL <i>r32, r32/m16</i> *	RM	Valid	Valid	Load: <i>r32</i> := segment limit, selector <i>r32/m16</i> .
REX.W + OF 03 /r	LSL <i>r64, r32/m16</i> *	RM	Valid	Valid	Load: <i>r64</i> := segment limit, selector <i>r32/m16</i> .

NOTES:

* For all loads (regardless of destination sizing), only bits 16-0 are used. Other bits are ignored.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (<i>w</i>)	ModRM:r/m (<i>r</i>)	NA	NA

Description

Loads the unscrambled segment limit from the segment descriptor specified with the second operand (source operand) into the first operand (destination operand) and sets the ZF flag in the EFLAGS register. The source operand (which can be a register or a memory location) contains the segment selector for the segment descriptor being accessed. The destination operand is a general-purpose register.

The processor performs access checks as part of the loading process. Once loaded in the destination register, software can compare the segment limit with the offset of a pointer.

The segment limit is a 20-bit value contained in bytes 0 and 1 and in the first 4 bits of byte 6 of the segment descriptor. If the descriptor has a byte granular segment limit (the granularity flag is set to 0), the destination operand is loaded with a byte granular value (byte limit). If the descriptor has a page granular segment limit (the granularity flag is set to 1), the LSL instruction will translate the page granular limit (page limit) into a byte limit before loading it into the destination operand. The translation is performed by shifting the 20-bit "raw" limit left 12 bits and filling the low-order 12 bits with 1s.

When the operand size is 32 bits, the 32-bit byte limit is stored in the destination operand. When the operand size is 16 bits, a valid 32-bit limit is computed; however, the upper 16 bits are truncated and only the low-order 16 bits are loaded into the destination operand.

This instruction performs the following checks before it loads the segment limit into the destination register:

- Checks that the segment selector is not NULL.
- Checks that the segment selector points to a descriptor that is within the limits of the GDT or LDT being accessed
- Checks that the descriptor type is valid for this instruction. All code and data segment descriptors are valid for (can be accessed with) the LSL instruction. The valid special segment and gate descriptor types are given in the following table.
- If the segment is not a conforming code segment, the instruction checks that the specified segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector are less than or equal to the DPL of the segment selector).

If the segment descriptor cannot be accessed or is an invalid type for the instruction, the ZF flag is cleared and no value is loaded in the destination operand.

Table 3-56. Segment and Gate Descriptor Types

Type	Protected Mode		IA-32e Mode	
	Name	Valid	Name	Valid
0	Reserved	No	Reserved	No
1	Available 16-bit TSS	Yes	Reserved	No
2	LDT	Yes	LDT ¹	Yes
3	Busy 16-bit TSS	Yes	Reserved	No
4	16-bit call gate	No	Reserved	No
5	16-bit/32-bit task gate	No	Reserved	No
6	16-bit interrupt gate	No	Reserved	No
7	16-bit trap gate	No	Reserved	No
8	Reserved	No	Reserved	No
9	Available 32-bit TSS	Yes	64-bit TSS ¹	Yes
A	Reserved	No	Reserved	No
B	Busy 32-bit TSS	Yes	Busy 64-bit TSS ¹	Yes
C	32-bit call gate	No	64-bit call gate	No
D	Reserved	No	Reserved	No
E	32-bit interrupt gate	No	64-bit interrupt gate	No
F	32-bit trap gate	No	64-bit trap gate	No

NOTES:

1. In this case, the descriptor comprises 16 bytes; bits 12:8 of the upper 4 bytes must be 0.

Operation

```
IF SRC(Offset) > descriptor table limit
  THEN ZF := 0; FI;
```

Read segment descriptor;

```
IF SegmentDescriptor(Type) ≠ conforming code segment
and (CPL > DPL) OR (RPL > DPL)
or Segment type is not valid for instruction
```

```
  THEN
```

```
    ZF := 0;
```

```
  ELSE
```

```
    temp := SegmentLimit([SRC]);
```

```
    IF (SegmentDescriptor(G) = 1)
```

```
      THEN temp := (temp << 12) OR 00000FFFH;
```

```
    ELSE IF OperandSize = 32
```

```
      THEN DEST := temp; FI;
```

```
    ELSE IF OperandSize = 64 (* REX.W used *)
```

```
      THEN DEST := temp(* Zero-extended *); FI;
```

```
    ELSE (* OperandSize = 16 *)
```

```
      DEST := temp AND FFFFH;
```

```
    FI;
```

```
FI;
```

Flags Affected

The ZF flag is set to 1 if the segment limit is loaded successfully; otherwise, it is set to 0.

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 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 the memory operand effective address is unaligned while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#UD	The LSL instruction cannot be executed in real-address mode.
-----	--

Virtual-8086 Mode Exceptions

#UD	The LSL instruction cannot be executed in virtual-8086 mode.
-----	--

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If the memory operand effective address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory operand effective address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and the memory operand effective address is unaligned while the current privilege level is 3.
#UD	If the LOCK prefix is used.

6. Updates to Chapter 4, Volume 2B

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

Changes to this chapter include: Removed virtualization information from the PCONFIG instruction; this information is now documented correctly in Volume 3.

PCONFIG – Platform Configuration

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 C5 PCONFIG	A	V/V	PCONFIG	This instruction is used to execute functions for configuring platform features.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	NA	NA	NA	NA

Description

PCONFIG allows software to configure certain platform features. PCONFIG supports multiple leaf functions, with a leaf function identified by the value in EAX. The registers RBX, RCX, and RDX may provide input or output information for certain leaves. All leaves write status information to EAX but do not modify RBX, RCX, or RDX unless they are being used as leaf-specific output.

Each PCONFIG leaf function applies to a specific hardware block called a PCONFIG target, and each PCONFIG target is associated with a numerical **target identifier**. Supported target identifiers are enumerated, along with other PCONFIG capabilities, in the sub-leaves of the PCONFIG-information leaf of CPUID (EAX = 1BH). An attempt to execute an undefined leaf function, or a leaf function that applies to an unsupported target identifier, results in a general-protection exception (#GP). (In the future, the PCONFIG-information leaf of CPUID may enumerate PCONFIG capabilities in addition to the supported target identifiers.)

Addresses and operands are 32 bits outside 64-bit mode and are 64 bits in 64-bit mode. The value of CS.D does not affect operand size or address size.

Table 4-15 shows the leaf encodings for PCONFIG, and Table 4-16 shows the leaf register usage for PCONFIG.

Table 4-15. PCONFIG Leaf Encodings

Leaf	Encoding	Description
MKTME_KEY_PROGRAM	00000000H	This leaf is used to program the key and encryption mode associated with a KeyID.
RESERVED	00000001H - FFFFFFFFH	Reserved for future use (#GP(0) if used).

Table 4-16. PCONFIG Leaf Register Usage

Leaf	RBX	RCX	RDX
MKTME_KEY_PROGRAM	Input only.	Input only.	Input only.
RESERVED	Reserved for future use.	Reserved for future use.	Reserved for future use.

The MKTME_KEY_PROGRAM leaf of PCONFIG pertains to the MKTME¹ target, which has target identifier 1. It is used by software to manage the key associated with a KeyID. The leaf function is invoked by setting the leaf value of 0 in EAX and the address of MKTME_KEY_PROGRAM_STRUCT in RBX. Successful execution of the leaf clears RAX (set to zero) and ZF, CF, PF, AF, OF, and SF are cleared. In case of failure, the failure reason is indicated in RAX with ZF set to 1 and CF, PF, AF, OF, and SF are cleared. The MKTME_KEY_PROGRAM leaf uses the MKTME_KEY_PROGRAM_STRUCT in memory shown in Table 4-17.

1. Further details on MKTME usage can be found here:

<https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf>

Table 4-17. MKTME_KEY_PROGRAM_STRUCT Format

Field	Offset (bytes)	Size (bytes)	Comments
KEYID	0	2	Key Identifier.
KEYID_CTRL	2	4	KeyID control: <ul style="list-style-type: none"> ▪ Bits [7:0]: COMMAND. ▪ Bits [23:8]: ENC_ALG. ▪ Bits [31:24]: Reserved, must be zero.
RESERVED	6	58	Reserved, must be zero.
KEY_FIELD_1	64	64	Software supplied KeyID data key or entropy for KeyID data key.
KEY_FIELD_2	128	64	Software supplied KeyID tweak key or entropy for KeyID tweak key.

A description of each of the fields in MKTME_KEY_PROGRAM_STRUCT is provided below:

- **KEYID:** Key Identifier being programmed to the MKTME engine.
- **KEYID_CTRL:** The KEYID_CTRL field carries two sub-fields used by software to control the behavior of a KeyID: Command and KeyID encryption algorithm.

The command used controls the encryption mode for a KeyID. Table 4-18 provides a summary of the commands supported.

Table 4-18. Supported Key Programming Commands

Command	Encoding	Description
KEYID_SET_KEY_DIRECT	0	Software uses this mode to directly program a key for use with KeyID.
KEYID_SET_KEY_RANDOM	1	CPU generates and assigns an ephemeral key for use with a KeyID. Each time the instruction is executed, the CPU generates a new key using a hardware random number generator and the keys are discarded on reset.
KEYID_CLEAR_KEY	2	Clear the (software programmed) key associated with the KeyID. On execution of this command, the KeyID gets TME behavior (encrypt with platform TME key or bypass TME encryption).
KEYID_NO_ENCRYPT	3	Do not encrypt memory when this KeyID is in use.

The encryption algorithm field (ENC_ALG) allows software to select one of the activated encryption algorithms for the KeyID. The BIOS can activate a set of algorithms to allow for use when programming keys using the IA32_TME_ACTIVATE MSR (does not apply to KeyID 0 which uses the TME policy when TME encryption is not bypassed). The processor checks to ensure that the algorithm selected by software is one of the algorithms that has been activated by the BIOS.

- **KEY_FIELD_1:** This field carries the software supplied data key to be used for the KeyID if the direct key programming option is used (KEYID_SET_KEY_DIRECT). When the random key programming option is used (KEYID_SET_KEY_RANDOM), this field carries the software supplied entropy to be mixed in the CPU generated random data key. It is software's responsibility to ensure that the key supplied for the direct programming option or the entropy supplied for the random programming option does not result in weak keys. There are no explicit checks in the instruction to detect or prevent weak keys. When AES XTS-128 is used, the upper 48B are treated as reserved and must be zeroed out by software before executing the instruction.
- **KEY_FIELD_2:** This field carries the software supplied tweak key to be used for the KeyID if the direct key programming option is used (KEYID_SET_KEY_DIRECT). When the random key programming option is used (KEYID_SET_KEY_RANDOM), this field carries the software supplied entropy to be mixed in the CPU generated random tweak key. It is software's responsibility to ensure that the key supplied for the direct programming option or the entropy supplied for the random programming option does not result in weak keys. There are no explicit checks in the instruction to detect or prevent weak keys. When AES XTS-128 is used, the upper 48B are treated as reserved and must be zeroed out by software before executing the instruction.

All KeyIDs default to TME behavior (encrypt with TME key or bypass encryption) on MKTME activation. Software can at any point decide to change the key for a KeyID using the PCONFIG instruction. Change of

keys for a KeyID does NOT change the state of the TLB caches or memory pipeline. It is software's responsibility to take appropriate actions to ensure correct behavior.

Table 4-19 shows the return values associated with the MKTME_KEY_PROGRAM leaf of PCONFIG. On instruction execution, RAX is populated with the return value.

Table 4-19. Supported Key Error Codes

Return Value	Encoding	Description
PROG_SUCCESS	0	KeyID was successfully programmed.
INVALID_PROG_CMD	1	Invalid KeyID programming command.
ENTROPY_ERROR	2	Insufficient entropy.
INVALID_KEYID	3	KeyID not valid.
INVALID_ENC_ALG	4	Invalid encryption algorithm chosen (not supported).
DEVICE_BUSY	5	Failure to access key table.

PCONFIG Concurrency

In a scenario where the MKTME_KEY_PROGRAM leaf of PCONFIG is executed concurrently on multiple logical processors, only one logical processor will succeed in updating the key table. PCONFIG execution will return with an error code (DEVICE_BUSY) on other logical processors and software must retry. In cases where the instruction execution fails with a DEVICE_BUSY error code, the key table is not updated, thereby ensuring that either the key table is updated in its entirety with the information for a KeyID, or it is not updated at all. In order to accomplish this, the MKTME_KEY_PROGRAM leaf of PCONFIG maintains a writer lock for updating the key table. This lock is referred to as the Key table lock and denoted in the instruction flows as KEY_TABLE_LOCK. The lock can either be unlocked, when no logical processor is holding the lock (also the initial state of the lock) or be in an exclusive state where a logical processor is trying to update the key table. There can be only one logical processor holding the lock in exclusive state. The lock, being exclusive, can only be acquired when the lock is in unlocked state.

PCONFIG uses the following syntax to acquire KEY_TABLE_LOCK in exclusive mode and release the lock:

- KEY_TABLE_LOCK.ACQUIRE(WRITE)
- KEY_TABLE_LOCK.RELEASE()

Operation

Table 4-20. PCONFIG Operation Variables

Variable Name	Type	Size (Bytes)	Description
TMP_KEY_PROGRAM_STRUCT	MKTME_KEY_PROGRAM_STRUCT	192	Structure holding the key programming structure.
TMP_RND_DATA_KEY	UINT128	16	Random data key generated for random key programming option.
TMP_RND_TWEAK_KEY	UINT128	16	Random tweak key generated for random key programming option.

```
(* #UD if PCONFIG is not enumerated or CPL>0 *)
IF (CPUID.7.0:EDX[18] == 0 OR CPL > 0) #UD;
```

```
(* #GP(0) for an unsupported leaf *)
IF (EAX != 0) #GP(0)
```

```
(* KEY_PROGRAM leaf flow *)
```

```
IF (EAX == 0)
```

```
{
  (* #GP(0) if TME_ACTIVATE MSR is not locked or does not enable hardware encryption or multiple keys are not enabled *)
  IF (IA32_TME_ACTIVATE.LOCK != 1 OR IA32_TME_ACTIVATE.ENABLE != 1 OR IA32_TME_ACTIVATE.MK_TME_KEYID_BITS == 0)
  #GP(0)
```

```
(* Check MKTME_KEY_PROGRAM_STRUCT is 256B aligned *)
```

```
IF (DS:RBX is not 256B aligned) #GP(0);
```

```
(* Check that MKTME_KEY_PROGRAM_STRUCT is read accessible *)
```

```
<<DS: RBX should be read accessible>>
```

```
(* Copy MKTME_KEY_PROGRAM_STRUCT to a temporary variable *)
```

```
TMP_KEY_PROGRAM_STRUCT = DS:RBX.*;
```

```
(* RSVD field check *)
```

```
IF (TMP_KEY_PROGRAM_STRUCT.RSVD != 0) #GP(0);
```

```
IF (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.RSVD != 0) #GP(0);
```

```
IF (TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1.BYTES[63:16] != 0) #GP(0);
```

```
IF (TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2.BYTES[63:16] != 0) #GP(0);
```

```
(* Check for a valid command *)
```

```
IF (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.COMMAND is not a valid command)
```

```
{
  RFLAGS.ZF = 1;
  RAX = INVALID_PROG_CMD;
  goto EXIT;
}
```

```
(* Check that the KEYID being operated upon is a valid KEYID *)
```

```
IF (TMP_KEY_PROGRAM_STRUCT.KEYID >
      2^IA32_TME_ACTIVATE.MK_TME_KEYID_BITS - 1
  OR TMP_KEY_PROGRAM_STRUCT.KEYID >
      IA32_TME_CAPABILITY.MK_TME_MAX_KEYS
  OR TMP_KEY_PROGRAM_STRUCT.KEYID == 0)
```

```
{
  RFLAGS.ZF = 1;
  RAX = INVALID_KEYID;
  goto EXIT;
}
```

```
(* Check that only one algorithm is requested for the KeyID and it is one of the activated algorithms *)
```

```
IF (NUM_BITS(TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG) != 1 ||
    (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG &
     IA32_TME_ACTIVATE.MK_TME_CRYPTO_ALGS == 0))
```



```

{
    RFLAGS.ZF = 1;
    RAX = INVALID_ENC_ALG;
    goto EXIT;
}
(* Try to acquire exclusive lock *)
IF (NOT KEY_TABLE_LOCK.ACQUIRE(WRITE))
{
    //PCONFIG failure
    RFLAGS.ZF = 1;
    RAX = DEVICE_BUSY;
    goto EXIT;
}

(* Lock is acquired and key table will be updated as per the command
   Before this point no changes to the key table are made *)

switch(TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.COMMAND)
{
case KEYID_SET_KEY_DIRECT:
    <<Write
        DATA_KEY=TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1,
        TWEAK_KEY=TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2,
        ENCRYPTION_MODE=ENCRYPT_WITH_KEYID_KEY,
        to MKTME Key table at index TMP_KEY_PROGRAM_STRUCT.KEYID
    >>
    break;

case KEYID_SET_KEY_RANDOM:
    TMP_RND_DATA_KEY = <<Generate a random key using hardware RNG>>
    IF (NOT ENOUGH_ENTROPY)
    {
        RFLAGS.ZF = 1;
        RAX = ENTROPY_ERROR;
        goto EXIT;
    }
    TMP_RND_TWEAK_KEY = <<Generate a random key using hardware RNG>>
    IF (NOT ENOUGH_ENTROPY)
    {
        RFLAGS.ZF = 1;
        RAX = ENTROPY_ERROR;
        goto EXIT;
    }
    (* Mix user supplied entropy to the data key and tweak key *)
    TMP_RND_DATA_KEY = TMP_RND_KEY XOR
        TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1.BYTES[15:0];
    TMP_RND_TWEAK_KEY = TMP_RND_TWEAK_KEY XOR
        TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2.BYTES[15:0];

    <<Write
        DATA_KEY=TMP_RND_DATA_KEY,
        TWEAK_KEY=TMP_RND_TWEAK_KEY,
        ENCRYPTION_MODE=ENCRYPT_WITH_KEYID_KEY,
        to MKTME_KEY_TABLE at index TMP_KEY_PROGRAM_STRUCT.KEYID

```

```

    >>
    break;

case KEYID_CLEAR_KEY:
    <<Write
    DATA_KEY=0,
    TWEAK_KEY=0,
    ENCRYPTION_MODE = ENCRYPT_WITH_TME_KEY_OR_BYPASS,
    to MKTME_KEY_TABLE at index TMP_KEY_PROGRAM_STRUCT.KEYID
    >>

    break;
case KD_NO_ENCRYPT:
    <<Write
    ENCRYPTION_MODE=NO_ENCRYPTION,
    to MKTME_KEY_TABLE at index TMP_KEY_PROGRAM_STRUCT.KEYID
    >>
    break;
}

RAX = 0;
RFLAGS.ZF = 0;

//Release Lock
KEY_TABLE_LOCK(RELEASE);

EXIT:
RFLAGS.CF=0;
RFLAGS.PF=0;
RFLAGS.AF=0;
RFLAGS.OF=0;
RFLAGS.SF=0;
}

end_of_flow

```

Protected Mode Exceptions

#GP(0)	<p>If input value in EAX encodes an unsupported leaf.</p> <p>If IA32_TME_ACTIVATE MSR is not locked.</p> <p>If hardware encryption and MKTME capability are not enabled in IA32_TME_ACTIVATE MSR.</p> <p>If the memory operand is not 256B aligned.</p> <p>If any of the reserved bits in MKTME_KEY_PROGRAM_STRUCT are set.</p> <p>If a memory operand effective address is outside the DS segment limit.</p>
#PF(fault-code)	If a page fault occurs in accessing memory operands.
#UD	<p>If any of the LOCK/REP/OSIZE/VEX prefixes are used.</p> <p>If current privilege level is not 0.</p> <p>If CPUID.7.0:EDX[bit 18] = 0</p>

Real-Address Mode Exceptions

#GP	<p>If input value in EAX encodes an unsupported leaf.</p> <p>If IA32_TME_ACTIVATE MSR is not locked.</p> <p>If hardware encryption and MKTME capability are not enabled in IA32_TME_ACTIVATE MSR.</p> <p>If a memory operand is not 256B aligned.</p> <p>If any of the reserved bits in MKTME_KEY_PROGRAM_STRUCT are set.</p>
#UD	<p>If any of the LOCK/REP/OSIZE/VEX prefixes are used.</p> <p>If current privilege level is not 0.</p> <p>If CPUID.7.0:EDX.PCONFIG[bit 18] = 0</p>

Virtual-8086 Mode Exceptions

#UD	PCONFIG instruction is not recognized in virtual-8086 mode.
-----	---

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	<p>If input value in EAX encodes an unsupported leaf.</p> <p>If IA32_TME_ACTIVATE MSR is not locked.</p> <p>If hardware encryption and MKTME capability are not enabled in IA32_TME_ACTIVATE MSR.</p> <p>If a memory operand is not 256B aligned.</p> <p>If any of the reserved bits in MKTME_KEY_PROGRAM_STRUCT are set.</p> <p>If a memory operand is non-canonical form.</p>
#PF(fault-code)	If a page fault occurs in accessing memory operands.
#UD	<p>If any of the LOCK/REP/OSIZE/VEX prefixes are used.</p> <p>If the current privilege level is not 0.</p> <p>If CPUID.7.0:EDX.PCONFIG[bit 18] = 0.</p>

7. Updates to Chapter 5, Volume 2C

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

Changes to this chapter include :

- Typo correction in title of Table 5-2.
- Update to VERR/VERW pseudocode to clear ZF if segment not readable/writable.
- Update to VPGATHERDD/VPGATHERDQ and VPGATHERQD/VPGATHERQQ pseudocode to remove extraneous characters.
- Typo corrections in VSCALEFPD, VSCALEFPS, VSCALEFSD, and VSCALEFSS descriptions.
- Corrected memory operand to be "r, w" instead of "w" in XSAVE and XSAVEOPT.

VERR/VERW—Verify a Segment for Reading or Writing

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 00 /4	VERR <i>r/m16</i>	M	Valid	Valid	Set ZF=1 if segment specified with <i>r/m16</i> can be read.
OF 00 /5	VERW <i>r/m16</i>	M	Valid	Valid	Set ZF=1 if segment specified with <i>r/m16</i> can be written.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM: <i>r/m</i> (<i>r</i>)	NA	NA	NA

Description

Verifies whether the code or data segment specified with the source operand is readable (VERR) or writable (VERW) from the current privilege level (CPL). The source operand is a 16-bit register or a memory location that contains the segment selector for the segment to be verified. If the segment is accessible and readable (VERR) or writable (VERW), the ZF flag is set; otherwise, the ZF flag is cleared. Code segments are never verified as writable. This check cannot be performed on system segments.

To set the ZF flag, the following conditions must be met:

- The segment selector is not NULL.
- The selector must denote a descriptor within the bounds of the descriptor table (GDT or LDT).
- The selector must denote the descriptor of a code or data segment (not that of a system segment or gate).
- For the VERR instruction, the segment must be readable.
- For the VERW instruction, the segment must be a writable data segment.
- If the segment is not a conforming code segment, the segment's DPL must be greater than or equal to (have less or the same privilege as) both the CPL and the segment selector's RPL.

The validation performed is the same as is performed when a segment selector is loaded into the DS, ES, FS, or GS register, and the indicated access (read or write) is performed. The segment selector's value cannot result in a protection exception, enabling the software to anticipate possible segment access problems.

This instruction's operation is the same in non-64-bit modes and 64-bit mode. The operand size is fixed at 16 bits.

Operation

```
IF SRC(Offset) > (GDTR(Limit) or (LDTR(Limit)))
  THEN ZF := 0; FI;
```

Read segment descriptor;

```
IF SegmentDescriptor(DescriptorType) = 0 (* System segment *)
or (SegmentDescriptor(Type) ≠ conforming code segment)
and (CPL > DPL) or (RPL > DPL)
  THEN
    ZF := 0;
  ELSE
    IF ((Instruction = VERR) and (Segment readable))
    or ((Instruction = VERW) and (Segment writable))
      THEN
        ZF := 1;
      ELSE
        ZF := 0;
```

FI;
FI;

Flags Affected

The ZF flag is set to 1 if the segment is accessible and readable (VERR) or writable (VERW); otherwise, it is set to 0.

Protected Mode Exceptions

The only exceptions generated for these instructions are those related to illegal addressing of the source operand.

#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 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

#UD	The VERR and VERW instructions are not recognized in real-address mode. If the LOCK prefix is used.
-----	--

Virtual-8086 Mode Exceptions

#UD	The VERR and VERW instructions are not recognized in virtual-8086 mode. 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.

VPGATHERDD/VPGATHERDQ—Gather Packed Dword, Packed Qword with Signed Dword Indices

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F38.W0 90 /vsib VPGATHERDD xmm1 {k1}, vm32x	A	V/V	AVX512VL AVX512F	Using signed dword indices, gather dword values from memory using writemask k1 for merging-masking.
EVEX.256.66.0F38.W0 90 /vsib VPGATHERDD ymm1 {k1}, vm32y	A	V/V	AVX512VL AVX512F	Using signed dword indices, gather dword values from memory using writemask k1 for merging-masking.
EVEX.512.66.0F38.W0 90 /vsib VPGATHERDD zmm1 {k1}, vm32z	A	V/V	AVX512F	Using signed dword indices, gather dword values from memory using writemask k1 for merging-masking.
EVEX.128.66.0F38.W1 90 /vsib VPGATHERDQ xmm1 {k1}, vm32x	A	V/V	AVX512VL AVX512F	Using signed dword indices, gather quadword values from memory using writemask k1 for merging-masking.
EVEX.256.66.0F38.W1 90 /vsib VPGATHERDQ ymm1 {k1}, vm32x	A	V/V	AVX512VL AVX512F	Using signed dword indices, gather quadword values from memory using writemask k1 for merging-masking.
EVEX.512.66.0F38.W1 90 /vsib VPGATHERDQ zmm1 {k1}, vm32y	A	V/V	AVX512F	Using signed dword indices, gather quadword values from memory using writemask k1 for merging-masking.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	BaseReg (R): VSIB:base, VectorReg(R): VSIB:index	NA	NA

Description

A set of 16 or 8 doubleword/quadword memory locations pointed to by base address `BASE_ADDR` and index vector `VINDEX` with scale `SCALE` are gathered. The result is written into vector `zmm1`. The elements are specified via the `VSIB` (i.e., the index register is a `zmm`, holding packed indices). Elements will only be loaded if their corresponding mask bit is one. If an element's mask bit is not set, the corresponding element of the destination register (`zmm1`) is left unchanged. The entire mask register will be set to zero by this instruction unless it triggers an exception.

This instruction can be suspended by an exception if at least one element is already gathered (i.e., if the exception is triggered by an element other than the rightmost one with its mask bit set). When this happens, the destination register and the mask register (`k1`) are partially updated; those elements that have been gathered are placed into the destination register and have their mask bits set to zero. If any traps or interrupts are pending from already gathered elements, they will be delivered in lieu of the exception; in this case, `EFLAG.RF` is set to one so an instruction breakpoint is not re-triggered when the instruction is continued.

If the data element size is less than the index element size, the higher part of the destination register and the mask register do not correspond to any elements being gathered. This instruction sets those higher parts to zero. It may update these unused elements to one or both of those registers even if the instruction triggers an exception, and even if the instruction triggers the exception before gathering any elements.

Note that:

- The values may be read from memory in any order. Memory ordering with other instructions follows the Intel-64 memory-ordering model.
- Faults are delivered in a right-to-left manner. That is, if a fault is triggered by an element and delivered, all elements closer to the LSB of the destination `zmm` will be completed (and non-faulting). Individual elements closer to the MSB may or may not be completed. If a given element triggers multiple faults, they are delivered in the conventional order.
- Elements may be gathered in any order, but faults must be delivered in a right-to-left order; thus, elements to the left of a faulting one may be gathered before the fault is delivered. A given implementation of this instruction is repeatable - given the same input values and architectural state, the same set of elements to the left of the faulting one will be gathered.
- This instruction does not perform AC checks, and so will never deliver an AC fault.
- Not valid with 16-bit effective addresses. Will deliver a `#UD` fault.
- These instructions do not accept zeroing-masking since the 0 values in `k1` are used to determine completion.

Note that the presence of VSIB byte is enforced in this instruction. Hence, the instruction will #UD fault if ModRM.rm is different than 100b.

This instruction has the same $\text{disp}8*N$ and alignment rules as for scalar instructions (Tuple 1).

The instruction will #UD fault if the destination vector zmm1 is the same as index vector VINDEX. The instruction will #UD fault if the k0 mask register is specified.

The scaled index may require more bits to represent than the address bits used by the processor (e.g., in 32-bit mode, if the scale is greater than one). In this case, the most significant bits beyond the number of address bits are ignored.

Operation

BASE_ADDR stands for the memory operand base address (a GPR); may not exist

VINDEX stands for the memory operand vector of indices (a ZMM register)

SCALE stands for the memory operand scalar (1, 2, 4 or 8)

DISP is the optional 1 or 4 byte displacement

VPGATHERDD (EVEX encoded version)

(KL, VL) = (4, 128), (8, 256), (16, 512)

FOR j := 0 TO KL-1

 i := j * 32

 IF k1[j]

 THEN DEST[i+31:i] := MEM[BASE_ADDR +
 SignExtend(VINDEX[i+31:i]) * SCALE + DISP]

 k1[j] := 0

 ELSE *DEST[i+31:i] := remains unchanged* ; Only merging masking is allowed

 FI;

ENDFOR

k1[MAX_KL-1:KL] := 0

DEST[MAXVL-1:VL] := 0

VPGATHERDQ (EVEX encoded version)

(KL, VL) = (2, 128), (4, 256), (8, 512)

FOR j := 0 TO KL-1

 i := j * 64

 k := j * 32

 IF k1[j]

 THEN DEST[i+63:i] :=
 MEM[BASE_ADDR + SignExtend(VINDEX[k+31:k]) * SCALE + DISP]

 k1[j] := 0

 ELSE *DEST[i+63:i] := remains unchanged* ; Only merging masking is allowed

 FI;

ENDFOR

k1[MAX_KL-1:KL] := 0

DEST[MAXVL-1:VL] := 0

Intel C/C++ Compiler Intrinsic Equivalent

```

VPGATHERDD __m512i __mm512_i32gather_epi32( __m512i vdx, void * base, int scale);
VPGATHERDD __m512i __mm512_mask_i32gather_epi32(__m512i s, __mmask16 k, __m512i vdx, void * base, int scale);
VPGATHERDD __m256i __mm256_mask_i32gather_epi32(__m256i s, __mmask8 k, __m256i vdx, void * base, int scale);
VPGATHERDD __m128i __mm_mask_i32gather_epi32(__m128i s, __mmask8 k, __m128i vdx, void * base, int scale);
VPGATHERDQ __m512i __mm512_i32logather_epi64( __m256i vdx, void * base, int scale);
VPGATHERDQ __m512i __mm512_mask_i32logather_epi64(__m512i s, __mmask8 k, __m256i vdx, void * base, int scale);
VPGATHERDQ __m256i __mm256_mask_i32logather_epi64(__m256i s, __mmask8 k, __m128i vdx, void * base, int scale);
VPGATHERDQ __m128i __mm_mask_i32gather_epi64(__m128i s, __mmask8 k, __m128i vdx, void * base, int scale);

```

SIMD Floating-Point Exceptions

None

Other Exceptions

See Table 2-61, “Type E12 Class Exception Conditions”.

VPGATHERQD/VPGATHERQQ—Gather Packed Dword, Packed Qword with Signed Qword Indices

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F38.W0 91 /vsib VPGATHERQD xmm1 {k1}, vm64x	A	V/V	AVX512VL AVX512F	Using signed qword indices, gather dword values from memory using writemask k1 for merging-masking.
EVEX.256.66.0F38.W0 91 /vsib VPGATHERQD xmm1 {k1}, vm64y	A	V/V	AVX512VL AVX512F	Using signed qword indices, gather dword values from memory using writemask k1 for merging-masking.
EVEX.512.66.0F38.W0 91 /vsib VPGATHERQD ymm1 {k1}, vm64z	A	V/V	AVX512F	Using signed qword indices, gather dword values from memory using writemask k1 for merging-masking.
EVEX.128.66.0F38.W1 91 /vsib VPGATHERQQ xmm1 {k1}, vm64x	A	V/V	AVX512VL AVX512F	Using signed qword indices, gather quadword values from memory using writemask k1 for merging-masking.
EVEX.256.66.0F38.W1 91 /vsib VPGATHERQQ ymm1 {k1}, vm64y	A	V/V	AVX512VL AVX512F	Using signed qword indices, gather quadword values from memory using writemask k1 for merging-masking.
EVEX.512.66.0F38.W1 91 /vsib VPGATHERQQ zmm1 {k1}, vm64z	A	V/V	AVX512F	Using signed qword indices, gather quadword values from memory using writemask k1 for merging-masking.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	BaseReg (R): VSIB:base, VectorReg(R): VSIB:index	NA	NA

Description

A set of 8 doubleword/quadword memory locations pointed to by base address `BASE_ADDR` and index vector `VINDEX` with scale `SCALE` are gathered. The result is written into a vector register. The elements are specified via the `VSIB` (i.e., the index register is a vector register, holding packed indices). Elements will only be loaded if their corresponding mask bit is one. If an element's mask bit is not set, the corresponding element of the destination register is left unchanged. The entire mask register will be set to zero by this instruction unless it triggers an exception.

This instruction can be suspended by an exception if at least one element is already gathered (i.e., if the exception is triggered by an element other than the rightmost one with its mask bit set). When this happens, the destination register and the mask register (`k1`) are partially updated; those elements that have been gathered are placed into the destination register and have their mask bits set to zero. If any traps or interrupts are pending from already gathered elements, they will be delivered in lieu of the exception; in this case, `EFLAG.RF` is set to one so an instruction breakpoint is not re-triggered when the instruction is continued.

If the data element size is less than the index element size, the higher part of the destination register and the mask register do not correspond to any elements being gathered. This instruction sets those higher parts to zero. It may update these unused elements to one or both of those registers even if the instruction triggers an exception, and even if the instruction triggers the exception before gathering any elements.

Note that:

- The values may be read from memory in any order. Memory ordering with other instructions follows the Intel-64 memory-ordering model.
- Faults are delivered in a right-to-left manner. That is, if a fault is triggered by an element and delivered, all elements closer to the LSB of the destination `zmm` will be completed (and non-faulting). Individual elements closer to the MSB may or may not be completed. If a given element triggers multiple faults, they are delivered in the conventional order.
- Elements may be gathered in any order, but faults must be delivered in a right-to-left order; thus, elements to the left of a faulting one may be gathered before the fault is delivered. A given implementation of this instruction is repeatable - given the same input values and architectural state, the same set of elements to the left of the faulting one will be gathered.

- This instruction does not perform AC checks, and so will never deliver an AC fault.
- Not valid with 16-bit effective addresses. Will deliver a #UD fault.
- These instructions do not accept zeroing-masking since the 0 values in k1 are used to determine completion.

Note that the presence of VSIB byte is enforced in this instruction. Hence, the instruction will #UD fault if ModRM.rm is different than 100b.

This instruction has the same $\text{disp8} \cdot N$ and alignment rules as for scalar instructions (Tuple 1).

The instruction will #UD fault if the destination vector zmm1 is the same as index vector VINDEX. The instruction will #UD fault if the k0 mask register is specified.

The scaled index may require more bits to represent than the address bits used by the processor (e.g., in 32-bit mode, if the scale is greater than one). In this case, the most significant bits beyond the number of address bits are ignored.

Operation

BASE_ADDR stands for the memory operand base address (a GPR); may not exist

VINDEX stands for the memory operand vector of indices (a ZMM register)

SCALE stands for the memory operand scalar (1, 2, 4 or 8)

DISP is the optional 1 or 4 byte displacement

VPGATHERQD (EVEX encoded version)

(KL, VL) = (2, 128), (4, 256), (8, 512)

FOR j := 0 TO KL-1

 i := j * 32

 k := j * 64

 IF k1[jj]

 THEN DEST[i+31:i] := MEM[BASE_ADDR + (VINDEX[k+63:k]) * SCALE + DISP]

 k1[jj] := 0

 ELSE *DEST[i+31:i] := remains unchanged* ; Only merging masking is allowed

 FI;

ENDFOR

k1[MAX_KL-1:KL] := 0

DEST[MAXVL-1:VL/2] := 0

VPGATHERQQ (EVEX encoded version)

(KL, VL) = (2, 64), (4, 128), (8, 256)

FOR j := 0 TO KL-1

 i := j * 64

 IF k1[jj]

 THEN DEST[i+63:i] :=

 MEM[BASE_ADDR + (VINDEX[i+63:i]) * SCALE + DISP]

 k1[jj] := 0

 ELSE *DEST[i+63:i] := remains unchanged* ; Only merging masking is allowed

 FI;

ENDFOR

k1[MAX_KL-1:KL] := 0

DEST[MAXVL-1:VL] := 0

Intel C/C++ Compiler Intrinsic Equivalent

```

VPGATHERQD __m256i __mm512_i64gather_epi32(__m512i vdx, void * base, int scale);
VPGATHERQD __m256i __mm512_mask_i64gather_epi32lo(__m256i s, __mmask8 k, __m512i vdx, void * base, int scale);
VPGATHERQD __m128i __mm256_mask_i64gather_epi32lo(__m128i s, __mmask8 k, __m256i vdx, void * base, int scale);
VPGATHERQD __m128i __mm_mask_i64gather_epi32(__m128i s, __mmask8 k, __m128i vdx, void * base, int scale);
VPGATHERQQ __m512i __mm512_i64gather_epi64(__m512i vdx, void * base, int scale);
VPGATHERQQ __m512i __mm512_mask_i64gather_epi64(__m512i s, __mmask8 k, __m512i vdx, void * base, int scale);
VPGATHERQQ __m256i __mm256_mask_i64gather_epi64(__m256i s, __mmask8 k, __m256i vdx, void * base, int scale);
VPGATHERQQ __m128i __mm_mask_i64gather_epi64(__m128i s, __mmask8 k, __m128i vdx, void * base, int scale);

```

SIMD Floating-Point Exceptions

None

Other Exceptions

See Table 2-61, “Type E12 Class Exception Conditions”.

VSCALEFPD—Scale Packed Float64 Values With Float64 Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F38.W1 2C /r VSCALEFPD xmm1 {k1}{z}, xmm2, xmm3/m128/m64bcst	A	V/V	AVX512VL AVX512F	Scale the packed double-precision floating-point values in xmm2 using values from xmm3/m128/m64bcst. Under writemask k1.
EVEX.256.66.0F38.W1 2C /r VSCALEFPD ymm1 {k1}{z}, ymm2, ymm3/m256/m64bcst	A	V/V	AVX512VL AVX512F	Scale the packed double-precision floating-point values in ymm2 using values from ymm3/m256/m64bcst. Under writemask k1.
EVEX.512.66.0F38.W1 2C /r VSCALEFPD zmm1 {k1}{z}, zmm2, zmm3/m512/m64bcst{er}	A	V/V	AVX512F	Scale the packed double-precision floating-point values in zmm2 using values from zmm3/m512/m64bcst. Under writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Performs a floating-point scale of the packed double-precision floating-point values in the first source operand by multiplying them by 2 to the power of the double-precision floating-point values in second source operand.

The equation of this operation is given by:

$$zmm1 := zmm2 * 2^{\text{floor}(zmm3)}$$

Floor(zmm3) means maximum integer value \leq zmm3.

If the result cannot be represented in double precision, then the proper overflow response (for positive scaling operand), or the proper underflow response (for negative scaling operand) is issued. The overflow and underflow responses are dependent on the rounding mode (for IEEE-compliant rounding), as well as on other settings in MXCSR (exception mask bits, FTZ bit), and on the SAE bit.

The first source operand is a ZMM/YMM/XMM register. The second 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 64-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1.

Handling of special-case input values are listed in Table 5-21 and Table 5-22.

Table 5-21. VSCALEFPD/SD/PS/SS Special Cases

		Src2				Set IE
		\pm NaN	+Inf	-Inf	0/Denorm/Norm	
Src1	\pm QNaN	QNaN(Src1)	+INF	+0	QNaN(Src1)	IF either source is SNAN
	\pm SNaN	QNaN(Src1)	QNaN(Src1)	QNaN(Src1)	QNaN(Src1)	YES
	\pm Inf	QNaN(Src2)	Src1	QNaN_Indefinite	Src1	IF Src2 is SNAN or -INF
	\pm 0	QNaN(Src2)	QNaN_Indefinite	Src1	Src1	IF Src2 is SNAN or +INF
	Denorm/Norm	QNaN(Src2)	\pm INF (Src1 sign)	\pm 0 (Src1 sign)	Compute Result	IF Src2 is SNAN

Table 5-22. Additional VSCALEFPD/SD Special Cases

Special Case	Returned value	Faults
$ \text{result} < 2^{-1074}$	± 0 or $\pm \text{Min-Denormal}$ (Src1 sign)	Underflow
$ \text{result} \geq 2^{1024}$	$\pm \text{INF}$ (Src1 sign) or $\pm \text{Max-normal}$ (Src1 sign)	Overflow

Operation

```

SCALE(SRC1, SRC2)
{
  TMP_SRC2 := SRC2
  TMP_SRC1 := SRC1
  IF (SRC2 is denormal AND MXCSR.DAZ) THEN TMP_SRC2=0
  IF (SRC1 is denormal AND MXCSR.DAZ) THEN TMP_SRC1=0
  /* SRC2 is a 64 bits floating-point value */
  DEST[63:0] := TMP_SRC1[63:0] * POW(2, Floor(TMP_SRC2[63:0]))
}
VSCALEFPD (EVEX encoded versions)
(KL, VL) = (2, 128), (4, 256), (8, 512)
IF (VL = 512) AND (EVEX.b = 1) AND (SRC2 *is register*)
  THEN
    SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(EVEX.RC);
  ELSE
    SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(MXCSR.RC);
FI;
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] := SCALE(SRC1[i+63:i], SRC2[63:0]);
      ELSE DEST[i+63:i] := SCALE(SRC1[i+63:i], SRC2[i+63:i]);
    FI;
  ELSE
    IF *merging-masking* ; merging-masking
      THEN *DEST[i+63:i] remains unchanged*
      ELSE ; zeroing-masking
        DEST[i+63:i] := 0
    FI
  FI;
ENDFOR
DEST[MAXVL-1:VL] := 0

```

Intel C/C++ Compiler Intrinsic Equivalent

```

VSCALEFPD __m512d __mm512_scaleg_round_pd(__m512d a, __m512d b, int rounding);
VSCALEFPD __m512d __mm512_mask_scaleg_round_pd(__m512d s, __mmask8 k, __m512d a, __m512d b, int rounding);
VSCALEFPD __m512d __mm512_maskz_scaleg_round_pd(__mmask8 k, __m512d a, __m512d b, int rounding);
VSCALEFPD __m512d __mm512_scaleg_pd(__m512d a, __m512d b);
VSCALEFPD __m512d __mm512_mask_scaleg_pd(__m512d s, __mmask8 k, __m512d a, __m512d b);
VSCALEFPD __m512d __mm512_maskz_scaleg_pd(__mmask8 k, __m512d a, __m512d b);
VSCALEFPD __m256d __mm256_scaleg_pd(__m256d a, __m256d b);
VSCALEFPD __m256d __mm256_mask_scaleg_pd(__m256d s, __mmask8 k, __m256d a, __m256d b);
VSCALEFPD __m256d __mm256_maskz_scaleg_pd(__mmask8 k, __m256d a, __m256d b);
VSCALEFPD __m128d __mm_scaleg_pd(__m128d a, __m128d b);
VSCALEFPD __m128d __mm_mask_scaleg_pd(__m128d s, __mmask8 k, __m128d a, __m128d b);
VSCALEFPD __m128d __mm_maskz_scaleg_pd(__mmask8 k, __m128d a, __m128d b);

```

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal (for Src1).

Denormal is not reported for Src2.

Other Exceptions

See Table 2-46, “Type E2 Class Exception Conditions”.

VSCALEFPS—Scale Packed Float32 Values With Float32 Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F38.W0 2C /r VSCALEFPS xmm1 {k1}{z}, xmm2, xmm3/m128/m32bcst	A	V/V	AVX512VL AVX512F	Scale the packed single-precision floating-point values in xmm2 using values from xmm3/m128/m32bcst. Under writemask k1.
EVEX.256.66.0F38.W0 2C /r VSCALEFPS ymm1 {k1}{z}, ymm2, ymm3/m256/m32bcst	A	V/V	AVX512VL AVX512F	Scale the packed single-precision values in ymm2 using floating point values from ymm3/m256/m32bcst. Under writemask k1.
EVEX.512.66.0F38.W0 2C /r VSCALEFPS zmm1 {k1}{z}, zmm2, zmm3/m512/m32bcst{er}	A	V/V	AVX512F	Scale the packed single-precision floating-point values in zmm2 using floating-point values from zmm3/m512/m32bcst. Under writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Performs a floating-point scale of the packed single-precision floating-point values in the first source operand by multiplying them by 2 to the power of the float32 values in second source operand.

The equation of this operation is given by:

$$zmm1 := zmm2 * 2^{\text{floor}(zmm3)}$$

Floor(zmm3) means maximum integer value \leq zmm3.

If the result cannot be represented in single precision, then the proper overflow response (for positive scaling operand), or the proper underflow response (for negative scaling operand) is issued. The overflow and underflow responses are dependent on the rounding mode (for IEEE-compliant rounding), as well as on other settings in MXCSR (exception mask bits, FTZ bit), and on the SAE bit.

EVEX.512 encoded version: The first source operand is a ZMM register. The second source operand is a ZMM register, a 512-bit memory location or a 512-bit vector broadcasted from a 32-bit memory location. The destination operand is a ZMM register conditionally updated with writemask k1.

EVEX.256 encoded version: The first source operand is a YMM register. The second source operand is a YMM register, a 256-bit memory location, or a 256-bit vector broadcasted from a 32-bit memory location. The destination operand is a YMM register, conditionally updated using writemask k1.

EVEX.128 encoded version: The first source operand is an XMM register. The second source operand is a XMM register, a 128-bit memory location, or a 128-bit vector broadcasted from a 32-bit memory location. The destination operand is a XMM register, conditionally updated using writemask k1.

Handling of special-case input values are listed in Table 5-21 and Table 5-23.

Table 5-23. Additional VSCALEFPS/SS Special Cases

Special Case	Returned value	Faults
$ \text{result} < 2^{-149}$	± 0 or $\pm \text{Min-Denormal}$ (Src1 sign)	Underflow
$ \text{result} \geq 2^{128}$	$\pm \text{INF}$ (Src1 sign) or $\pm \text{Max-normal}$ (Src1 sign)	Overflow

Operation

```

SCALE(SRC1, SRC2)
{
    ; Check for denormal operands
    TMP_SRC2 := SRC2
    TMP_SRC1 := SRC1
    IF (SRC2 is denormal AND MXCSR.DAZ) THEN TMP_SRC2=0
    IF (SRC1 is denormal AND MXCSR.DAZ) THEN TMP_SRC1=0
    /* SRC2 is a 32 bits floating-point value */
    DEST[31:0] := TMP_SRC1[31:0] * POW(2, Floor(TMP_SRC2[31:0]))
}

```

VSCALEFPS (EVEX encoded versions)

(KL, VL) = (4, 128), (8, 256), (16, 512)

```

IF (VL = 512) AND (EVEX.b = 1) AND (SRC2 *is register*)
    THEN
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(EVEX.RC);
    ELSE
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(MXCSR.RC);
FI;
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] := SCALE(SRC1[i+31:i], SRC2[31:0]);
            ELSE DEST[i+31:i] := SCALE(SRC1[i+31:i], SRC2[i+31:i]);
        FI;
    ELSE
        IF *merging-masking* ; merging-masking
            THEN *DEST[i+31:i] remains unchanged*
            ELSE ; zeroing-masking
                DEST[i+31:i] := 0
            FI
    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0;

```

Intel C/C++ Compiler Intrinsic Equivalent

```

VSCALEFPS __m512 __mm512_scaleg_round_ps(__m512 a, __m512 b, int rounding);
VSCALEFPS __m512 __mm512_mask_scaleg_round_ps(__m512 s, __mmask16 k, __m512 a, __m512 b, int rounding);
VSCALEFPS __m512 __mm512_maskz_scaleg_round_ps(__mmask16 k, __m512 a, __m512 b, int rounding);
VSCALEFPS __m512 __mm512_scaleg_ps(__m512 a, __m512 b);
VSCALEFPS __m512 __mm512_mask_scaleg_ps(__m512 s, __mmask16 k, __m512 a, __m512 b);
VSCALEFPS __m512 __mm512_maskz_scaleg_ps(__mmask16 k, __m512 a, __m512 b);
VSCALEFPS __m256 __mm256_scaleg_ps(__m256 a, __m256 b);
VSCALEFPS __m256 __mm256_mask_scaleg_ps(__m256 s, __mmask8 k, __m256 a, __m256 b);
VSCALEFPS __m256 __mm256_maskz_scaleg_ps(__mmask8 k, __m256 a, __m256 b);
VSCALEFPS __m128 __mm_scaleg_ps(__m128 a, __m128 b);
VSCALEFPS __m128 __mm_mask_scaleg_ps(__m128 s, __mmask8 k, __m128 a, __m128 b);
VSCALEFPS __m128 __mm_maskz_scaleg_ps(__mmask8 k, __m128 a, __m128 b);

```

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal (for Src1).
Denormal is not reported for Src2.

Other Exceptions

See Table 2-46, “Type E2 Class Exception Conditions”.

VSCALEFSD—Scale Scalar Float64 Values With Float64 Values

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.LLIG.66.0F38.W1 2D /r VSCALEFSD xmm1 {k1}{z}, xmm2, xmm3/m64{er}	A	V/V	AVX512F	Scale the scalar double-precision floating-point values in xmm2 using the value from xmm3/m64. Under writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Performs a floating-point scale of the **scalar** double-precision floating-point value in the first source operand by multiplying it by 2 to the power of the double-precision floating-point value in second source operand.

The equation of this operation is given by:

$$\text{xmm1} := \text{xmm2} * 2^{\text{floor}(\text{xmm3})}$$

Floor(xmm3) means maximum integer value \leq xmm3.

If the result cannot be represented in double precision, then the proper overflow response (for positive scaling operand), or the proper underflow response (for negative scaling operand) is issued. The overflow and underflow responses are dependent on the rounding mode (for IEEE-compliant rounding), as well as on other settings in MXCSR (exception mask bits, FTZ bit), and on the SAE bit.

EVEX encoded version: The first source operand is an XMM register. The second source operand is an XMM register or a memory location. The destination operand is an XMM register conditionally updated with writemask k1.

Handling of special-case input values are listed in Table 5-21 and Table 5-22.

Operation

```
SCALE(SRC1, SRC2)
{
    ; Check for denormal operands
    TMP_SRC2 := SRC2
    TMP_SRC1 := SRC1
    IF (SRC2 is denormal AND MXCSR.DAZ) THEN TMP_SRC2=0
    IF (SRC1 is denormal AND MXCSR.DAZ) THEN TMP_SRC1=0
    /* SRC2 is a 64 bits floating-point value */
    DEST[63:0] := TMP_SRC1[63:0] * POW(2, Floor(TMP_SRC2[63:0]))
}
```

VSCALEFSD (EVEX encoded version)

```

IF (EVEX.b= 1) and SRC2 *is a register*
    THEN
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(EVEX.RC);
    ELSE
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(MXCSR.RC);
FI;
IF k1[0] OR *no writemask*
    THEN DEST[63:0] := SCALE(SRC1[63:0], SRC2[63:0])
    ELSE
        IF *merging-masking* ; merging-masking
            THEN *DEST[63:0] remains unchanged*
            ELSE ; zeroing-masking
                DEST[63:0] := 0
        FI
FI;
DEST[127:64] := SRC1[127:64]
DEST[MAXVL-1:128] := 0

```

Intel C/C++ Compiler Intrinsic Equivalent

```

VSCALEFSD __m128d __mm_scalef_round_sd(__m128d a, __m128d b, int);
VSCALEFSD __m128d __mm_mask_scalef_round_sd(__m128d s, __mmask8 k, __m128d a, __m128d b, int);
VSCALEFSD __m128d __mm_maskz_scalef_round_sd(__mmask8 k, __m128d a, __m128d b, int);

```

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal (for Src1).

Denormal is not reported for Src2.

Other Exceptions

See Table 2-47, “Type E3 Class Exception Conditions”.

VSCALEFSS—Scale Scalar Float32 Value With Float32 Value

Opcode/ Instruction	Op/ En	64/32 bitMode Support	CPUID Feature Flag	Description
EVEX.LLIG.66.0F38.W0 2D /r VSCALEFSS xmm1 {k1}{z}, xmm2, xmm3/m32{er}	A	V/V	AVX512F	Scale the scalar single-precision floating-point value in xmm2 using floating-point value from xmm3/m32. Under writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Performs a floating-point scale of the scalar single-precision floating-point value in the first source operand by multiplying it by 2 to the power of the float32 value in second source operand.

The equation of this operation is given by:

$$\text{xmm1} := \text{xmm2} * 2^{\text{floor}(\text{xmm3})}$$

Floor(xmm3) means maximum integer value \leq xmm3.

If the result cannot be represented in single precision, then the proper overflow response (for positive scaling operand), or the proper underflow response (for negative scaling operand) is issued. The overflow and underflow responses are dependent on the rounding mode (for IEEE-compliant rounding), as well as on other settings in MXCSR (exception mask bits, FTZ bit), and on the SAE bit.

EVEX encoded version: The first source operand is an XMM register. The second source operand is an XMM register or a memory location. The destination operand is an XMM register conditionally updated with writemask k1.

Handling of special-case input values are listed in Table 5-21 and Table 5-23.

Operation

```

SCALE(SRC1, SRC2)
{
    ; Check for denormal operands
    TMP_SRC2 := SRC2
    TMP_SRC1 := SRC1
    IF (SRC2 is denormal AND MXCSR.DAZ) THEN TMP_SRC2=0
    IF (SRC1 is denormal AND MXCSR.DAZ) THEN TMP_SRC1=0
    /* SRC2 is a 32 bits floating-point value */
    DEST[31:0] := TMP_SRC1[31:0] * POW(2, Floor(TMP_SRC2[31:0]))
}

```

VSCALEFSS (EVEX encoded version)

```

IF (EVEX.b= 1) and SRC2 *is a register*
    THEN
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(EVEX.RC);
    ELSE
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(MXCSR.RC);
FI;
IF k1[0] OR *no writemask*
    THEN DEST[31:0] := SCALE(SRC1[31:0], SRC2[31:0])
    ELSE
        IF *merging-masking* ; merging-masking
            THEN *DEST[31:0] remains unchanged*
            ELSE ; zeroing-masking
                DEST[31:0] := 0
        FI
FI;
DEST[127:32] := SRC1[127:32]
DEST[MAXVL-1:128] := 0

```

Intel C/C++ Compiler Intrinsic Equivalent

```

VSCALEFSS __m128 __mm_scalef_round_ss(__m128 a, __m128 b, int);
VSCALEFSS __m128 __mm_mask_scalef_round_ss(__m128 s, __mmask8 k, __m128 a, __m128 b, int);
VSCALEFSS __m128 __mm_maskz_scalef_round_ss(__mmask8 k, __m128 a, __m128 b, int);

```

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal (for Src1).
Denormal is not reported for Src2.

Other Exceptions

See Table 2-47, “Type E3 Class Exception Conditions”.

XSAVE—Save Processor Extended States

Opcode / Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP OF AE /4 XSAVE <i>mem</i>	M	V/V	XSAVE	Save state components specified by EDX:EAX to <i>mem</i> .
NP REX.W + OF AE /4 XSAVE64 <i>mem</i>	M	V/N.E.	XSAVE	Save state components specified by EDX:EAX to <i>mem</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (<i>r, w</i>)	NA	NA	NA

Description

Performs a full or partial save of processor state components to the XSAVE area located at the memory address specified by the destination operand. The implicit EDX:EAX register pair specifies a 64-bit instruction mask. The specific state components saved correspond to the bits set in the requested-feature bitmap (RFBM), which is the logical-AND of EDX:EAX and XCR0.

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

Section 13.7, “Operation of XSAVE,” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* provides a detailed description of the operation of the XSAVE instruction. The following items provide a high-level outline:

- XSAVE saves state component *i* if and only if $RFBM[i] = 1$.¹
- XSAVE does not modify bytes 511:464 of the legacy region of the XSAVE area (see Section 13.4.1, “Legacy Region of an XSAVE Area” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).
- XSAVE reads the XSTATE_BV field of the XSAVE header (see Section 13.4.2, “XSAVE Header” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*) and writes a modified value back to memory as follows. If $RFBM[i] = 1$, XSAVE writes XSTATE_BV[*i*] with the value of XINUSE[*i*]. (XINUSE is a bitmap by which the processor tracks the status of various state components. See Section 13.6, “Processor Tracking of XSAVE-Managed State” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.) If $RFBM[i] = 0$, XSAVE writes XSTATE_BV[*i*] with the value that it read from memory (it does not modify the bit). XSAVE does not write to any part of the XSAVE header other than the XSTATE_BV field.
- XSAVE always uses the standard format of the extended region of the XSAVE area (see Section 13.4.3, “Extended Region of an XSAVE Area” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).

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

1. An exception is made for MXCSR and MXCSR_MASK, which belong to state component 1 — SSE. XSAVE saves these values to memory if either RFBM[1] or RFBM[2] is 1.

Operation

RFBM := XCRO AND EDX:EAX; /* bitwise logical AND */

OLD_BV := XSTATE_BV field from XSAVE header;

IF RFBM[0] = 1

THEN store x87 state into legacy region of XSAVE area;

FI;

IF RFBM[1] = 1

THEN store XMM registers into legacy region of XSAVE area; // this step does not save MXCSR or MXCSR_MASK

FI;

IF RFBM[1] = 1 OR RFBM[2] = 1

THEN store MXCSR and MXCSR_MASK into legacy region of XSAVE area;

FI;

FOR i := 2 TO 62

IF RFBM[i] = 1

THEN save XSAVE state component i at offset n from base of XSAVE area (n enumerated by CPUID(EAX=0DH,ECX=i):EBX);

FI;

ENDFOR;

XSTATE_BV field in XSAVE header := (OLD_BV AND NOT RFBM) OR (XINUSE AND RFBM);

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XSAVE: void _xsave(void *, unsigned __int64);

XSAVE: void _xsave64(void *, unsigned __int64);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 64-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 64-byte boundary, regardless of segment. If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form. If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 64-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

XSAVEOPT—Save Processor Extended States Optimized

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP OF AE /6 XSAVEOPT <i>mem</i>	M	V/V	XSAVEOPT	Save state components specified by EDX:EAX to <i>mem</i> , optimizing if possible.
NP REX.W + OF AE /6 XSAVEOPT64 <i>mem</i>	M	V/V	XSAVEOPT	Save state components specified by EDX:EAX to <i>mem</i> , optimizing if possible.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (<i>r, w</i>)	NA	NA	NA

Description

Performs a full or partial save of processor state components to the XSAVE area located at the memory address specified by the destination operand. The implicit EDX:EAX register pair specifies a 64-bit instruction mask. The specific state components saved correspond to the bits set in the requested-feature bitmap (RFBM), which is the logical-AND of EDX:EAX and XCR0.

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

Section 13.9, “Operation of XSAVEOPT,” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* provides a detailed description of the operation of the XSAVEOPT instruction. The following items provide a high-level outline:

- Execution of XSAVEOPT is similar to that of XSAVE. XSAVEOPT differs from XSAVE in that it may use the init and modified optimizations. The performance of XSAVEOPT will be equal to or better than that of XSAVE.
- XSAVEOPT saves state component *i* only if $RFBM[i] = 1$ and $XINUSE[i] = 1$.¹ (XINUSE is a bitmap by which the processor tracks the status of various state components. See Section 13.6, “Processor Tracking of XSAVE-Managed State” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.) Even if both bits are 1, XSAVEOPT may optimize and not save state component *i* if (1) state component *i* has not been modified since the last execution of XRSTOR or XRSTORS; and (2) this execution of XSAVEOPT corresponds to that last execution of XRSTOR or XRSTORS as determined by the internal value XRSTOR_INFO (see the Operation section below).
- XSAVEOPT does not modify bytes 511:464 of the legacy region of the XSAVE area (see Section 13.4.1, “Legacy Region of an XSAVE Area” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).
- XSAVEOPT reads the XSTATE_BV field of the XSAVE header (see Section 13.4.2, “XSAVE Header” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*) and writes a modified value back to memory as follows. If $RFBM[i] = 1$, XSAVEOPT writes $XSTATE_BV[i]$ with the value of $XINUSE[i]$. If $RFBM[i] = 0$, XSAVEOPT writes $XSTATE_BV[i]$ with the value that it read from memory (it does not modify the bit). XSAVEOPT does not write to any part of the XSAVE header other than the XSTATE_BV field.
- XSAVEOPT always uses the standard format of the extended region of the XSAVE area (see Section 13.4.3, “Extended Region of an XSAVE Area” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).

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

1. There is an exception made for MXCSR and MXCSR_MASK, which belong to state component 1 — SSE. XSAVEOPT always saves these to memory if $RFBM[1] = 1$ or $RFBM[2] = 1$, regardless of the value of XINUSE.

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

Operation

```
RFBM := XCRO AND EDX:EAX; /* bitwise logical AND */
OLD_BV := XSTATE_BV field from XSAVE header;
TO_BE_SAVED := RFBM AND XINUSE;
```

IF in VMX non-root operation

```
    THEN VMXNR := 1;
    ELSE VMXNR := 0;
```

FI;

LAXA := linear address of XSAVE area;

```
IF XRSTOR_INFO = <CPL,VMXNR,LAXA,00000000_00000000H>
```

```
    THEN TO_BE_SAVED := TO_BE_SAVED AND XMODIFIED;
```

FI;

IF TO_BE_SAVED[0] = 1

```
    THEN store x87 state into legacy region of XSAVE area;
```

FI;

IF TO_BE_SAVED[1]

```
    THEN store XMM registers into legacy region of XSAVE area; // this step does not save MXCSR or MXCSR_MASK
```

FI;

IF RFBM[1] = 1 or RFBM[2] = 1

```
    THEN store MXCSR and MXCSR_MASK into legacy region of XSAVE area;
```

FI;

FOR i := 2 TO 62

```
    IF TO_BE_SAVED[i] = 1
```

```
        THEN save XSAVE state component i at offset n from base of XSAVE area (n enumerated by CPUID(EAX=0DH,ECX=i):EBX);
```

```
    FI;
```

ENDFOR;

XSTATE_BV field in XSAVE header := (OLD_BV AND NOT RFBM) OR (XINUSE AND RFBM);

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XSAVEOPT: void _xsaveopt(void *, unsigned __int64);

XSAVEOPT: void _xsaveopt64(void *, unsigned __int64);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSAVEOPT[bit 0] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 64-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 64-byte boundary, regardless of segment. If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSAVEOPT[bit 0] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

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. If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSAVEOPT[bit 0] = 0. If CR4.OSXSAVE[bit 18] = 0. If the LOCK prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 64-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

8. Updates to Chapter 4, Volume 3A

Change bars and green text show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter: Added clarifying reference to Section 4.6.1, "Determination of Access Rights" and a typo correction in section 4.10.3.1, "Caches for Paging Structures".

Chapter 3 explains how segmentation converts logical addresses to linear addresses. **Paging** (or linear-address translation) is the process of translating linear addresses so that they can be used to access memory or I/O devices. Paging translates each linear address to a **physical address** and determines, for each translation, what accesses to the linear address are allowed (the address's **access rights**) and the type of caching used for such accesses (the address's **memory type**).

Intel-64 processors support four different paging modes. These modes are identified and defined in Section 4.1. Section 4.2 gives an overview of the translation mechanism that is used in all modes. Section 4.3, Section 4.4, and Section 4.5 discuss the four paging modes in detail.

Section 4.6 details how paging determines and uses access rights. Section 4.7 discusses exceptions that may be generated by paging (page-fault exceptions). Section 4.8 considers data which the processor writes in response to linear-address accesses (accessed and dirty flags).

Section 4.9 describes how paging determines the memory types used for accesses to linear addresses. Section 4.10 provides details of how a processor may cache information about linear-address translation. Section 4.11 outlines interactions between paging and certain VMX features. Section 4.12 gives an overview of how paging can be used to implement virtual memory.

4.1 PAGING MODES AND CONTROL BITS

Paging behavior is controlled by the following control bits:

- The WP and PG flags in control register CR0 (bit 16 and bit 31, respectively).
- The PSE, PAE, PGE, LA57, PCIDE, SMEP, SMAP, PKE, CET, and PKS flags in control register CR4 (bit 4, bit 5, bit 7, bit 12, bit 17, bit 20, bit 21, bit 22, bit 23, and bit 24, respectively).
- The LME and NXE flags in the IA32_EFER MSR (bit 8 and bit 11, respectively).
- The AC flag in the EFLAGS register (bit 18).
- The “enable HLAT” VM-execution control (tertiary processor-based VM-execution control bit 1; see Section 24.6.2, “Processor-Based VM-Execution Controls,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*).

Software enables paging by using the MOV to CR0 instruction to set CR0.PG. Before doing so, software should ensure that control register CR3 contains the physical address of the first paging structure that the processor will use for linear-address translation (see Section 4.2) and that that structure is initialized as desired. See Table 4-3, Table 4-7, and Table 4-12 for the use of CR3 in the different paging modes.

Section 4.1.1 describes how the values of CR0.PG, CR4.PAE, CR4.LA57, and IA32_EFER.LME determine whether paging is enabled and, if so, which of four paging modes is in use. Section 4.1.2 explains how to manage these bits to establish or make changes in paging modes. Section 4.1.3 discusses how CR0.WP, CR4.PSE, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE modify the operation of the different paging modes.

4.1.1 Four Paging Modes

If CR0.PG = 0, paging is not used. The logical processor treats all linear addresses as if they were physical addresses. CR4.PAE, CR4.LA57, and IA32_EFER.LME are ignored by the processor, as are CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, CR4.SMAP, and IA32_EFER.NXE. (CR4.CET is also ignored insofar as it affects linear-address access rights.)

Paging is enabled if CR0.PG = 1. Paging can be enabled only if protection is enabled (CR0.PE = 1). If paging is enabled, one of four paging modes is used. The values of CR4.PAE, CR4.LA57, and IA32_EFER.LME determine which paging mode is used:

PAGING

- If CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 4.3. 32-bit paging uses CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, CR4.SMAP, and CR4.CET as described in Section 4.1.3 and Section 4.6.
- If CR4.PAE = 1 and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 4.4. PAE paging uses CR0.WP, CR4.PGE, CR4.SMEP, CR4.SMAP, CR4.CET, and IA32_EFER.NXE as described in Section 4.1.3 and Section 4.6.
- If CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 0, **4-level paging**¹ is used.² 4-level paging is detailed in Section 4.5 (along with 5-level paging). 4-level paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE as described in Section 4.1.3 and Section 4.6.
- If CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 1, **5-level paging** is used. 5-level paging is detailed in Section 4.5 (along with 4-level paging). 5-level paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE as described in Section 4.1.3 and Section 4.6.

NOTE

32-bit paging and PAE paging can be used only in legacy protected mode (IA32_EFER.LME = 0). In contrast, 4-level paging and 5-level paging can be used only IA-32e mode (IA32_EFER.LME = 1).

The four paging modes differ with regard to the following details:

- Linear-address width. The size of the linear addresses that can be translated.
- Physical-address width. The size of the physical addresses produced by paging.
- Page size. The granularity at which linear addresses are translated. Linear addresses on the same page are translated to corresponding physical addresses on the same page.
- Support for execute-disable access rights. In some paging modes, software can be prevented from fetching instructions from pages that are otherwise readable.
- Support for PCIDs. With 4-level paging and 5-level paging, software can enable a facility by which a logical processor caches information for multiple linear-address spaces. The processor may retain cached information when software switches between different linear-address spaces.
- Support for protection keys. With 4-level paging and 5-level paging, each linear address is associated with a **protection key**. Software can use the protection-key rights registers to disable, for each protection key, how certain accesses to linear addresses associated with that protection key.

Table 4-1 illustrates the principal differences between the four paging modes.

Table 4-1. Properties of Different Paging Modes

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	LA57 in CR4	Lin.-Addr. Width	Phys.-Addr. Width ¹	Page Sizes	Supports Execute-Disable?	Supports PCIDs and protection keys?
None	0	N/A	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0 ²	N/A	32	Up to 40 ³	4 KB 4 MB ⁴	No	No
PAE	1	1	0	N/A	32	Up to 52	4 KB 2 MB	Yes ⁵	No
4-level	1	1	1	0	48	Up to 52	4 KB 2 MB 1 GB ⁶	Yes ⁵	Yes ⁷

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

2. The LMA flag in the IA32_EFER MSR (bit 10) is a status bit that indicates whether the logical processor is in IA-32e mode (and thus uses either 4-level paging or 5-level paging). The processor always sets IA32_EFER.LMA to CR0.PG & IA32_EFER.LME. Software cannot directly modify IA32_EFER.LMA; an execution of WRMSR to the IA32_EFER MSR ignores bit 10 of its source operand.

Table 4-1. Properties of Different Paging Modes (Contd.)

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	LA57 in CR4	Lin.-Addr. Width	Phys.-Addr. Width ¹	Page Sizes	Supports Execute-Disable?	Supports PCIDs and protection keys?
5-level	1	1	1	1	57	Up to 52	4 KB 2 MB 1 GB ⁶	Yes ⁵	Yes ⁷

NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.4.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.4 and Section 4.3.
4. 32-bit paging uses 4-MByte pages only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 4.6.
6. Processors that support 4-level paging or 5-level paging do not necessarily support 1-GByte pages; see Section 4.1.4.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 4.10.1. Protection keys are used only if certain conditions hold; see Section 4.6.2.

Because 32-bit paging and PAE paging are used only in legacy protected mode and because legacy protected mode cannot produce linear addresses larger than 32 bits, 32-bit paging and PAE paging translate 32-bit linear addresses.

4-level paging and 5-level paging are used only in IA-32e mode. IA-32e mode has two sub-modes:

- Compatibility mode. This sub-mode uses only 32-bit linear addresses. In this sub-mode, 4-level paging and 5-level paging treat bits 63:32 of such an address as all 0.
- 64-bit mode. While this sub-mode produces 64-bit linear addresses, the processor enforces **canonicity**, meaning that the upper bits of such an address are identical: bits 63:47 for 4-level paging and bits 63:56 for 5-level paging. 4-level paging (respectively, 5-level paging) does not use bits 63:48 (respectively, bits 63:57) of such addresses.

4.1.2 Paging-Mode Enabling

If CR0.PG = 1, a logical processor is in one of four paging modes, depending on the values of CR4.PAE, IA32_EFER.LME, and CR4.LA57. Figure 4-1 illustrates how software can enable these modes and make transitions between them. The following items identify certain limitations and other details:

- IA32_EFER.LME cannot be modified while paging is enabled (CR0.PG = 1). Attempts to do so using WRMSR cause a general-protection exception (#GP(0)).
- Paging cannot be enabled (by setting CR0.PG to 1) while CR4.PAE = 0 and IA32_EFER.LME = 1. Attempts to do so using MOV to CR0 cause a general-protection exception (#GP(0)).
- One node in Figure 4-1 is labeled "IA-32e mode." This node represents either 4-level paging (if CR4.LA57 = 0) or 5-level paging (if CR4.LA57 = 1). As noted in the following items, software cannot modify CR4.LA57 (effecting transition between 4-level paging and 5-level paging) without first disabling paging.
- CR4.PAE and CR4.LA57 cannot be modified while either 4-level paging or 5-level paging is in use (when CR0.PG = 1 and IA32_EFER.LME = 1). Attempts to do so using MOV to CR4 cause a general-protection exception (#GP(0)).
- Regardless of the current paging mode, software can disable paging by clearing CR0.PG with MOV to CR0.¹
- Software can transition between 32-bit paging and PAE paging by changing the value of CR4.PAE with MOV to CR4.

1. If the logical processor is in 64-bit mode or if CR4.PCIDE = 1, an attempt to clear CR0.PG causes a general-protection exception (#GP). Software should transition to compatibility mode and clear CR4.PCIDE before attempting to disable paging.

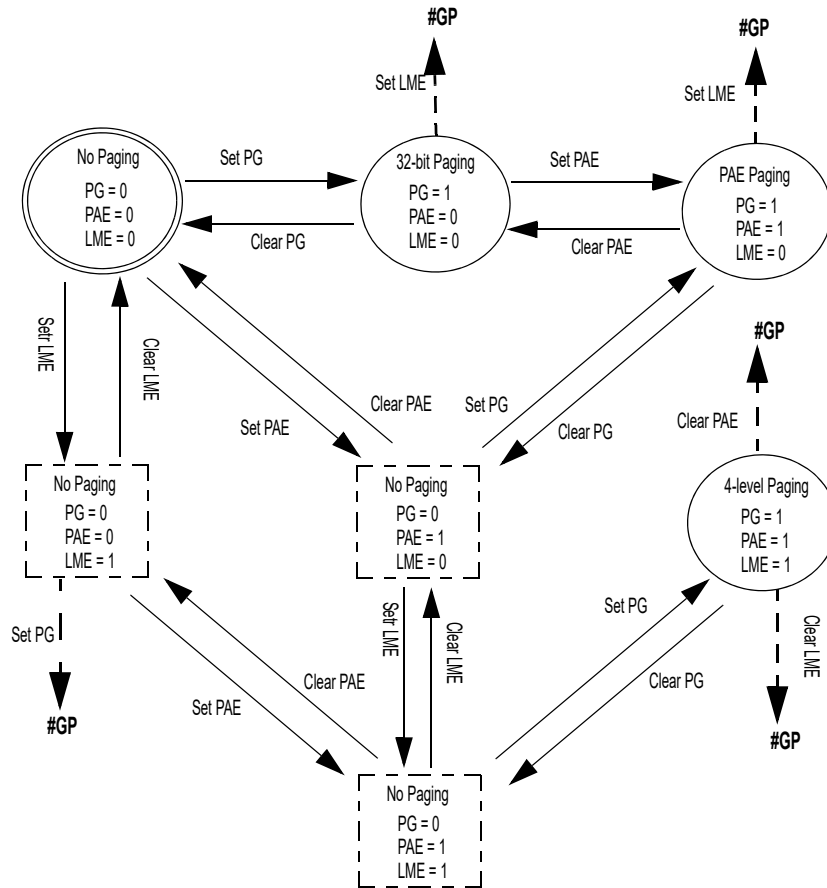


Figure 4-1. Enabling and Changing Paging Modes

- Software cannot transition directly between 4-level paging (or 5-level paging) and any of other paging mode. It must first disable paging (by clearing CR0.PG with MOV to CR0), then set CR4.PAE, IA32_EFER.LME, and CR4.LA57 to the desired values (with MOV to CR4 and WRMSR), and then re-enable paging (by setting CR0.PG with MOV to CR0). As noted earlier, an attempt to modify CR4.PAE, IA32_EFER.LME, or CR4.LA57 while 4-level paging or 5-level paging is enabled causes a general-protection exception (#GP(0)).
- VMX transitions allow transitions between paging modes that are not possible using MOV to CR or WRMSR. This is because VMX transitions can load CR0, CR4, and IA32_EFER in one operation. See Section 4.11.1.

4.1.3 Paging-Mode Modifiers

Details of how each paging mode operates are determined by the following control bits:

- The WP flag in CR0 (bit 16).
- The PSE, PGE, PCIDE, SMEP, SMAP, PKE, CET, and PKS flags in CR4 (bit 4, bit 7, bit 17, bit 20, bit 21, bit 22, bit 23, and bit 24, respectively).
- The NXE flag in the IA32_EFER MSR (bit 11).
- The “enable HLAT” VM-execution control (tertiary processor-based VM-execution control bit 1).

CR0.WP allows pages to be protected from supervisor-mode writes. If CR0.WP = 0, supervisor-mode write accesses are allowed to linear addresses with read-only access rights; if CR0.WP = 1, they are not. (User-mode write accesses are never allowed to linear addresses with read-only access rights, regardless of the value of

CR0.WP.) Section 4.6 explains how access rights are determined, including the definition of supervisor-mode and user-mode accesses.

CR4.PSE enables 4-MByte pages for 32-bit paging. If CR4.PSE = 0, 32-bit paging can use only 4-KByte pages; if CR4.PSE = 1, 32-bit paging can use both 4-KByte pages and 4-MByte pages. See Section 4.3 for more information. (PAE paging, 4-level paging, and 5-level paging can use multiple page sizes regardless of the value of CR4.PSE.)

CR4.PGE enables global pages. If CR4.PGE = 0, no translations are shared across address spaces; if CR4.PGE = 1, specified translations may be shared across address spaces. See Section 4.10.2.4 for more information.

CR4.PCIDE enables process-context identifiers (PCIDs) for 4-level paging and 5-level paging. PCIDs allow a logical processor to cache information for multiple linear-address spaces. See Section 4.10.1 for more information.

CR4.SMEP allows pages to be protected from supervisor-mode instruction fetches. If CR4.SMEP = 1, software operating in supervisor mode cannot fetch instructions from linear addresses that are accessible in user mode. Section 4.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

CR4.SMAP allows pages to be protected from supervisor-mode data accesses. If CR4.SMAP = 1, software operating in supervisor mode cannot access data at linear addresses that are accessible in user mode. Software can override this protection by setting EFLAGS.AC. Section 4.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

CR4.PKE and CR4.PKS enable specification of access rights based on **protection keys**. 4-level paging and 5-level paging associate each linear address with a protection key. When CR4.PKE = 1, the PKRU register specifies, for each protection key, whether user-mode linear addresses with that protection key can be read or written. When CR4.PKS = 1, the IA32_PKRS MSR does the same for supervisor-mode linear addresses. See Section 4.6 for more information.

CR4.CET enables **control-flow enforcement technology**, including the shadow-stack feature. If CR4.CET = 1, certain memory accesses are identified as **shadow-stack accesses** and certain linear addresses translate to **shadow-stack pages**. Section 4.6 explains how access rights are determined for these accesses and pages. (The processor allows CR4.CET to be set only if CR0.WP is also set.)

IA32_EFER.NXE enables execute-disable access rights for PAE paging, 4-level paging, and 5-level paging. If IA32_EFER.NXE = 1, instruction fetches can be prevented from specified linear addresses (even if data reads from the addresses are allowed). Section 4.6 explains how access rights are determined. (IA32_EFER.NXE has no effect with 32-bit paging. Software that wants to use this feature to limit instruction fetches from readable pages must use PAE paging, 4-level paging, or 5-level paging.)

The “enable HLAT” VM-execution control enables **HLAT paging** for 4-level paging and 5-level paging. HLAT paging does not use control register CR3 to identify the address of the first paging structure used for linear-address translation; instead, that structure is located using a field in the virtual-machine control structure (VMCS). In addition, HLAT paging interprets certain bits in paging-structure entries differently than ordinary paging. See Section 4.5 for details.

4.1.4 Enumeration of Paging Features by CPUID

Software can discover support for different paging features using the CPUID instruction:

- PSE: page-size extensions for 32-bit paging.
If CPUID.01H:EDX.PSE [bit 3] = 1, CR4.PSE may be set to 1, enabling support for 4-MByte pages with 32-bit paging (see Section 4.3).
- PAE: physical-address extension.
If CPUID.01H:EDX.PAE [bit 6] = 1, CR4.PAE may be set to 1, enabling PAE paging (this setting is also required for 4-level paging and 5-level paging).
- PGE: global-page support.
If CPUID.01H:EDX.PGE [bit 13] = 1, CR4.PGE may be set to 1, enabling the global-page feature (see Section 4.10.2.4).
- PAT: page-attribute table.
If CPUID.01H:EDX.PAT [bit 16] = 1, the 8-entry page-attribute table (PAT) is supported. When the PAT is

supported, three bits in certain paging-structure entries select a memory type (used to determine type of caching used) from the PAT (see Section 4.9.2).

- PSE-36: page-size extensions with 40-bit physical-address extension.
If CPUID.01H:EDX.PSE-36 [bit 17] = 1, the PSE-36 mechanism is supported, indicating that translations using 4-MByte pages with 32-bit paging may produce physical addresses with up to 40 bits (see Section 4.3).
- PCID: process-context identifiers.
If CPUID.01H:ECX.PCID [bit 17] = 1, CR4.PCIDE may be set to 1, enabling process-context identifiers (see Section 4.10.1).
- SMEP: supervisor-mode execution prevention.
If CPUID.(EAX=07H,ECX=0H):EBX.SMEP [bit 7] = 1, CR4.SMEP may be set to 1, enabling supervisor-mode execution prevention (see Section 4.6).
- SMAP: supervisor-mode access prevention.
If CPUID.(EAX=07H,ECX=0H):EBX.SMAP [bit 20] = 1, CR4.SMAP may be set to 1, enabling supervisor-mode access prevention (see Section 4.6).
- PKU: protection keys for user-mode pages.
If CPUID.(EAX=07H,ECX=0H):ECX.PKU [bit 3] = 1, CR4.PKE may be set to 1, enabling protection keys for user-mode pages (see Section 4.6).
- OSPKE: enabling of protection keys for user-mode pages.
CPUID.(EAX=07H,ECX=0H):ECX.OSPKE [bit 4] returns the value of CR4.PKE. Thus, protection keys for user-mode pages are enabled if this flag is 1 (see Section 4.6).
- CET: control-flow enforcement technology.
If CPUID.(EAX=07H,ECX=0H):ECX.CET_SS [bit 7] = 1, CR4.CET may be set to 1, enabling shadow-stack pages (see Section 4.6).
- LA57: 57-bit linear addresses and 5-level paging.
If CPUID.(EAX=07H,ECX=0):ECX.LA57 [bit 16] = 1, CR4.LA57 may be set to 1, enabling 5-level paging.
- PKS: protection keys for supervisor-mode pages.
If CPUID.(EAX=07H,ECX=0H):ECX.PKS [bit 31] = 1, CR4.PKS may be set to 1, enabling protection keys for supervisor-mode pages (see Section 4.6).
- NX: execute disable.
If CPUID.80000001H:EDX.NX [bit 20] = 1, IA32_EFER.NXE may be set to 1, allowing software to disable execute access to selected pages (see Section 4.6). (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.NXE to be set to 1.)
- Page1GB: 1-GByte pages.
If CPUID.80000001H:EDX.Page1GB [bit 26] = 1, 1-GByte pages may be supported with 4-level paging and 5-level paging (see Section 4.5).
- LM: IA-32e mode support.
If CPUID.80000001H:EDX.LM [bit 29] = 1, IA32_EFER.LME may be set to 1, enabling IA-32e mode (with either 4-level paging or 5-level paging). (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.LME to be set to 1.)
- CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. (For processors that do not support CPUID function 80000008H, the width is generally 36 if CPUID.01H:EDX.PAE [bit 6] = 1 and 32 otherwise.) This width is referred to as MAXPHYADDR. MAXPHYADDR is at most 52.
- CPUID.80000008H:EAX[15:8] reports the linear-address width supported by the processor. Generally, this value is reported as follows:
 - If CPUID.80000001H:EDX.LM [bit 29] = 0, the value is reported as 32.
 - If CPUID.80000001H:EDX.LM [bit 29] = 1 and CPUID.(EAX=07H,ECX=0):ECX.LA57 [bit 16] = 0, the value is reported as 48.
 - If CPUID.(EAX=07H,ECX=0):ECX.LA57 [bit 16] = 1, the value is reported as 57.
 (Processors that do not support CPUID function 80000008H, support a linear-address width of 32.)

4.2 HIERARCHICAL PAGING STRUCTURES: AN OVERVIEW

All four paging modes translate linear addresses using **hierarchical paging structures**. This section provides an overview of their operation. Section 4.3, Section 4.4, Section 4.5, and Section 4.6 provide details for the four paging modes.

Every paging structure is 4096 Bytes in size and comprises a number of individual **entries**. With 32-bit paging, each entry is 32 bits (4 bytes); there are thus 1024 entries in each structure. With the other paging modes, each entry is 64 bits (8 bytes); there are thus 512 entries in each structure. (PAE paging includes one exception, a paging structure that is 32 bytes in size, containing 4 64-bit entries.)

The processor uses the upper portion of a linear address to identify a series of paging-structure entries. The last of these entries identifies the physical address of the region to which the linear address translates (called the **page frame**). The lower portion of the linear address (called the **page offset**) identifies the specific address within that region to which the linear address translates.

Each paging-structure entry contains a physical address, which is either the address of another paging structure or the address of a page frame. In the first case, the entry is said to **reference** the other paging structure; in the latter, the entry is said to **map a page**.

The first paging structure used for any translation is located at the physical address in CR3.¹ A linear address is translated using the following iterative procedure. A portion of the linear address (initially the uppermost bits) selects an entry in a paging structure (initially the one located using CR3). If that entry references another paging structure, the process continues with that paging structure and with the portion of the linear address immediately below that just used. If instead the entry maps a page, the process completes: the physical address in the entry is that of the page frame and the remaining lower portion of the linear address is the page offset.

The following items give an example for each of the four paging modes (each example locates a 4-KByte page frame):

- With 32-bit paging, each paging structure comprises $1024 = 2^{10}$ entries. For this reason, the translation process uses 10 bits at a time from a 32-bit linear address. Bits 31:22 identify the first paging-structure entry and bits 21:12 identify a second. The latter identifies the page frame. Bits 11:0 of the linear address are the page offset within the 4-KByte page frame. (See Figure 4-2 for an illustration.)
- With PAE paging, the first paging structure comprises only $4 = 2^2$ entries. Translation thus begins by using bits 31:30 from a 32-bit linear address to identify the first paging-structure entry. Other paging structures comprise $512 = 2^9$ entries, so the process continues by using 9 bits at a time. Bits 29:21 identify a second paging-structure entry and bits 20:12 identify a third. This last identifies the page frame. (See Figure 4-5 for an illustration.)
- With 4-level paging, each paging structure comprises $512 = 2^9$ entries and translation uses 9 bits at a time from a 48-bit linear address. Bits 47:39 identify the first paging-structure entry, bits 38:30 identify a second, bits 29:21 a third, and bits 20:12 identify a fourth. Again, the last identifies the page frame. (See Figure 4-8 for an illustration.)
- 5-level paging is similar to 4-level paging except that 5-level paging translates 57-bit linear addresses. Bits 56:48 identify the first paging-structure entry, while the remaining bits are used as with 4-level paging.

The translation process in each of the examples above completes by identifying a page frame; the page frame is part of the **translation** of the original linear address. In some cases, however, the paging structures may be configured so that the translation process terminates before identifying a page frame. This occurs if the process encounters a paging-structure entry that is marked “not present” (because its P flag — bit 0 — is clear) or in which a reserved bit is set. In this case, there is no translation for the linear address; an access to that address causes a page-fault exception (see Section 4.7).

In the examples above, a paging-structure entry maps a page with a 4-KByte page frame when only 12 bits remain in the linear address; entries identified earlier always reference other paging structures. That may not apply in other cases. The following items identify when an entry maps a page and when it references another paging structure:

- If more than 12 bits remain in the linear address, bit 7 (PS — page size) of the current paging-structure entry is consulted. If the bit is 0, the entry references another paging structure; if the bit is 1, the entry maps a page.

1. If HLAT paging is in use, a different mechanism is used to identify the first paging structure. See Section 4.5 for more information.

PAGING

- If only 12 bits remain in the linear address, the current paging-structure entry always maps a page (bit 7 is used for other purposes).

If a paging-structure entry maps a page when more than 12 bits remain in the linear address, the entry identifies a page frame larger than 4 KBytes. For example, 32-bit paging uses the upper 10 bits of a linear address to locate the first paging-structure entry; 22 bits remain. If that entry maps a page, the page frame is 2^{22} Bytes = 4 MBytes. 32-bit paging can use 4-MByte pages if CR4.PSE = 1. The other paging modes can use 2-MByte pages (regardless of the value of CR4.PSE). 4-level paging and 5-level paging can use 1-GByte pages if the processor supports them (see Section 4.1.4).

Paging structures are given different names based on their uses in the translation process. Table 4-2 gives the names of the different paging structures. It also provides, for each structure, the source of the physical address used to locate it (CR3 or a different paging-structure entry); the bits in the linear address used to select an entry from the structure; and details of whether and how such an entry can map a page.

Table 4-2. Paging Structures in the Different Paging Modes

Paging Structure	Entry Name	Paging Mode	Physical Address of Structure	Bits Selecting Entry	Page Mapping
PML5 table	PML5E	32-bit, PAE, 4-level	N/A		
		5-level	CR3 ¹	56:48	N/A (PS must be 0)
PML4 table	PML4E	32-bit, PAE	N/A		
		4-level	CR3 ¹	47:39	N/A (PS must be 0)
		5-level	PML5E		
Page-directory-pointer table	PDPTE	32-bit	N/A		
		PAE	CR3	31:30	N/A (PS must be 0)
		4-level, 5-level	PML4E	38:30	1-GByte page if PS=1 ²
Page directory	PDE	32-bit	CR3	31:22	4-MByte page if PS=1 ³
		PAE, 4-level, 5-level	PDPTE	29:21	2-MByte page if PS=1
Page table	PTE	32-bit	PDE	21:12	4-KByte page
		PAE, 4-level, 5-level		20:12	

NOTES:

- If HLAT paging is in use, a different mechanism is used to identify the first paging structure. See Section 4.5 for more information.
- Not all processors support 1-GByte pages; see Section 4.1.4.
- 32-bit paging ignores the PS flag in a PDE (and uses the entry to reference a page table) unless CR4.PSE = 1. Not all processors support 4-MByte pages with 32-bit paging; see Section 4.1.4.

4.3 32-BIT PAGING

A logical processor uses 32-bit paging if $CR0.PG = 1$ and $CR4.PAE = 0$. 32-bit paging translates 32-bit linear addresses to 40-bit physical addresses.¹ Although 40 bits corresponds to 1 TByte, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

32-bit paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the page directory. Table 4-3 illustrates how CR3 is used with 32-bit paging.

32-bit paging may map linear addresses to either 4-KByte pages or 4-MByte pages. Figure 4-2 illustrates the translation process when it uses a 4-KByte page; Figure 4-3 covers the case of a 4-MByte page. The following items describe the 32-bit paging process in more detail as well as how the page size is determined:

- A 4-KByte naturally aligned page directory is located at the physical address specified in bits 31:12 of CR3 (see Table 4-3). A page directory comprises 1024 32-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
 - Bits 39:32 are all 0.
 - Bits 31:12 are from CR3.
 - Bits 11:2 are bits 31:22 of the linear address.
 - Bits 1:0 are 0.

Because a PDE is identified using bits 31:22 of the linear address, it controls access to a 4-Mbyte region of the linear-address space. Use of the PDE depends on $CR4.PSE$ and the PDE's PS flag (bit 7):

- If $CR4.PSE = 1$ and the PDE's PS flag is 1, the PDE maps a 4-MByte page (see Table 4-4). The final physical address is computed as follows:
 - Bits 39:32 are bits 20:13 of the PDE.
 - Bits 31:22 are bits 31:22 of the PDE.²
 - Bits 21:0 are from the original linear address.
- If $CR4.PSE = 0$ or the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 31:12 of the PDE (see Table 4-5). A page table comprises 1024 32-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
 - Bits 39:32 are all 0.
 - Bits 31:12 are from the PDE.
 - Bits 11:2 are bits 21:12 of the linear address.
 - Bits 1:0 are 0.
- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-6). The final physical address is computed as follows:
 - Bits 39:32 are all 0.
 - Bits 31:12 are from the PTE.
 - Bits 11:0 are from the original linear address.

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 4.7).

-
1. Bits in the range 39:32 are 0 in any physical address used by 32-bit paging except those used to map 4-MByte pages. If the processor does not support the PSE-36 mechanism, this is true also for physical addresses used to map 4-MByte pages. If the processor does support the PSE-36 mechanism and $MAXPHYADDR < 40$, bits in the range 39:MAXPHYADDR are 0 in any physical address used to map a 4-MByte page. (The corresponding bits are reserved in PDEs.) See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
 2. The upper bits in the final physical address do not all come from corresponding positions in the PDE; the physical-address bits in the PDE are not all contiguous.

PAGING

With 32-bit paging, there are reserved bits only if CR4.PSE = 1:

- If the P flag and the PS flag (bit 7) of a PDE are both 1, the bits reserved depend on MAXPHYADDR, and whether the PSE-36 mechanism is supported:¹
 - If the PSE-36 mechanism is not supported, bits 21:13 are reserved.
 - If the PSE-36 mechanism is supported, bits 21:(M-19) are reserved, where M is the minimum of 40 and MAXPHYADDR.
- If the PAT is not supported:²
 - If the P flag of a PTE is 1, bit 7 is reserved.
 - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

(If CR4.PSE = 0, no bits are reserved with 32-bit paging.)

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

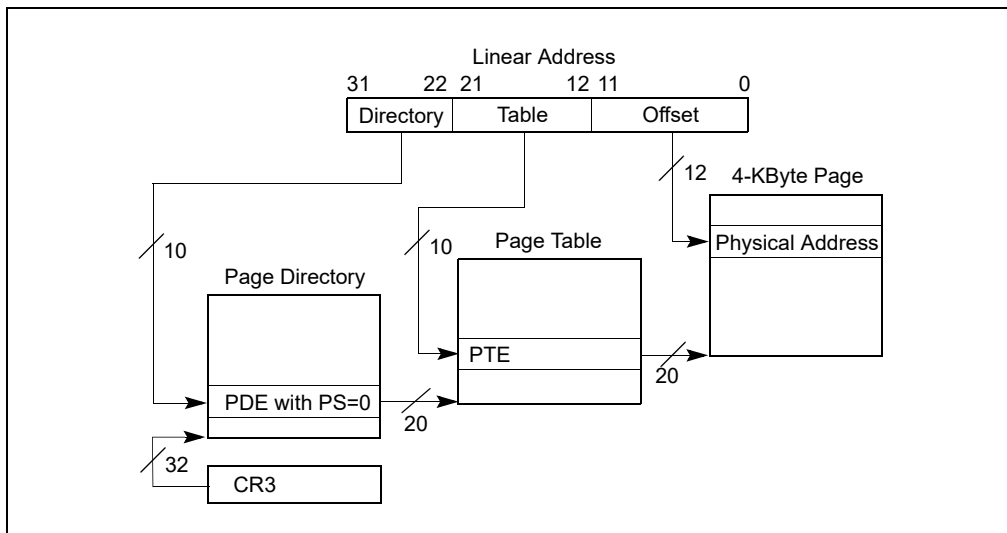


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

1. See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
2. See Section 4.1.4 for how to determine whether the PAT is supported.

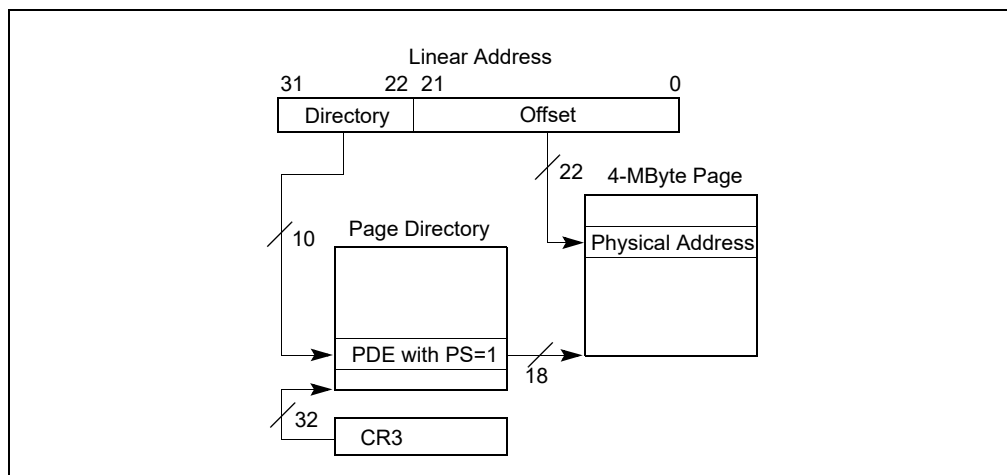


Figure 4-3. Linear-Address Translation to a 4-MByte Page using 32-Bit Paging

Figure 4-4 gives a summary of the formats of CR3 and the paging-structure entries with 32-bit paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are “not present”; bit 0 (P) and bit 7 (PS) are highlighted because they determine how such an entry is used.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Address of page directory ¹												Ignored						P	P	Ignored				CR3									
Bits 31:22 of address of 4MB page frame						Reserved (must be 0)						P	A	T	Ignored	G	1	D	A	P	P	U	R	C	W	/	/	1	PDE: 4MB page				
Address of page table												Ignored						0	I	g	n	A	P	P	U	R	C	W	/	/	1	PDE: page table	
Ignored																0				PDE: not present													
Address of 4KB page frame												Ignored						G	P	A	T	D	A	P	P	U	R	C	W	/	/	1	PTE: 4KB page
Ignored																0				PTE: not present													

Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

NOTES:

1. CR3 has 64 bits on processors supporting the Intel-64 architecture. These bits are ignored with 32-bit paging.
2. This example illustrates a processor in which MAXPHYADDR is 36. If this value is larger or smaller, the number of bits reserved in positions 20:13 of a PDE mapping a 4-MByte page will change.

Table 4-3. Use of CR3 with 32-Bit Paging

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 4.9)
11:5	Ignored
31:12	Physical address of the 4-KByte aligned page directory used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

Table 4-4. Format of a 32-Bit Page-Directory Entry that Maps a 4-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-5)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0) ¹
(M-20):13	Bits (M-1):32 of physical address of the 4-MByte page referenced by this entry ²
21:(M-19)	Reserved (must be 0)
31:22	Bits 31:22 of physical address of the 4-MByte page referenced by this entry

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.
2. If the PSE-36 mechanism is not supported, M is 32, and this row does not apply. If the PSE-36 mechanism is supported, M is the minimum of 40 and MAXPHYADDR (this row does not apply if MAXPHYADDR = 32). See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.

Table 4-5. Format of a 32-Bit Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	If CR4.PSE = 1, must be 0 (otherwise, this entry maps a 4-MByte page; see Table 4-4); otherwise, ignored
11:8	Ignored
31:12	Physical address of 4-KByte aligned page table referenced by this entry

Table 4-6. Format of a 32-Bit Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0) ¹
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
31:12	Physical address of the 4-KByte page referenced by this entry

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.

4.4 PAE PAGING

A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. PAE paging translates 32-bit linear addresses to 52-bit physical addresses.¹ Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

With PAE paging, a logical processor maintains a set of four (4) PDPTE registers, which are loaded from an address in CR3. Linear address are translated using 4 hierarchies of in-memory paging structures, each located using one of the PDPTE registers. (This is different from the other paging modes, in which there is one hierarchy referenced by CR3.)

Section 4.4.1 discusses the PDPTE registers. Section 4.4.2 describes linear-address translation with PAE paging.

4.4.1 PDPTE Registers

When PAE paging is used, CR3 references the base of a 32-Byte **page-directory-pointer table**. Table 4-7 illustrates how CR3 is used with PAE paging.

Table 4-7. Use of CR3 with PAE Paging

Bit Position(s)	Contents
4:0	Ignored
31:5	Physical address of the 32-Byte aligned page-directory-pointer table used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

The page-directory-pointer-table comprises four (4) 64-bit entries called PDPTEs. Each PDPTE controls access to a 1-GByte region of the linear-address space. Corresponding to the PDPTEs, the logical processor maintains a set of four (4) internal, non-architectural PDPTE registers, called PDPTE0, PDPTE1, PDPTE2, and PDPTE3. The logical processor loads these registers from the PDPTEs in memory as part of certain operations:

- If PAE paging would be in use following an execution of MOV to CR0 or MOV to CR4 (see Section 4.1.1) and the instruction is modifying any of CR0.CD, CR0.NW, CR0.PG, CR4.PAE, CR4.PGE, CR4.PSE, or CR4.SMEP; then the PDPTEs are loaded from the address in CR3.
- If MOV to CR3 is executed while the logical processor is using PAE paging, the PDPTEs are loaded from the address being loaded into CR3.
- If PAE paging is in use and a task switch changes the value of CR3, the PDPTEs are loaded from the address in the new CR3 value.
- Certain VMX transitions load the PDPTE registers. See Section 4.11.1.

Table 4-8 gives the format of a PDPTE. If any of the PDPTEs sets both the P flag (bit 0) and any reserved bit, the MOV to CR instruction causes a general-protection exception (#GP(0)) and the PDPTEs are not loaded.² As shown in Table 4-8, bits 2:1, 8:5, and 63:MAXPHYADDR are reserved in the PDPTEs.

1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by PAE paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.

2. On some processors, reserved bits are checked even in PDPTEs in which the P flag (bit 0) is 0.

Table 4-8. Format of a PAE Page-Directory-Pointer-Table Entry (PDPTE)

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
2:1	Reserved (must be 0)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
8:5	Reserved (must be 0)
11:9	Ignored
(M-1):12	Physical address of 4-KByte aligned page directory referenced by this entry ¹
63:M	Reserved (must be 0)

NOTES:

1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

4.4.2 Linear-Address Translation with PAE Paging

PAE paging may map linear addresses to either 4-KByte pages or 2-MByte pages. Figure 4-5 illustrates the translation process when it produces a 4-KByte page; Figure 4-6 covers the case of a 2-MByte page. The following items describe the PAE paging process in more detail as well as how the page size is determined:

- Bits 31:30 of the linear address select a PDPTE register (see Section 4.4.1); this is PDPTE_{*i*}, where *i* is the value of bits 31:30.¹ Because a PDPTE register is identified using bits 31:30 of the linear address, it controls access to a 1-GByte region of the linear-address space. If the P flag (bit 0) of PDPTE_{*i*} is 0, the processor ignores bits 63:1, and there is no mapping for the 1-GByte region controlled by PDPTE_{*i*}. A reference using a linear address in this region causes a page-fault exception (see Section 4.7).
- If the P flag of PDPTE_{*i*} is 1, 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of PDPTE_{*i*} (see Table 4-8 in Section 4.4.1). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
 - Bits 51:12 are from PDPTE_{*i*}.
 - Bits 11:3 are bits 29:21 of the linear address.
 - Bits 2:0 are 0.

Because a PDE is identified using bits 31:21 of the linear address, it controls access to a 2-Mbyte region of the linear-address space. Use of the PDE depends on its PS flag (bit 7):

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table 4-9). The final physical address is computed as follows:
 - Bits 51:21 are from the PDE.
 - Bits 20:0 are from the original linear address.
- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table 4-10). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
 - Bits 51:12 are from the PDE.

1. With PAE paging, the processor does not use CR3 when translating a linear address (as it does in the other paging modes). It does not access the PDPTEs in the page-directory-pointer table during linear-address translation.

PAGING

- Bits 11:3 are bits 20:12 of the linear address.
- Bits 2:0 are 0.
- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-11). The final physical address is computed as follows:
 - Bits 51:12 are from the PTE.
 - Bits 11:0 are from the original linear address.

If the P flag (bit 0) of a PDE or a PTE is 0 or if a PDE or a PTE sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 4.7).

The following bits are reserved with PAE paging:

- If the P flag (bit 0) of a PDE or a PTE is 1, bits 62:MAXPHYADDR are reserved.
- If the P flag and the PS flag (bit 7) of a PDE are both 1, bits 20:13 are reserved.
- If IA32_EFER.NXE = 0 and the P flag of a PDE or a PTE is 1, the XD flag (bit 63) is reserved.
- If the PAT is not supported:¹
 - If the P flag of a PTE is 1, bit 7 is reserved.
 - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

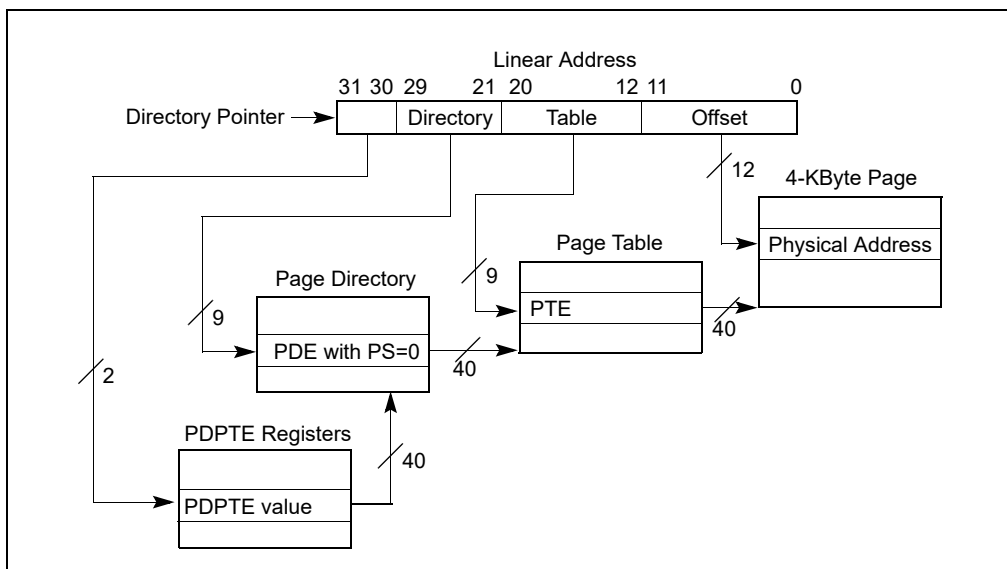


Figure 4-5. Linear-Address Translation to a 4-KByte Page using PAE Paging

1. See Section 4.1.4 for how to determine whether the PAT is supported.

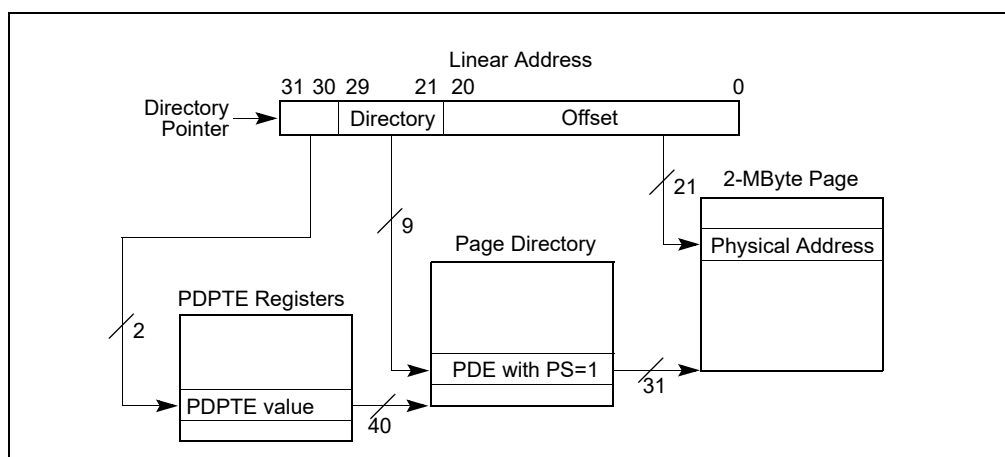


Figure 4-6. Linear-Address Translation to a 2-MByte Page using PAE Paging

Table 4-9. Format of a PAE Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-10)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0) ¹
20:13	Reserved (must be 0)
(M-1):21	Physical address of the 2-MByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.

Table 4-10. Format of a PAE Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 4-9)
11:8	Ignored
(M-1):12	Physical address of 4-KByte aligned page table referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-11. Format of a PAE Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0) ¹
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise

Table 4-11. Format of a PAE Page-Table Entry that Maps a 4-KByte Page (Contd.)

Bit Position(s)	Contents
11:9	Ignored
(M-1):12	Physical address of the 4-KByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.

Figure 4-7 gives a summary of the formats of CR3 and the paging-structure entries with PAE paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are “not present”; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ignored ²												Address of page-directory-pointer table												Ignored				CR3																																						
Reserved ³												Address of page directory												Ign.	Rsvd.	P C D	P W T	R s v d	1	PDPTE: present																																				
Ignored												Ignored												Ignored				0	PDPTE: not present																																					
X D	Reserved												Address of 2MB page frame												Reserved	P A T	Ign.	G	1	D	A	P C D	P W T	U / S	R / W	1	PDE: 2MB page																													
X D	Reserved												Address of page table												Ign.	Ign.	0	A	P C D	P W T	U / S	R / W	1	PDE: page table																																
Ignored												Ignored												Ignored				0	PDE: not present																																					
X D	Reserved												Address of 4KB page frame												Ign.	G	P A T	D	A	P C D	P W T	U / S	R / W	1	PTE: 4KB page																															
Ignored												Ignored												Ignored				0	PTE: not present																																					

Figure 4-7. Formats of CR3 and Paging-Structure Entries with PAE Paging

NOTES:

1. M is an abbreviation for MAXPHYADDR.
2. CR3 has 64 bits only on processors supporting the Intel-64 architecture. These bits are ignored with PAE paging.
3. Reserved fields must be 0.
4. If IA32_EFER.NXE = 0 and the P flag of a PDE or a PTE is 1, the XD flag (bit 63) is reserved.

4.5 4-LEVEL PAGING AND 5-LEVEL PAGING

Because the operation of 4-level paging and 5-level paging is very similar, they are described together in this section. The following items highlight the distinctions between the two paging modes:

- A logical processor uses 4-level paging if $CR0.PG = 1$, $CR4.PAE = 1$, $IA32_EFER.LME = 1$, and $CR4.LA57 = 0$. 4-level paging translates 48-bit linear addresses to 52-bit physical addresses.¹ Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time.
- A logical processor uses 5-level paging if $CR0.PG = 1$, $CR4.PAE = 1$, $IA32_EFER.LME = 1$, and $CR4.LA57 = 1$. 5-level paging translates 57-bit linear addresses to 52-bit physical addresses. Thus, 5-level paging supports a linear-address space sufficient to access the entire physical-address space.

4.5.1 Ordinary Paging and HLAT Paging

There are two forms of 4-level paging and 5-level paging that differ principally with regard to how linear-address translation identifies the first paging structure.

The normal form is called **ordinary paging**, and it uses CR3 to locate the first paging structure, similar to what is done for 32-bit paging. Section 4.5.2 provides details of this use of CR3.

An alternative form of paging may be used with the VMX feature called hypervisor-managed linear-address translation (HLAT). Called **HLAT paging**, this form is used only in VMX non-root operation and only if the “enable HLAT” VM-execution control is 1.² HLAT paging locates the first paging structure using a VM-execution control field in the VMCS called the **HLAT pointer (HLATP)**. Section 4.5.3 provides details.

Whether HLAT paging is used to translate a specific linear address depends on the address and on the value of a VM-execution control field in the VMCS called the **HLAT prefix size**:

- If the HLAT prefix size is zero, every linear address is translated using HLAT paging.
- If the HLAT prefix size is not zero, a linear address is translated using HLAT paging if bit 63 of the address is 1.³ The address is translated using ordinary paging if bit 63 of the address is 0.

In some cases, HLAT paging may specify that a translation of a linear address must be restarted. When this occurs, the linear address is then translated using ordinary paging (starting with a paging structure identified using CR3). The situations leading to this restart are detailed in Section 4.5.4, and additional details of the restart process are given in Section 4.5.5.

4.5.2 Use of CR3 with Ordinary 4-Level Paging and 5-Level Paging

Ordinary 4-level paging and 5-level paging each translate linear addresses using a hierarchy of in-memory paging structures located using the contents of CR3, which is used to locate the first paging structure. For 4-level paging, this is the PML4 table, and for 5-level paging it is the PML5 table. Use of CR3 with 4-level paging and 5-level paging depends on whether process-context identifiers (PCIDs) have been enabled by setting CR4.PCIDE:

- Table 4-12 illustrates how CR3 is used with 4-level paging and 5-level paging if $CR4.PCIDE = 0$.

Table 4-12. Use of CR3 with 4-Level Paging and 5-level Paging and $CR4.PCIDE = 0$

Bit Position(s)	Contents
2:0	Ignored

1. If $MAXPHYADDR < 52$, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by 4-level paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.
2. HLAT paging is used only with 4-level paging and 5-level paging. It is never used with 32-bit paging or PAE paging, regardless of the value of the “enable HLAT” VM-execution control.
3. This behavior applies if the CPU enumerates a maximum HLAT prefix size of 1 in $IA32_VMX_EPT_VPID_CAP[53:48]$ (see Appendix A.10). Behavior when a different value is enumerated is not currently defined.

Table 4-12. Use of CR3 with 4-Level Paging and 5-level Paging and CR4.PCIDE = 0 (Contd.)

Bit Position(s)	Contents
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table or PML5 table during linear-address translation (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table or PML5 table during linear-address translation (see Section 4.9.2)
11:5	Ignored
M-1:12	Physical address of the 4-KByte aligned PML4 table or PML5 table used for linear-address translation ¹
63:M	Reserved (must be 0)

NOTES:

1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

- Table 4-13 illustrates how CR3 is used with 4-level paging and 5-level paging if CR4.PCIDE = 1.

Table 4-13. Use of CR3 with 4-Level Paging and 5-Level Paging and CR4.PCIDE = 1

Bit Position(s)	Contents
11:0	PCID (see Section 4.10.1) ¹
M-1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation ²
63:M	Reserved (must be 0) ³

NOTES:

1. Section 4.9.2 explains how the processor determines the memory type used to access the PML4 table during linear-address translation with CR4.PCIDE = 1.

2. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

3. See Section 4.10.4.1 for use of bit 63 of the source operand of the MOV to CR3 instruction.

After software modifies the value of CR4.PCIDE, the logical processor immediately begins using CR3 as specified for the new value. For example, if software changes CR4.PCIDE from 1 to 0, the current PCID immediately changes from CR3[11:0] to 000H (see also Section 4.10.4.1). In addition, the logical processor subsequently determines the memory type used to access the PML4 table using CR3.PWT and CR3.PCD, which had been bits 4:3 of the PCID.

4.5.3 Use of HLATP with HLAT 4-Level Paging and 5-Level Paging

With HLAT paging, 4-level paging and 5-level paging each translate linear addresses using a hierarchy of in-memory paging structures located using the value of HLATP (a VM-execution control field in the VMCS), which is used to locate the first paging structure. For 4-level paging, this is the PML4 table, and for 5-level paging it is the PML5 table.

HLATP has the same format as that given for CR3 in Table 4-12, with the exception that bits 2:0 and bits 11:5 are reserved and must be zero (these bits are checked by VM entry). HLATP does not contain a PCID value. HLAT paging with CR4.PCIDE = 1 uses the PCID value in CR3[11:0].

4.5.4 Linear-Address Translation with 4-Level Paging and 5-Level Paging

4-level paging and 5-level paging may map linear addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages.¹ Figure 4-8 illustrates the translation process for 4-level paging when it produces a 4-KByte page; Figure 4-9 covers the case of a 2-MByte page, and Figure 4-10 the case of a 1-GByte page. (The process for 5-level paging is similar.)

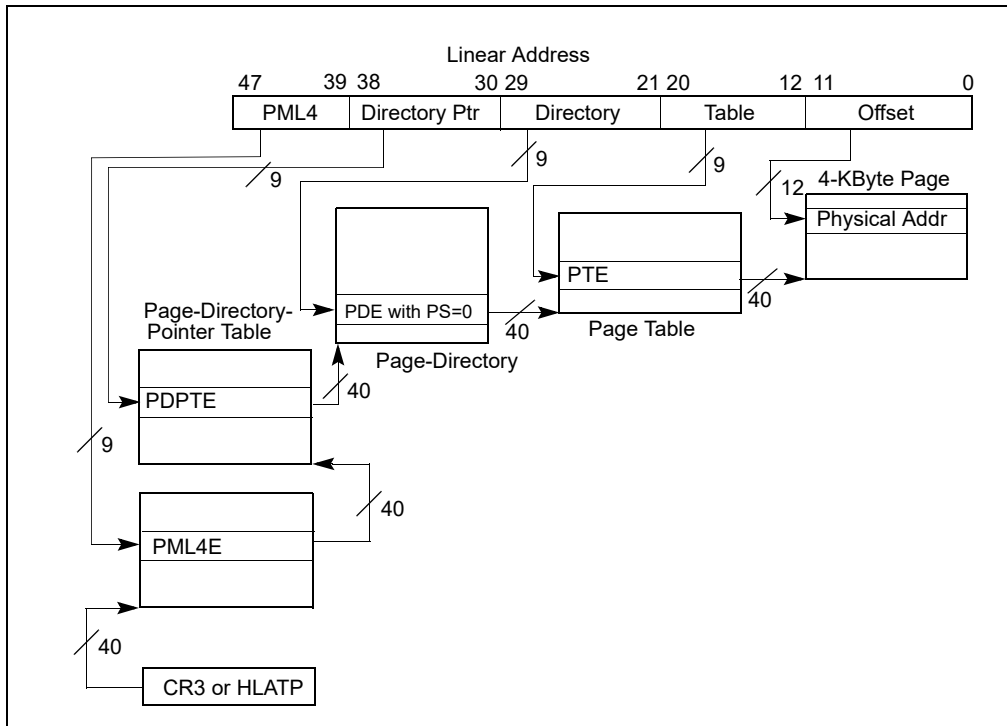


Figure 4-8. Linear-Address Translation to a 4-KByte Page using 4-Level Paging

1. Not all processors support 1-GByte pages; see Section 4.1.4.

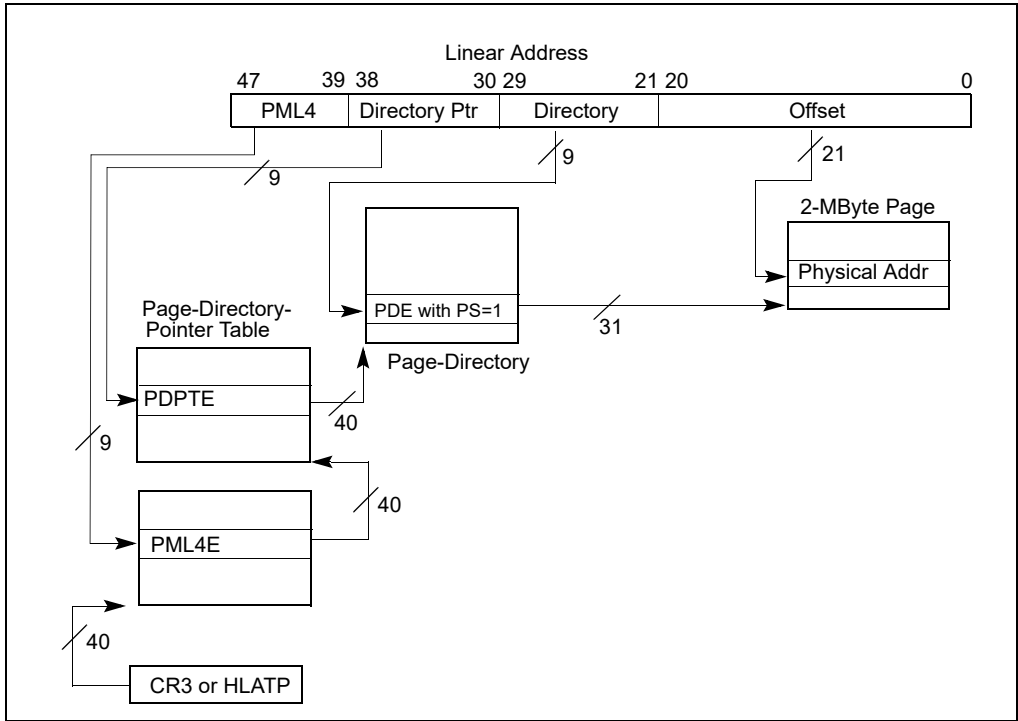


Figure 4-9. Linear-Address Translation to a 2-MByte Page using 4-Level Paging

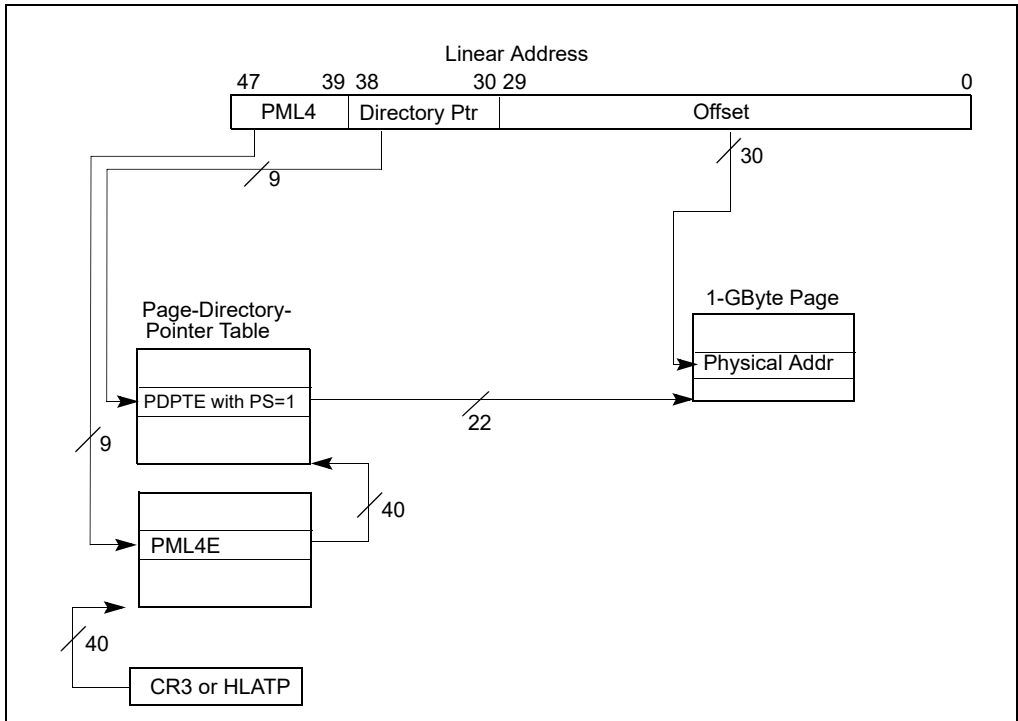


Figure 4-10. Linear-Address Translation to a 1-GByte Page using 4-Level Paging

4-level paging and 5-level paging associate with each linear address a **protection key**. Section 4.6 explains how the processor uses the protection key in its determination of the access rights of each linear address.

The remainder of this section describes the translation process used by 4-level paging and 5-level paging in more detail, as well as how the page size and protection key are determined. Because the process used by the two paging modes is similar, they are described together, with any differences identified, in the following items:

- With 5-level paging, a 4-KByte naturally aligned PML5 table is located at the physical address specified in bits 51:12 of CR3 (see Table 4-12). (4-level paging does not use a PML5 table and omits this step.) A PML5 table comprises 512 64-bit entries (PML5Es). A PML5E is selected using the physical address defined as follows:
 - Bits 51:12 are from CR3 or HLATP.
 - Bits 11:3 are bits 56:48 of the linear address.
 - Bits 2:0 are all 0.

Because a PML5E is identified using bits 56:48 of the linear address, it controls access to a 256-TByte region of the linear-address space.

With HLAT paging, if bit 11 of the PML5E is 1, translation is restarted with ordinary paging with a maximum page size of 256-TBytes (see Section 4.5.5). Otherwise, the translation process continues as described in the next item.

- A 4-KByte naturally aligned PML4 table is located at the physical address specified in bits 51:12 of CR3 (for 4-level paging; see Table 4-12) or in bits 51:12 of the PML4E (for 5-level paging; see Table 4-14). A PML4 table comprises 512 64-bit entries (PML4Es). A PML4E is selected using the physical address defined as follows:
 - Bits 51:12 are from CR3 or the HLATP (for 4-level paging) or in bits 51:12 of the PML4E (for 5-level paging).
 - Bits 11:3 are bits 47:39 of the linear address.
 - Bits 2:0 are all 0.

Because a PML4E is identified using bits 47:39 of the linear address, it controls access to a 512-GByte region of the linear-address space.

With HLAT paging, if bit 11 of the PML4E is 1, translation is restarted with ordinary paging with a maximum page size of 512-GBytes (see Section 4.5.5). Otherwise, the translation process continues as described in the next item.

- A 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in bits 51:12 of the PML4E (see Table 4-15). A page-directory-pointer table comprises 512 64-bit entries (PDPTEs). A PDPTE is selected using the physical address defined as follows:
 - Bits 51:12 are from the PML4E.
 - Bits 11:3 are bits 38:30 of the linear address.
 - Bits 2:0 are all 0.

Because a PDPTE is identified using bits 47:30 of the linear address, it controls access to a 1-GByte region of the linear-address space.

With HLAT paging, if bit 11 of the PDPTE is 1, translation is restarted with ordinary paging with a maximum page size of 1-GByte (see Section 4.5.5). Otherwise, the translation process continues as described below.

Use of the PDPTE depends on its PS flag (bit 7):¹

- If the PDPTE's PS flag is 1, the PDPTE maps a 1-GByte page (see Table 4-16). The final physical address is computed as follows:
 - Bits 51:30 are from the PDPTE.
 - Bits 29:0 are from the original linear address.

The linear address's protection key is the value of bits 62:59 of the PDPTE (see Section 4.6.2).

1. The PS flag of a PDPTE is reserved and must be 0 (if the P flag is 1) if 1-GByte pages are not supported. See Section 4.1.4 for how to determine whether 1-GByte pages are supported.

- If the PDPTE's PS flag is 0, a 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of the PDPTE (see Table 4-17). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
 - Bits 51:12 are from the PDPTE.
 - Bits 11:3 are bits 29:21 of the linear address.
 - Bits 2:0 are all 0.

Because a PDE is identified using bits 47:21 of the linear address, it controls access to a 2-MByte region of the linear-address space.

With HLAT paging, if bit 11 of the PDE is 1, translation is restarted with ordinary paging with a maximum page size of 2-MBytes (see Section 4.5.5). Otherwise, the translation process continues as described below.

Use of the PDE depends on its PS flag:

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table 4-18). The final physical address is computed as follows:
 - Bits 51:21 are from the PDE.
 - Bits 20:0 are from the original linear address.

The linear address's protection key is the value of bits 62:59 of the PDE (see Section 4.6.2).

- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table 4-19). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:

- Bits 51:12 are from the PDE.
- Bits 11:3 are bits 20:12 of the linear address.
- Bits 2:0 are all 0.

- Because a PTE is identified using bits 47:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-20).

With HLAT paging, if bit 11 of the PTE is 1, translation is restarted with ordinary paging with a maximum page size of 4-KBytes (see Section 4.5.5). Otherwise, the final physical address is computed as follows:

- Bits 51:12 are from the PTE.
- Bits 11:0 are from the original linear address.

The linear address's protection key is the value of bits 62:59 of the PTE (see Section 4.6.2).

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 4.7).

The following bits in a paging-structure entry are reserved with 4-level paging and 5-level paging (assuming that the entry's P flag is 1):

- Bits 51:MAXPHYADDR are reserved in every paging-structure entry.
- The PS flag is reserved in a PML5E or a PML4E.
- If 1-GByte pages are not supported, the PS flag is reserved in a PDPTE.¹
- If the PS flag in a PDPTE is 1, bits 29:13 of the entry are reserved.
- If the PS flag in a PDE is 1, bits 20:13 of the entry are reserved.
- If IA32_EFER.NXE = 0, the XD flag (bit 63) is reserved in every paging-structure entry.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

1. See Section 4.1.4 for how to determine whether 1-GByte pages are supported.

Figure 4-11 gives a summary of the formats of CR3 and the 4-level and 5-level paging-structure entries. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are “not present”; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

Table 4-14. Format of a PML5 Entry (PML5E) that References a PML4 Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a PML4 table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 256-TByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 256-TByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Reserved (must be 0)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M-1:12	Physical address of 4-KByte aligned PML4 table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 256-TByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-15. Format of a PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 512-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored

Table 4-15. Format of a PML4 Entry (PML4E) that References a Page-Directory-Pointer Table (Contd.)

Bit Position(s)	Contents
7 (PS)	Reserved (must be 0)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M-1:12	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 512-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-16. Format of a Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 1-GByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 1-GByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 1-GByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page directory; see Table 4-17)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
12 (PAT)	Indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2) ¹
29:13	Reserved (must be 0)
(M-1):30	Physical address of the 1-GByte page referenced by this entry

Table 4-16. Format of a Page-Directory-Pointer-Table Entry (PDPTTE) that Maps a 1-GBYTE Page (Contd.)

Bit Position(s)	Contents
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 4.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GBYTE page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:

1. The PAT is supported on all processors that support 4-level paging.

Table 4-17. Format of a Page-Directory-Pointer-Table Entry (PDPTTE) that References a Page Directory

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GBYTE region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GBYTE region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 1-GBYTE page; see Table 4-16)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
(M-1):12	Physical address of 4-KBYTE aligned page directory referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GBYTE region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-18. Format of a Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-19)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
12 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
20:13	Reserved (must be 0)
(M-1):21	Physical address of the 2-MByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 4.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-19. Format of a Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 4-18)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
(M-1):12	Physical address of 4-KByte aligned page table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-20. Format of a Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
(M-1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 4.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

4.5.5 Restart of HLAT Paging

As noted in Section 4.5.1, HLAT paging may specify that a translation of a linear address must be restarted. Specifically, this occurs when HLAT paging encounters a paging-structure entry that sets bit 11 (see Section 4.5.4).

When this occurs, translation of the linear address is restarted using ordinary paging (starting with a paging structure identified using CR3). The restarted translation proceeds just as if the HLAT feature were not enabled. The entire linear address is translated again, including those portions that had been used by HLAT paging prior to the restart.

The process of restarting HLAT paging (using ordinary paging) always specifies a maximum page size to be used when a resulting translation is cached in the TLBs. This maximum page size depends on the level of the paging-structure entry that restarts the translation by setting bit 11; details are given in Section 4.5.4. The page size of the translation produced by the restarted process is never greater than this maximum page size. See Section 4.10.2.2 for more discussion.

4.6 ACCESS RIGHTS

There is a translation for a linear address if the processes described in Section 4.3, Section 4.4.2, and Section 4.5 (depending upon the paging mode) completes and produces a physical address. Whether an access is permitted by a translation is determined by the access rights specified by the paging-structure entries controlling the translation;¹ paging-mode modifiers in CR0, CR4, and the IA32_EFER MSR; EFLAGS.AC; and the mode of the access.

Section 4.6.1 describes how the processor determines the access rights for each linear address. Section 4.6.2 provides additional information about how protection keys contribute to access-rights determination. (They do so only with 4-level paging and 5-level paging, and only if CR4.PKE = 1 or CR4.PKS = 1.)

NOTE

If HLAT paging is restarted, permissions are determined only by the access rights specified by the paging-structure entries that the subsequent ordinary paging used to translate the linear address. The access rights specified by the entries used earlier by HLAT paging do not apply.

4.6.1 Determination of Access Rights

Every access to a linear address is either a **supervisor-mode access** or a **user-mode access**. For all instruction fetches and most data accesses, this distinction is determined by the current privilege level (CPL): accesses made while $CPL < 3$ are supervisor-mode accesses, while accesses made while $CPL = 3$ are user-mode accesses.

Some operations implicitly access system data structures with linear addresses; the resulting accesses to those data structures are supervisor-mode accesses regardless of CPL. Examples of such accesses include the following: accesses to the global descriptor table (GDT) or local descriptor table (LDT) to load a segment descriptor; accesses to the interrupt descriptor table (IDT) when delivering an interrupt or exception; and accesses to the task-state segment (TSS) as part of a task switch or change of CPL. All these accesses are called **implicit supervisor-mode accesses** regardless of CPL. Other accesses made while $CPL < 3$ are called **explicit supervisor-mode accesses**.

Access rights are also controlled by the **mode** of a linear address as specified by the paging-structure entries controlling the translation of the linear address. If the U/S flag (bit 2) is 0 in at least one of the paging-structure entries, the address is a **supervisor-mode address**. Otherwise, the address is a **user-mode address**.

When the shadow-stack feature of control-flow enforcement technology (CET) is enabled, certain accesses to linear addresses are considered **shadow-stack accesses** (see Section 18.2, "Shadow Stacks" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*). Like ordinary data accesses, each shadow-stack access is defined as being either a user access or a supervisor access. In general, a shadow-stack access is a user access if $CPL = 3$ and a supervisor access if $CPL < 3$. The WRUSS instruction is an exception; although it can be executed only if $CPL = 0$, the processor treats its shadow-stack accesses as user accesses.

1. With PAE paging, the PDPTes do not determine access rights.

Shadow-stack accesses are allowed only to **shadow-stack addresses**. A linear address is a shadow-stack address if the following are true of the translation of the linear address: (1) the R/W flag (bit 1) is 0 and the dirty flag (bit 6) is 1 in the paging-structure entry that maps the page containing the linear address; and (2) the R/W flag is 1 in every other paging-structure entry controlling the translation of the linear address.

The following items detail how paging determines access rights (only the items noted explicitly apply to shadow-stack accesses):

NOTE

Many of the items below refer to an address with a protection key for which read (or write) access is permitted. Section 4.6.2 provides details on when a protection key will permit (or not permit) a data access (read or write) to a linear address using that protection key.

- For supervisor-mode accesses:
 - Data may be read (implicitly or explicitly) from any supervisor-mode address with a protection key for which read access is permitted.
 - Data reads from user-mode pages.
Access rights depend on the value of CR4.SMAP:
 - If CR4.SMAP = 0, data may be read from any user-mode address with a protection key for which read access is permitted.
 - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
 - If EFLAGS.AC = 1 and the access is explicit, data may be read from any user-mode address with a protection key for which read access is permitted.
 - If EFLAGS.AC = 0 or the access is implicit, data may not be read from any user-mode address.
 - Data writes to supervisor-mode addresses.
Access rights depend on the value of CR0.WP:
 - If CR0.WP = 0, data may be written to any supervisor-mode address with a protection key for which write access is permitted.
 - If CR0.WP = 1, data may be written to any supervisor-mode address with a translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any supervisor-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
 - Data writes to user-mode addresses.
Access rights depend on the value of CR0.WP:
 - If CR0.WP = 0, access rights depend on the value of CR4.SMAP:
 - If CR4.SMAP = 0, data may be written to any user-mode address with a protection key for which write access is permitted.
 - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
 - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a protection key for which write access is permitted.
 - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
 - If CR0.WP = 1, access rights depend on the value of CR4.SMAP:
 - If CR4.SMAP = 0, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
 - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:

- If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
 - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
- Instruction fetches from supervisor-mode addresses.
 - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any supervisor-mode address.
 - For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any supervisor-mode address with a translation for which the XD flag (bit 63) is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any supervisor-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.
 - Instruction fetches from user-mode addresses.

Access rights depend on the values of CR4.SMEP:

 - If CR4.SMEP = 0, access rights depend on the paging mode and the value of IA32_EFER.NXE:
 - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
 - For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any user-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.
 - If CR4.SMEP = 1, instructions may not be fetched from any user-mode address.
 - Supervisor-mode shadow-stack accesses are allowed only to supervisor-mode shadow-stack addresses (see above).
 - For user-mode accesses:
 - Data reads.

Access rights depend on the mode of the linear address:

 - Data may be read from any user-mode address with a protection key for which read access is permitted.
 - Data may not be read from any supervisor-mode address.
 - Data writes.

Access rights depend on the mode of the linear address:

 - Data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted.
 - Data may not be written to any supervisor-mode address.
 - Instruction fetches.

Access rights depend on the mode of the linear address, the paging mode, and the value of IA32_EFER.NXE:

 - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
 - For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation.
 - Instructions may not be fetched from any supervisor-mode address.
 - User-mode shadow-stack accesses made outside enclave mode are allowed only to user-mode shadow-stack addresses (see above). User-mode shadow-stack accesses made in enclave mode are treated like ordinary data accesses (see above).

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about access rights. The processor may enforce access rights based on the TLBs and paging-structure caches instead of on the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change access rights, the processor might not use that change for a subsequent access to an affected linear address (see Section 4.10.4.3). See Section 4.10.4.2 for how software can ensure that the processor uses the modified access rights.

4.6.2 Protection Keys

4-level paging and 5-level paging associate a 4-bit protection key with each linear address (the protection key located in bits 62:59 of the paging-structure entry that mapped the page containing the linear address; see Section 4.5). Two protection key features control accesses to linear addresses based on their protection keys:

- If CR4.PKE = 1, the PKRU register determines, for each protection key, whether user-mode addresses with that protection key may be read or written.
- If CR4.PKS = 1, the IA32_PKRS MSR (MSR index 6E1H) determines, for each protection key, whether supervisor-mode addresses with that protection key may be read or written.

32-bit paging and PAE paging do not associate linear addresses with protection keys. For the purposes of Section 4.6.1, reads and writes are implicitly permitted for all protection keys with either of those paging modes.

The PKRU register (protection-key rights for user pages) is a 32-bit register with the following format: for each i ($0 \leq i \leq 15$), PKRU[2*i*] is the **access-disable bit** for protection key i (AD*i*); PKRU[2*i*+1] is the **write-disable bit** for protection key i (WD*i*). The IA32_PKRS MSR has the same format (bits 63:32 of the MSR are reserved and must be zero).

Software can use the RDPKRU and WRPKRU instructions with ECX = 0 to read and write PKRU. In addition, the PKRU register is XSAVE-managed state and can thus be read and written by instructions in the XSAVE feature set. See Chapter 13, “Managing State Using the XSAVE Feature Set,” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* for more information about the XSAVE feature set.

Software can use the RDMSR and WRMSR instructions to read and write the IA32_PKRS MSR. Writes to the IA32_PKRS MSR using WRMSR are not serializing. The IA32_PKRS MSR is not XSAVE-managed.

How a linear address’s protection key controls access to the address depends on the mode of the linear address:

- A linear address’s protection key controls only data accesses to the address. It does not in any way affect instructions fetches from the address.
- If CR4.PKE = 0, the protection key of a user-mode address does not control data accesses to the address (for the purposes of Section 4.6.1, reads and writes of user-mode addresses are implicitly permitted for all protection keys).

If CR4.PKE = 1, use of the protection key i of a user-mode address depends on the value of the PKRU register:

- If AD*i* = 1, no data accesses are permitted.
- If WD*i* = 1, permission may be denied to certain data write accesses:
 - User-mode write accesses are not permitted.
 - Supervisor-mode write accesses are not permitted if CR0.WP = 1. (If CR0.WP = 0, WD*i* does not affect supervisor-mode write accesses to user-mode addresses with protection key i .)

- If CR4.PKS = 0, the protection key of a supervisor-mode address does not control data accesses to the address (for the purposes of Section 4.6.1, reads and writes of supervisor-mode addresses are implicitly permitted for all protection keys).

If CR4.PKS = 1, use of the protection key i of a supervisor-mode address depends on the value of the IA32_PKRS MSR:

- If AD*i* = 1, no data accesses are permitted.
- If WD*i* = 1, write accesses are not permitted if CR0.WP = 1. (If CR0.WP = 0, IA32_PKRS.WD*i* does not affect write accesses to supervisor-mode addresses with protection key i .)

Protection keys apply to shadow-stack accesses just as they do to ordinary data accesses.

4.7 PAGE-FAULT EXCEPTIONS

Accesses using linear addresses may cause **page-fault exceptions** (#PF; exception 14). An access to a linear address may cause a page-fault exception for either of two reasons: (1) there is no translation for the linear address; or (2) there is a translation for the linear address, but its access rights do not permit the access.

As noted in Section 4.3, Section 4.4.2, and Section 4.5, there is no translation for a linear address if the translation process for that address would use a paging-structure entry in which the P flag (bit 0) is 0 or one that sets a reserved bit.¹ If there is a translation for a linear address, its access rights are determined as specified in Section 4.6.

When Intel® Software Guard Extensions (Intel® SGX) are enabled, the processor may deliver exception 14 for reasons unrelated to paging. See Section 34.3, “Access-control Requirements” and Section 34.20, “Enclave Page Cache Map (EPCM)” in Chapter 34, “Enclave Access Control and Data Structures.” Such an exception is called an **SGX-induced page fault**. The processor uses the error code to distinguish SGX-induced page faults from ordinary page faults.

Figure 4-12 illustrates the error code that the processor provides on delivery of a page-fault exception. The following items explain how the bits in the error code describe the nature of the page-fault exception:

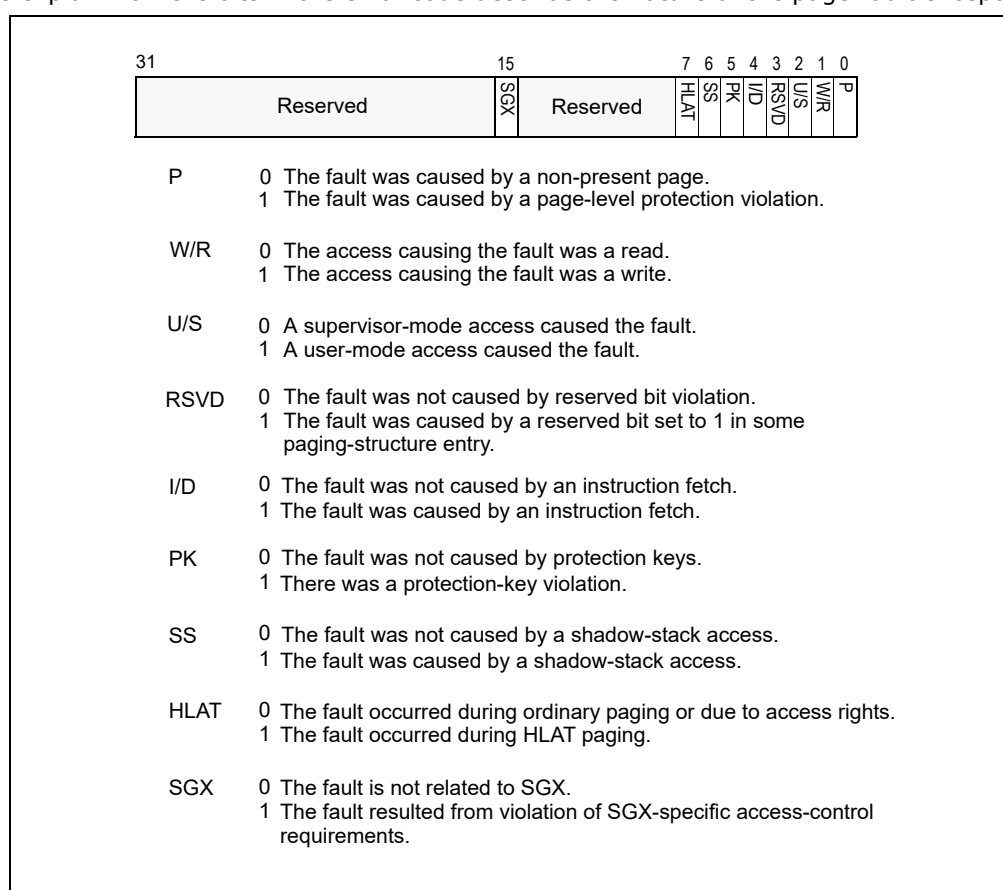


Figure 4-12. Page-Fault Error Code

- P flag (bit 0).
This flag is 0 if there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.

1. If HLAT paging encounters a paging-structure entry that sets a reserved bit, there is no translation even if the bit 11 of the entry indicates a restart. In this case, there is a page fault and the translation is not restarted.

PAGING

- W/R (bit 1).
If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- U/S (bit 2).
If a user-mode access caused the page-fault exception, this flag is 1; it is 0 if a supervisor-mode access did so. This flag describes the access causing the page-fault exception, not the access rights specified by paging. User-mode and supervisor-mode accesses are defined in Section 4.6.
- RSVD flag (bit 3).
This flag is 1 if there is no translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address. (Because reserved bits are not checked in a paging-structure entry whose P flag is 0, bit 3 of the error code can be set only if bit 0 is also set.¹)
Bits reserved in the paging-structure entries are reserved for future functionality. Software developers should be aware that such bits may be used in the future and that a paging-structure entry that causes a page-fault exception on one processor might not do so in the future.
- I/D flag (bit 4).
This flag is 1 if (1) the access causing the page-fault exception was an instruction fetch; and (2) either (a) CR4.SMEP = 1; or (b) both (i) CR4.PAE = 1 (either PAE paging, 4-level paging, or 5-level paging is in use); and (ii) IA32_EFER.NXE = 1. Otherwise, the flag is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- PK flag (bit 5).
This flag is 1 only for data accesses and only with 4-level paging and 5-level paging. In these cases, the setting depends on the mode of the address being accessed:
 - For accesses to supervisor-mode addresses, the flag is set if (1) CR4.PKS = 1; (2) the linear address has protection key i ; and (3) the IA32_PKRS MSR (see Section 4.6.2) is such that either (a) $AD_i = 1$; or (b) the following all hold: (i) $WD_i = 1$; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode access. (Note that this flag may be set on page faults due to user-mode accesses to supervisor-mode addresses.)
 - For accesses to user-mode addresses, the flag is set if (1) CR4.PKE = 1; (2) the linear address has protection key i ; and (3) the PKRU register (see Section 4.6.2) is such that either (a) $AD_i = 1$; or (b) the following all hold: (i) $WD_i = 1$; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode access.
- SS (bit 6).
If the access causing the page-fault exception was a shadow-stack access (including shadow-stack accesses in enclave mode), this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- HLAT (bit 7).
This flag is 1 if there is no translation for the linear address using HLAT paging because, in one of the paging-structure entries used to translate that address, either the P flag was 0 or a reserved bit was set. An error code will set this flag only if it clears bit 0 or sets bit 3. This flag will not be set by a page fault resulting from a violation of access rights, nor for one encountered during ordinary paging, including the case in which there has been a restart of HLAT paging.
- SGX flag (bit 15).
This flag is 1 if the exception is unrelated to paging and resulted from violation of SGX-specific access-control requirements. Because such a violation can occur only if there is no ordinary page fault, this flag is set only if the P flag (bit 0) is 1 and the RSVD flag (bit 3) and the PK flag (bit 5) are both 0.

Page-fault exceptions occur only due to an attempt to use a linear address. Failures to load the PDPTE registers with PAE paging (see Section 4.4.1) cause general-protection exceptions ($\#GP(0)$) and not page-fault exceptions.

1. Some past processors had errata for some page faults that occur when there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address. Due to these errata, some such page faults produced error codes that cleared bit 0 (P flag) and set bit 3 (RSVD flag).

4.8 ACCESSED AND DIRTY FLAGS

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag.¹ For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).

The previous two paragraphs apply also to HLAT paging. If HLAT paging encounters a paging-structure entry that sets bit 11, indicating a restart, the processor will set the accessed flag in that entry; it will not set the dirty flag because, if an entry indicates a restart, it does not identify the final physical address for the linear address being translated.

NOTE

If software on one logical processor writes to a page while software on another logical processor concurrently clears the R/W flag in the paging-structure entry that maps the page, execution on some processors may result in the entry's dirty flag being set (due to the write on the first logical processor) and the entry's R/W flag being clear (due to the update to the entry on the second logical processor). This will never occur on a processor that supports control-flow enforcement technology (CET). Specifically, a processor that supports CET will never set the dirty flag in a paging-structure entry in which the R/W flag is clear.

Memory-management software may clear these flags when a page or a paging structure is initially loaded into physical memory. These flags are "sticky," meaning that, once set, the processor does not clear them; only software can clear them.

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the processor might not set the corresponding bit in memory on a subsequent access using an affected linear address (see Section 4.10.4.3). See Section 4.10.4.2 for how software can ensure that these bits are updated as desired.

NOTE

The accesses used by the processor to set these flags may or may not be exposed to the processor's self-modifying code detection logic. If the processor is executing code from the same memory area that is being used for the paging structures, the setting of these flags may or may not result in an immediate change to the executing code stream.

4.9 PAGING AND MEMORY TYPING

The **memory type** of a memory access refers to the type of caching used for that access. Chapter 11, "Memory Cache Control" provides many details regarding memory typing in the Intel-64 and IA-32 architectures. This section describes how paging contributes to the determination of memory typing.

The way in which paging contributes to memory typing depends on whether the processor supports the **Page Attribute Table (PAT)**; see Section 11.12).² Section 4.9.1 and Section 4.9.2 explain how paging contributes to memory typing depending on whether the PAT is supported.

1. With PAE paging, the PDPTes are not used during linear-address translation but only to load the PDPTe registers for some executions of the MOV CR instruction (see Section 4.4.1). For this reason, the PDPTes do not contain accessed flags with PAE paging.
2. The PAT is supported on Pentium III and more recent processor families. See Section 4.1.4 for how to determine whether the PAT is supported.

4.9.1 Paging and Memory Typing When the PAT is Not Supported (Pentium Pro and Pentium II Processors)

NOTE

The PAT is supported on all processors that support 4-level paging or 5-level paging. Thus, this section applies only to 32-bit paging and PAE paging.

If the PAT is not supported, paging contributes to memory typing in conjunction with the memory-type range registers (MTRRs) as specified in Table 11-6 in Section 11.5.2.1.

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a PCD value and a PWT value. The latter two values are determined as follows:

- For an access to a PDE with 32-bit paging, the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, the PCD and PWT values come from the relevant PDPTTE register.
- For an access to a PTE, the PCD and PWT values come from the relevant PDE.
- For an access to the physical address that is the translation of a linear address, the PCD and PWT values come from the relevant PTE (if the translation uses a 4-KByte page) or the relevant PDE (otherwise).
- With PAE paging, the UC memory type is used when loading the PDPTTEs (see Section 4.4.1).

4.9.2 Paging and Memory Typing When the PAT is Supported (Pentium III and More Recent Processor Families)

If the PAT is supported, paging contributes to memory typing in conjunction with the PAT and the memory-type range registers (MTRRs) as specified in Table 11-7 in Section 11.5.2.2.

The PAT is a 64-bit MSR (IA32_PAT; MSR index 277H) comprising eight (8) 8-bit entries (entry i comprises bits $8i+7:8i$ of the MSR).

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a memory type selected from the PAT. Table 11-11 in Section 11.12.3 specifies how a memory type is selected from the PAT. Specifically, it comes from entry i of the PAT, where i is defined as follows:

- For an access to an entry in a paging structure whose address is in CR3 (e.g., the PML4 table with 4-level paging):
 - For 4-level paging or 5-level paging with $CR4.PCIDE = 1$, $i = 0$.
 - Otherwise, $i = 2*PCD+PWT$, where the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, $i = 2*PCD+PWT$, where the PCD and PWT values come from the relevant PDPTTE register.
- For an access to a paging-structure entry X whose address is in another paging-structure entry Y, $i = 2*PCD+PWT$, where the PCD and PWT values come from Y.
- For an access to the physical address that is the translation of a linear address, $i = 4*PAT+2*PCD+PWT$, where the PAT, PCD, and PWT values come from the relevant PTE (if the translation uses a 4-KByte page), the relevant PDE (if the translation uses a 2-MByte page or a 4-MByte page), or the relevant PDPTTE (if the translation uses a 1-GByte page).
- With PAE paging, the WB memory type is used when loading the PDPTTEs (see Section 4.4.1).¹

1. Some older IA-32 processors used the UC memory type when loading the PDPTTEs. Some processors may use the UC memory type if $CRO.CD = 1$ or if the MTRRs are disabled. These behaviors are model-specific and not architectural.

4.9.3 Caching Paging-Related Information about Memory Typing

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about memory typing. The processor may use memory-typing information from the TLBs and paging-structure caches instead of from the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change the memory-typing bits, the processor might not use that change for a subsequent translation using that entry or for access to an affected linear address. See Section 4.10.4.2 for how software can ensure that the processor uses the modified memory typing.

4.10 CACHING TRANSLATION INFORMATION

The Intel-64 and IA-32 architectures may accelerate the address-translation process by caching data from the paging structures on the processor. Because the processor does not ensure that the data that it caches are always consistent with the structures in memory, it is important for software developers to understand how and when the processor may cache such data. They should also understand what actions software can take to remove cached data that may be inconsistent and when it should do so. This section provides software developers information about the relevant processor operation.

Section 4.10.1 introduces process-context identifiers (PCIDs), which a logical processor may use to distinguish information cached for different linear-address spaces. Section 4.10.2 and Section 4.10.3 describe how the processor may cache information in translation lookaside buffers (TLBs) and paging-structure caches, respectively. Section 4.10.4 explains how software can remove inconsistent cached information by invalidating portions of the TLBs and paging-structure caches. Section 4.10.5 describes special considerations for multiprocessor systems.

4.10.1 Process-Context Identifiers (PCIDs)

Process-context identifiers (**PCIDs**) are a facility by which a logical processor may cache information for multiple linear-address spaces. The processor may retain cached information when software switches to a different linear-address space with a different PCID (e.g., by loading CR3; see Section 4.10.4.1 for details).

A PCID is a 12-bit identifier. Non-zero PCIDs are enabled by setting the PCIDE flag (bit 17) of CR4. If CR4.PCIDE = 0, the current PCID is always 000H; otherwise, the current PCID is the value of bits 11:0 of CR3.¹ Not all processors allow CR4.PCIDE to be set to 1; see Section 4.1.4 for how to determine whether this is allowed.

The processor ensures that CR4.PCIDE can be 1 only in IA-32e mode (thus, 32-bit paging and PAE paging use only PCID 000H). In addition, software can change CR4.PCIDE from 0 to 1 only if CR3[11:0] = 000H. These requirements are enforced by the following limitations on the MOV CR instruction:

- MOV to CR4 causes a general-protection exception (#GP) if it would change CR4.PCIDE from 0 to 1 and either IA32_EFER.LMA = 0 or CR3[11:0] ≠ 000H.
- MOV to CR0 causes a general-protection exception if it would clear CR0.PG to 0 while CR4.PCIDE = 1.

When a logical processor creates entries in the TLBs (Section 4.10.2) and paging-structure caches (Section 4.10.3), it associates those entries with the current PCID. When using entries in the TLBs and paging-structure caches to translate a linear address, a logical processor uses only those entries associated with the current PCID (see Section 4.10.2.4 for an exception).

If CR4.PCIDE = 0, a logical processor does not cache information for any PCID other than 000H. This is because (1) if CR4.PCIDE = 0, the logical processor will associate any newly cached information with the current PCID, 000H; and (2) if MOV to CR4 clears CR4.PCIDE, all cached information is invalidated (see Section 4.10.4.1).

1. Note that, while HLAT paging (Section 4.5.3) does not use CR3 to locate the first paging structure, it does use the PCID in CR3[11:0] when CR4.PCIDE = 1.

NOTE

In revisions of this manual that were produced when no processors allowed CR4.PCIDE to be set to 1, the Section “Caching Translation Information” discussed the caching of translation information without any reference to PCIDs. While the section now refers to PCIDs in its specification of this caching, this documentation change is not intended to imply any change to the behavior of processors that do not allow CR4.PCIDE to be set to 1.

4.10.2 Translation Lookaside Buffers (TLBs)

A processor may cache information about the translation of linear addresses in translation lookaside buffers (TLBs). In general, TLBs contain entries that map page numbers to page frames; these terms are defined in Section 4.10.2.1. Section 4.10.2.2 describes how information may be cached in TLBs, and Section 4.10.2.3 gives details of TLB usage. Section 4.10.2.4 explains the global-page feature, which allows software to indicate that certain translations should receive special treatment when cached in the TLBs.

4.10.2.1 Page Numbers, Page Frames, and Page Offsets

Section 4.3, Section 4.4.2, and Section 4.5 give details of how the different paging modes translate linear addresses to physical addresses. Specifically, the upper bits of a linear address (called the **page number**) determine the upper bits of the physical address (called the **page frame**); the lower bits of the linear address (called the **page offset**) determine the lower bits of the physical address. The boundary between the page number and the page offset is determined by the **page size**. Specifically:

- 32-bit paging:
 - If the translation does not use a PTE (because CR4.PSE = 1 and the PS flag is 1 in the PDE used), the page size is 4 MBytes and the page number comprises bits 31:22 of the linear address.
 - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.
- PAE paging:
 - If the translation does not use a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 31:21 of the linear address.
 - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.
- 4-level paging and 5-level paging:
 - If the translation does not use a PDE (because the PS flag is 1 in the PDPTE used), the page size is 1 GByte and the page number comprises bits 47:30 of the linear address.
 - If the translation does use a PDE but does not use a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 47:21 of the linear address.
 - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 47:12 of the linear address.
 - The page size identified by the preceding items may be reduced if there has been a restart of HLAT paging (see Section 4.5.5). Restart of HLAT paging always specifies a maximum page size; this page size is determined by the level of the paging-structure entry that caused the restart. The page size used by the translation is the minimum of the maximum page size specified by the restart and the page size determined by the restarted translation (as specified by the previous items).

For example, suppose that HLAT paging encounters a PDE that sets bit 11, indicating a restart. As a result, the restart uses a maximum page size of 2 MBytes. Suppose that the restarted translation encounters a PDPTE that sets bit 7, indicating a 1-GByte page. In this case, the translation produced will have a page size of 2 MBytes (the smaller of the two sizes).

4.10.2.2 Caching Translations in TLBs

The processor may accelerate the paging process by caching individual translations in **translation lookaside buffers (TLBs)**. Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number (see Section 4.6):
 - The logical-AND of the R/W flags.
 - The logical-AND of the U/S flags.
 - The logical-OR of the XD flags (necessary only if IA32_EFER.NXE = 1).
 - The protection key (only with 4-level paging and 5-level paging).
- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry in which the PS flag is 1):
 - The dirty flag (see Section 4.8).
 - The memory type (see Section 4.9).

(TLB entries may contain other information as well. A processor may implement multiple TLBs, and some of these may be for special purposes, e.g., only for instruction fetches. Such special-purpose TLBs may not contain some of this information if it is not necessary. For example, a TLB used only for instruction fetches need not contain information about the R/W and dirty flags.)

As noted in Section 4.10.1, any TLB entries created by a logical processor are associated with the current PCID.

Processors need not implement any TLBs. Processors that do implement TLBs may invalidate any TLB entry at any time. Software should not rely on the existence of TLBs or on the retention of TLB entries.

4.10.2.3 Details of TLB Use

Because the TLBs cache entries only for linear addresses with translations, there can be a TLB entry for a page number only if the P flag is 1 and the reserved bits are 0 in each of the paging-structure entries used to translate that page number. In addition, the processor does not cache a translation for a page number unless the accessed flag is 1 in each of the paging-structure entries used during translation; before caching a translation, the processor sets any of these accessed flags that is not already 1.

Subject to the limitations given in the previous paragraph, the processor may cache a translation for any linear address, even if that address is not used to access memory. For example, the processor may cache translations required for prefetches and for accesses that result from speculative execution that would never actually occur in the executed code path.

If the page number of a linear address corresponds to a TLB entry associated with the current PCID, the processor may use that TLB entry to determine the page frame, access rights, and other attributes for accesses to that linear address. In this case, the processor may not actually consult the paging structures in memory. The processor may retain a TLB entry unmodified even if software subsequently modifies the relevant paging-structure entries in memory. See Section 4.10.4.2 for how software can ensure that the processor uses the modified paging-structure entries.

If the paging structures specify a translation using a page larger than 4 KBytes, some processors may cache multiple smaller-page TLB entries for that translation. Each such TLB entry would be associated with a page number corresponding to the smaller page size (e.g., bits 47:12 of a linear address with 4-level paging), even though part of that page number (e.g., bits 20:12) is part of the offset with respect to the page specified by the paging structures. The upper bits of the physical address in such a TLB entry are derived from the physical address in the PDE used to create the translation, while the lower bits come from the linear address of the access for which the translation is created. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. For example, an execution of INVLPG for a linear address on such a page invalidates any and all smaller-page TLB entries for the translation of any linear address on that page.

If software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes, the TLBs may subsequently contain multiple translations for the address range (one for each page size). A reference to a linear address in the address range may use any of these translations. Which translation is used may vary from one execution to another, and the choice may be implementation-specific.

4.10.2.4 Global Pages

The Intel-64 and IA-32 architectures also allow for **global pages** when the PGE flag (bit 7) is 1 in CR4. If the G flag (bit 8) is 1 in a paging-structure entry that maps a page (either a PTE or a paging-structure entry in which the PS flag is 1), any TLB entry cached for a linear address using that paging-structure entry is considered to be **global**. Because the G flag is used only in paging-structure entries that map a page, and because information from such entries is not cached in the paging-structure caches, the global-page feature does not affect the behavior of the paging-structure caches.

A logical processor may use a global TLB entry to translate a linear address, even if the TLB entry is associated with a PCID different from the current PCID.

4.10.3 Paging-Structure Caches

In addition to the TLBs, a processor may cache other information about the paging structures in memory.

4.10.3.1 Caches for Paging Structures

A processor may support any or all of the following paging-structure caches:

- **PML5E cache** (5-level paging only). Each PML5E-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 56:48 have that value. The entry contains information from the PML5E used to translate such linear addresses:
 - The physical address from the PML5E (the address of the PML4 table).
 - The value of the R/W flag of the PML5E.
 - The value of the U/S flag of the PML5E.
 - The value of the XD flag of the PML5E.
 - The values of the PCD and PWT flags of the PML5E.

The following items detail how a processor may use the PML5E cache:

- If the processor has a PML5E-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E in memory).
- The processor does not create a PML5E-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML5E in memory.
- The processor does not create a PML5E-cache entry unless the accessed flag is 1 in the PML5E in memory; before caching a translation, the processor sets the accessed flag if it is not already 1.
- The processor may create a PML5E-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced PML4 table).
- If the processor creates a PML5E-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E in memory.
- **PML4E cache** (4-level paging and 5-level paging only). The use of the PML4E cache depends on the paging mode:
 - For 4-level paging, each PML4E-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 47:39 have that value.
 - For 5-level paging, each PML4E-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 56:39 have that value.

A PML4E-cache entry contains information from the PML5E and PML4E used to translate the relevant linear addresses (for 4-level paging, the PML5E does not apply):

- The physical address from the PML4E (the address of the page-directory-pointer table).
- The logical-AND of the R/W flags in the PML5E and the PML4E.
- The logical-AND of the U/S flags in the PML5E and the PML4E.
- The logical-OR of the XD flags in the PML5E and the PML4E.
- The values of the PCD and PWT flags of the PML4E.

The following items detail how a processor may use the PML4E cache:

- If the processor has a PML4E-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E and PML4E in memory).
- The processor does not create a PML4E-cache entry unless the P flags are 1 and all reserved bits are 0 in the PML5E and the PML4E in memory.
- The processor does not create a PML4E-cache entry unless the accessed flags are 1 in the PML5E and the PML4E in memory; before caching a translation, the processor sets any accessed flags that are not already 1.
- The processor may create a PML4E-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-directory-pointer table).
- If the processor creates a PML4E-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E in memory.

- **PDPTe cache** (4-level paging and 5-level paging only).¹ The use of the PML4E cache depends on the paging mode:

- For 4-level paging, each PDPTe-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 47:30 have that value.
- For 5-level paging, each PDPTe-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 56:30 have that value.

A PDPTe-cache entry contains information from the PML5E, PML4E, PDPTe used to translate the relevant linear addresses (for 4-level paging, the PML5E does not apply):

- The physical address from the PDPTe (the address of the page directory). (No PDPTe-cache entry is created for a PDPTe that maps a 1-GByte page.)
- The logical-AND of the R/W flags in the PML5E, PML4E, and PDPTe.
- The logical-AND of the U/S flags in the PML5E, PML4E, and PDPTe.
- The logical-OR of the XD flags in the PML5E, PML4E, and PDPTe.
- The values of the PCD and PWT flags of the PDPTe.

The following items detail how a processor may use the PDPTe cache:

- If the processor has a PDPTe-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E, PML4E, and PDPTe in memory).
- The processor does not create a PDPTe-cache entry unless the P flags are 1, the PS flags are 0, and the reserved bits are 0 in the PML5E, PML4E, and PDPTe in memory.
- The processor does not create a PDPTe-cache entry unless the accessed flags are 1 in the PML5E, PML4E and PDPTe in memory; before caching a translation, the processor sets any accessed flags that are not already 1.
- The processor may create a PDPTe-cache entry even if there are no translations for any linear address that might use that entry.
- If the processor creates a PDPTe-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E, PML4E, or PDPTe in memory.

1. With PAE paging, the PDPTes are stored in internal, non-architectural registers. The operation of these registers is described in Section 4.4.1 and differs from that described here.

- **PDE cache.** The use of the PDE cache depends on the paging mode:
 - For 32-bit paging, each PDE-cache entry is referenced by a 10-bit value and is used for linear addresses for which bits 31:22 have that value.
 - For PAE paging, each PDE-cache entry is referenced by an 11-bit value and is used for linear addresses for which bits 31:21 have that value.
 - For 4-level paging, each PDE-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 47:21 have that value.
 - For 5-level paging, each PDE-cache entry is referenced by a 36-bit value and is used for linear addresses for which bits 56:21 have that value.

A PDE-cache entry contains information from the PML5E, PML4E, PDPTE, and PDE used to translate the relevant linear addresses (for 32-bit paging and PAE paging, only the PDE applies; for 4-level paging, the PML5E does not apply):

- The physical address from the PDE (the address of the page table). (No PDE-cache entry is created for a PDE that maps a page.)
- The logical-AND of the R/W flags in the PML5E, PML4E, PDPTE, and PDE.
- The logical-AND of the U/S flags in the PML5E, PML4E, PDPTE, and PDE.
- The logical-OR of the XD flags in the PML5E, PML4E, PDPTE, and PDE.
- The values of the PCD and PWT flags of the PDE.

The following items detail how a processor may use the PDE cache (references below to PML5Es, PML4Es, and PDPTEs apply only to 4-level paging and to 5-level paging, as appropriate):

- If the processor has a PDE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E, PML4E, PDPTE, and PDE in memory).
- The processor does not create a PDE-cache entry unless the P flags are 1, the PS flags are 0, and the reserved bits are 0 in the PML5E, PML4E, PDPTE, and PDE in memory.
- The processor does not create a PDE-cache entry unless the accessed flag is 1 in the PML5E, PML4E, PDPTE, and PDE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.
- The processor may create a PDE-cache entry even if there are no translations for any linear address that might use that entry.
- If the processor creates a PDE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E, PML4E, PDPTE, or PDE in memory.

Information from a paging-structure entry can be included in entries in the paging-structure caches for other paging-structure entries referenced by the original entry. For example, if the R/W flag is 0 in a PML4E, then the R/W flag will be 0 in any PDPTE-cache entry for a PDPTE from the page-directory-pointer table referenced by that PML4E. This is because the R/W flag of each such PDPTE-cache entry is the logical-AND of the R/W flags in the appropriate PML4E and PDPTE.

On processors that support HLAT paging (see Section 4.5.1), each entry in a paging-structure cache indicates whether the entry was cached during ordinary paging or HLAT paging. When the processor commences linear-address translation using ordinary paging (respectively, HLAT paging), it will use only entries that indicate that they were cached during ordinary paging (respectively, HLAT paging).

Entries that were cached during HLAT paging also include the restart flag (bit 11) of the original paging-structure entry. When the processor commences HLAT paging using such an entry, it immediately restarts (using ordinary paging) if this cached restart flag is 1.

The paging-structure caches contain information only from paging-structure entries that reference other paging structures (and not those that map pages). Because the G flag is not used in such paging-structure entries, the global-page feature does not affect the behavior of the paging-structure caches.

The processor may create entries in paging-structure caches for translations required for prefetches and for accesses that are a result of speculative execution that would never actually occur in the executed code path.

As noted in Section 4.10.1, any entries created in paging-structure caches by a logical processor are associated with the current PCID.

A processor may or may not implement any of the paging-structure caches. Software should rely on neither their presence nor their absence. The processor may invalidate entries in these caches at any time. Because the processor may create the cache entries at the time of translation and not update them following subsequent modifications to the paging structures in memory, software should take care to invalidate the cache entries appropriately when causing such modifications. The invalidation of TLBs and the paging-structure caches is described in Section 4.10.4.

4.10.3.2 Using the Paging-Structure Caches to Translate Linear Addresses

When a linear address is accessed, the processor uses a procedure such as the following to determine the physical address to which it translates and whether the access should be allowed:

- If the processor finds a TLB entry that is for the page number of the linear address and that is associated with the current PCID (or which is global), it may use the physical address, access rights, and other attributes from that entry.
- If the processor does not find a relevant TLB entry, it may use the upper bits of the linear address to select an entry from the PDE cache that is associated with the current PCID (Section 4.10.3.1 indicates which bits are used in each paging mode). It can then use that entry to complete the translation process (locating a PTE, etc.) as if it had traversed the PDE (and, for 4-level paging and 5-level paging, the PDPTE, PML4E, and PML5E, as appropriate) corresponding to the PDE-cache entry.
- The following items apply when 4-level paging or 5-level paging is used:
 - If the processor does not find a relevant TLB entry or PDE-cache entry, it may use the upper bits of the linear address (for 4-level paging, bits 47:30; for 5-level paging, bits 56:30) to select an entry from the PDPTE cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDE, etc.) as if it had traversed the PDPTE, the PML4E, and (for 5-level paging) the PML5E corresponding to the PDPTE-cache entry.
 - If the processor does not find a relevant TLB entry, PDE-cache entry, or PDPTE-cache entry, it may use the upper bits of the linear address (for 4-level paging, bits 47:39; for 5-level paging, bits 56:39) to select an entry from the PML4E cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDPTE, etc.) as if it had traversed the corresponding PML4E.
 - With 5-level paging, if the processor does not find a relevant TLB entry, PDE-cache entry, PDPTE-cache entry, or PML4E-cache entry, it may use bits 56:48 of the linear address to select an entry from the PML5E cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PML4E, etc.) as if it had traversed the corresponding PML5E.

(Any of the above steps would be skipped if the processor does not support the cache in question.)

If the processor does not find a TLB or paging-structure-cache entry for the linear address, it uses the linear address to traverse the entire paging-structure hierarchy, as described in Section 4.3, Section 4.4.2, and Section 4.5.

4.10.3.3 Multiple Cached Entries for a Single Paging-Structure Entry

The paging-structure caches and TLBs may contain multiple entries associated with a single PCID and with information derived from a single paging-structure entry. The following items give some examples for 4-level paging:

- Suppose that two PML4Es contain the same physical address and thus reference the same page-directory-pointer table. Any PDPTE in that table may result in two PDPTE-cache entries, each associated with a different set of linear addresses. Specifically, suppose that the n_1^{th} and n_2^{th} entries in the PML4 table contain the same physical address. This implies that the physical address in the m^{th} PDPTE in the page-directory-pointer table would appear in the PDPTE-cache entries associated with both p_1 and p_2 , where $(p_1 \gg 9) = n_1$, $(p_2 \gg 9) = n_2$, and $(p_1 \& 1FFH) = (p_2 \& 1FFH) = m$. This is because both PDPTE-cache entries use the same PDPTE, one resulting from a reference from the n_1^{th} PML4E and one from the n_2^{th} PML4E.
- Suppose that the first PML4E (i.e., the one in position 0) contains the physical address X in CR3 (the physical address of the PML4 table). This implies the following:

- Any PML4-cache entry associated with linear addresses with 0 in bits 47:39 contains address X.
- Any PDPTTE-cache entry associated with linear addresses with 0 in bits 47:30 contains address X. This is because the translation for a linear address for which the value of bits 47:30 is 0 uses the value of bits 47:39 (0) to locate a page-directory-pointer table at address X (the address of the PML4 table). It then uses the value of bits 38:30 (also 0) to find address X again and to store that address in the PDPTTE-cache entry.
- Any PDE-cache entry associated with linear addresses with 0 in bits 47:21 contains address X for similar reasons.
- Any TLB entry for page number 0 (associated with linear addresses with 0 in bits 47:12) translates to page frame $X \gg 12$ for similar reasons.

The same PML4E contributes its address X to all these cache entries because the self-referencing nature of the entry causes it to be used as a PML4E, a PDPTTE, a PDE, and a PTE.

4.10.4 Invalidation of TLBs and Paging-Structure Caches

As noted in Section 4.10.2 and Section 4.10.3, the processor may create entries in the TLBs and the paging-structure caches when linear addresses are translated, and it may retain these entries even after the paging structures used to create them have been modified. To ensure that linear-address translation uses the modified paging structures, software should take action to invalidate any cached entries that may contain information that has since been modified.

4.10.4.1 Operations that Invalidate TLBs and Paging-Structure Caches

The following instructions invalidate entries in the TLBs and the paging-structure caches:

- **INVLPG.** This instruction takes a single operand, which is a linear address. The instruction invalidates any TLB entries that are for a page number corresponding to the linear address and that are associated with the current PCID. It also invalidates any global TLB entries with that page number, regardless of PCID (see Section 4.10.2.4).¹ INVLPG also invalidates all entries in all paging-structure caches associated with the current PCID, regardless of the linear addresses to which they correspond.
- **INVPCID.** The operation of this instruction is based on instruction operands, called the INVPCID type and the INVPCID descriptor. Four INVPCID types are currently defined:
 - **Individual-address.** If the INVPCID type is 0, the logical processor invalidates mappings—except global translations—associated with the PCID specified in the INVPCID descriptor and that would be used to translate the linear address specified in the INVPCID descriptor.² (The instruction may also invalidate global translations, as well as mappings associated with other PCIDs and for other linear addresses.)
 - **Single-context.** If the INVPCID type is 1, the logical processor invalidates all mappings—except global translations—associated with the PCID specified in the INVPCID descriptor. (The instruction may also invalidate global translations, as well as mappings associated with other PCIDs.)
 - **All-context, including globals.** If the INVPCID type is 2, the logical processor invalidates mappings—including global translations—associated with all PCIDs.
 - **All-context.** If the INVPCID type is 3, the logical processor invalidates mappings—except global translations—associated with all PCIDs. (The instruction may also invalidate global translations.)

See Chapter 3 of the *Intel 64 and IA-32 Architecture Software Developer's Manual, Volume 2A* for details of the INVPCID instruction.

- **MOV to CR0.** The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if it changes the value of CR0.PG from 1 to 0.
- **MOV to CR3.** The behavior of the instruction depends on the value of CR4.PCIDE:

1. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 4.10.2.3), the instruction invalidates all of them.
2. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 4.10.2.3), the instruction invalidates all of them.

- If CR4.PCIDE = 0, the instruction invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.
- If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, the instruction invalidates all TLB entries associated with the PCID specified in bits 11:0 of the instruction's source operand except those for global pages. It also invalidates all entries in all paging-structure caches associated with that PCID. It is not required to invalidate entries in the TLBs and paging-structure caches that are associated with other PCIDs.
- If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1, the instruction is not required to invalidate any TLB entries or entries in paging-structure caches.
- MOV to CR4. The behavior of the instruction depends on the bits being modified:
 - The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if (1) it changes the value of CR4.PGE;¹ or (2) it changes the value of the CR4.PCIDE from 1 to 0.
 - The instruction invalidates all TLB entries and all entries in all paging-structure caches for the current PCID if (1) it changes the value of CR4.PAE; or (2) it changes the value of CR4.SMEP from 0 to 1.
- Task switch. If a task switch changes the value of CR3, it invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.²
- VMX transitions. See Section 4.11.1.

The processor is always free to invalidate additional entries in the TLBs and paging-structure caches. The following are some examples:

- INVLPG may invalidate TLB entries for pages other than the one corresponding to its linear-address operand. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the current PCID.
- INVPCID may invalidate TLB entries for pages other than the one corresponding to the specified linear address. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the specified PCID.
- MOV to CR0 may invalidate TLB entries even if CR0.PG is not changing. For example, this may occur if either CR0.CD or CR0.NW is modified.
- MOV to CR3 may invalidate TLB entries for global pages. If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, it may invalidate TLB entries and entries in the paging-structure caches associated with PCIDs other than the PCID it is establishing. It may invalidate entries if CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1.
- MOV to CR4 may invalidate TLB entries when changing CR4.PSE or when changing CR4.SMEP from 1 to 0.
- On a processor supporting Hyper-Threading Technology, invalidations performed on one logical processor may invalidate entries in the TLBs and paging-structure caches used by other logical processors.

(Other instructions and operations may invalidate entries in the TLBs and the paging-structure caches, but the instructions identified above are recommended.)

In addition to the instructions identified above, page faults invalidate entries in the TLBs and paging-structure caches. In particular, a page-fault exception resulting from an attempt to use a linear address will invalidate any TLB entries that are for a page number corresponding to that linear address and that are associated with the current PCID. It also invalidates all entries in the paging-structure caches that would be used for that linear address and that are associated with the current PCID.³ These invalidations ensure that the page-fault exception will not recur (if the faulting instruction is re-executed) if it would not be caused by the contents of the paging structures

-
1. If CR4.PGE is changing from 0 to 1, there were no global TLB entries before the execution; if CR4.PGE is changing from 1 to 0, there will be no global TLB entries after the execution.
 2. Task switches do not occur in IA-32e mode and thus cannot occur with 4-level paging. Since CR4.PCIDE can be set only with 4-level paging, task switches occur only with CR4.PCIDE = 0.
 3. Unlike INVLPG, page faults need not invalidate **all** entries in the paging-structure caches, only those that would be used to translate the faulting linear address.

in memory (and if, therefore, it resulted from cached entries that were not invalidated after the paging structures were modified in memory).

As noted in Section 4.10.2, some processors may choose to cache multiple smaller-page TLB entries for a translation specified by the paging structures to use a page larger than 4 KBytes. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. The INVLPG instruction and page faults provide the same assurances that they provide when a single TLB entry is used: they invalidate all TLB entries corresponding to the translation specified by the paging structures.

4.10.4.2 Recommended Invalidation

The following items provide some recommendations regarding when software should perform invalidations:

- If software modifies a paging-structure entry that maps a page (rather than referencing another paging structure), it should execute INVLPG for any linear address with a page number whose translation uses that paging-structure entry.¹

(If the paging-structure entry may be used in the translation of different page numbers — see Section 4.10.3.3 — software should execute INVLPG for linear addresses with each of those page numbers; alternatively, it could use MOV to CR3 or MOV to CR4.)
- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:
 - Execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry. However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute INVLPG at least once.
 - Execute MOV to CR3 if the modified entry controls no global pages.
 - Execute MOV to CR4 to modify CR4.PGE.
- If CR4.PCIDE = 1 and software modifies a paging-structure entry that does not map a page or in which the G flag (bit 8) is 0, additional steps are required if the entry may be used for PCIDs other than the current one. Any one of the following suffices:
 - Execute MOV to CR4 to modify CR4.PGE, either immediately or before again using any of the affected PCIDs. For example, software could use different (previously unused) PCIDs for the processes that used the affected PCIDs.
 - For each affected PCID, execute MOV to CR3 to make that PCID current (and to load the address of the appropriate PML4 table). If the modified entry controls no global pages and bit 63 of the source operand to MOV to CR3 was 0, no further steps are required. Otherwise, execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry; if no page numbers that would use the entry have translations, execute INVLPG at least once.
- If software using PAE paging modifies a PDPTE, it should reload CR3 with the register's current value to ensure that the modified PDPTE is loaded into the corresponding PDPTE register (see Section 4.4.1).
- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 4.10.3.3), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute INVLPG with two (or more) linear addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use MOV to CR3 or MOV to CR4.)
- As noted in Section 4.10.2, the TLBs may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes. A reference to a linear address in the address range may use any of these translations.

Software wishing to prevent this uncertainty should not write to a paging-structure entry in a way that would change, for any linear address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g.,

1. One execution of INVLPG is sufficient even for a page with size greater than 4 KBytes.

PDE); then invalidate any translations for the affected linear addresses (see above); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.

- Software should clear bit 63 of the source operand to a MOV to CR3 instruction that establishes a PCID that had been used earlier for a different linear-address space (e.g., with a different value in bits 51:12 of CR3). This ensures invalidation of any information that may have been cached for the previous linear-address space.

This assumes that both linear-address spaces use the same global pages and that it is thus not necessary to invalidate any global TLB entries. If that is not the case, software should invalidate those entries by executing MOV to CR4 to modify CR4.PGE.

4.10.4.3 Optional Invalidation

The following items describe cases in which software may choose not to invalidate and the potential consequences of that choice:

- If a paging-structure entry is modified to change the P flag from 0 to 1, no invalidation is necessary. This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the P flag is 0.¹
- If a paging-structure entry is modified to change the accessed flag from 0 to 1, no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, failure to perform an invalidation may result in a “spurious” page-fault exception (e.g., in response to an attempted write access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If CR4.SMEP = 0 and a paging-structure entry is modified to change the U/S flag from 0 to 1, failure to perform an invalidation may result in a “spurious” page-fault exception (e.g., in response to an attempted user-mode access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If a paging-structure entry is modified to change the XD flag from 1 to 0, failure to perform an invalidation may result in a “spurious” page-fault exception (e.g., in response to an attempted instruction fetch) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If a paging-structure entry is modified to change the accessed flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent access to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such an access has not occurred.
- If software modifies a paging-structure entry that identifies the final physical address for a linear address (either a PTE or a paging-structure entry in which the PS flag is 1) to change the dirty flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent write to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such a write has not occurred.
- The read of a paging-structure entry in translating an address being used to fetch an instruction may appear to execute before an earlier write to that paging-structure entry if there is no serializing instruction between the write and the instruction fetch. Note that the invalidating instructions identified in Section 4.10.4.1 are all serializing instructions.
- Section 4.10.3.3 describes situations in which a single paging-structure entry may contain information cached in multiple entries in the paging-structure caches. Because all entries in these caches are invalidated by any execution of INVLPG, it is not necessary to follow the modification of such a paging-structure entry by executing INVLPG multiple times solely for the purpose of invalidating these multiple cached entries. (It may be necessary to do so to invalidate multiple TLB entries.)

1. If it is also the case that no invalidation was performed the last time the P flag was changed from 1 to 0, the processor may use a TLB entry or paging-structure cache entry that was created when the P flag had earlier been 1.

4.10.4.4 Delayed Invalidation

Required invalidations may be delayed under some circumstances. Software developers should understand that, between the modification of a paging-structure entry and execution of the invalidation instruction recommended in Section 4.10.4.2, the processor may use translations based on either the old value or the new value of the paging-structure entry. The following items describe some of the potential consequences of delayed invalidation:

- If a paging-structure entry is modified to change the P flag from 1 to 0, an access to a linear address whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, write accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the U/S flag from 0 to 1, user-mode accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the XD flag from 1 to 0, instruction fetches from linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.

As noted in Section 8.1.1, an x87 instruction or an SSE instruction that accesses data larger than a quadword may be implemented using multiple memory accesses. If such an instruction stores to memory and invalidation has been delayed, some of the accesses may complete (writing to memory) while another causes a page-fault exception.¹ In this case, the effects of the completed accesses may be visible to software even though the overall instruction caused a fault.

In some cases, the consequences of delayed invalidation may not affect software adversely. For example, when freeing a portion of the linear-address space (by marking paging-structure entries “not present”), invalidation using INVLPG may be delayed if software does not re-allocate that portion of the linear-address space or the memory that had been associated with it. However, because of speculative execution (or errant software), there may be accesses to the freed portion of the linear-address space before the invalidations occur. In this case, the following can happen:

- Reads can occur to the freed portion of the linear-address space. Therefore, invalidation should not be delayed for an address range that has read side effects.
- The processor may retain entries in the TLBs and paging-structure caches for an extended period of time. Software should not assume that the processor will not use entries associated with a linear address simply because time has passed.
- As noted in Section 4.10.3.1, the processor may create an entry in a paging-structure cache even if there are no translations for any linear address that might use that entry. Thus, if software has marked “not present” all entries in a page table, the processor may subsequently create a PDE-cache entry for the PDE that references that page table (assuming that the PDE itself is marked “present”).
- If software attempts to write to the freed portion of the linear-address space, the processor might not generate a page fault. (Such an attempt would likely be the result of a software error.) For that reason, the page frames previously associated with the freed portion of the linear-address space should not be reallocated for another purpose until the appropriate invalidations have been performed.

4.10.5 Propagation of Paging-Structure Changes to Multiple Processors

As noted in Section 4.10.4, software that modifies a paging-structure entry may need to invalidate entries in the TLBs and paging-structure caches that were derived from the modified entry before it was modified. In a system containing more than one logical processor, software must account for the fact that there may be entries in the TLBs and paging-structure caches of logical processors other than the one used to modify the paging-structure entry. The process of propagating the changes to a paging-structure entry is commonly referred to as “TLB shutdown.”

TLB shutdown can be done using memory-based semaphores and/or interprocessor interrupts (IPI). The following items describe a simple but inefficient example of a TLB shutdown algorithm for processors supporting the Intel-64 and IA-32 architectures:

1. If the accesses are to different pages, this may occur even if invalidation has not been delayed.

1. Begin barrier: Stop all but one logical processor; that is, cause all but one to execute the HLT instruction or to enter a spin loop.
2. Allow the active logical processor to change the necessary paging-structure entries.
3. Allow all logical processors to perform invalidations appropriate to the modifications to the paging-structure entries.
4. Allow all logical processors to resume normal operation.

Alternative, performance-optimized, TLB shutdown algorithms may be developed; however, software developers must take care to ensure that the following conditions are met:

- All logical processors that are using the paging structures that are being modified must participate and perform appropriate invalidations after the modifications are made.
- If the modifications to the paging-structure entries are made before the barrier or if there is no barrier, the operating system must ensure one of the following: (1) that the affected linear-address range is not used between the time of modification and the time of invalidation; or (2) that it is prepared to deal with the consequences of the affected linear-address range being used during that period. For example, if the operating system does not allow pages being freed to be reallocated for another purpose until after the required invalidations, writes to those pages by errant software will not unexpectedly modify memory that is in use.
- Software must be prepared to deal with reads, instruction fetches, and prefetch requests to the affected linear-address range that are a result of speculative execution that would never actually occur in the executed code path.

When multiple logical processors are using the same linear-address space at the same time, they must coordinate before any request to modify the paging-structure entries that control that linear-address space. In these cases, the barrier in the TLB shutdown routine may not be required. For example, when freeing a range of linear addresses, some other mechanism can assure no logical processor is using that range before the request to free it is made. In this case, a logical processor freeing the range can clear the P flags in the PTEs associated with the range, free the physical page frames associated with the range, and then signal the other logical processors using that linear-address space to perform the necessary invalidations. All the affected logical processors must complete their invalidations before the linear-address range and the physical page frames previously associated with that range can be reallocated.

4.11 INTERACTIONS WITH VIRTUAL-MACHINE EXTENSIONS (VMX)

The architecture for virtual-machine extensions (VMX) includes features that interact with paging. Section 4.11.1 discusses ways in which VMX-specific control transfers, called VMX transitions specially affect paging. Section 4.11.2 gives an overview of VMX features specifically designed to support address translation.

4.11.1 VMX Transitions

The VMX architecture defines two control transfers called **VM entries** and **VM exits**; collectively, these are called **VMX transitions**. VM entries and VM exits are described in detail in Chapter 26 and Chapter 27, respectively, in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. The following items identify paging-related details:

- VMX transitions modify the CR0 and CR4 registers and the IA32_EFER MSR concurrently. For this reason, they allow transitions between paging modes that would not otherwise be possible:
 - VM entries allow transitions from 4-level paging directly to either 32-bit paging or PAE paging.
 - VM exits allow transitions from either 32-bit paging or PAE paging directly to 4-level paging or 5-level paging.
- VMX transitions that result in PAE paging load the PDPTTE registers (see Section 4.4.1) as follows:
 - VM entries load the PDPTTE registers either from the physical address being loaded into CR3 or from the virtual-machine control structure (VMCS); see Section 26.3.2.4.
 - VM exits load the PDPTTE registers from the physical address being loaded into CR3; see Section 27.5.4.

- VMX transitions invalidate the TLBs and paging-structure caches based on certain control settings. See Section 26.3.2.5 and Section 27.5.5 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

4.11.2 VMX Support for Address Translation

Chapter 28, “VMX Support for Address Translation,” in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* describe two features of the virtual-machine extensions (VMX) that interact directly with paging. These are **virtual-processor identifiers (VPIDs)** and the **extended page table** mechanism (**EPT**).

VPIDs provide a way for software to identify to the processor the address spaces for different “virtual processors.” The processor may use this identification to maintain concurrently information for multiple address spaces in its TLBs and paging-structure caches, even when non-zero PCIDs are not being used. See Section 28.1 for details.

When EPT is in use, the addresses in the paging-structures are not used as physical addresses to access memory and memory-mapped I/O. Instead, they are treated as **guest-physical** addresses and are translated through a set of EPT paging structures to produce physical addresses. EPT can also specify its own access rights and memory typing; these are used on conjunction with those specified in this chapter. See Section 28.3 for more information.

Both VPIDs and EPT may change the way that a processor maintains information in TLBs and paging structure caches and the ways in which software can manage that information. Some of the behaviors documented in Section 4.10 may change. See Section 28.4 for details.

4.12 USING PAGING FOR VIRTUAL MEMORY

With paging, portions of the linear-address space need not be mapped to the physical-address space; data for the unmapped addresses can be stored externally (e.g., on disk). This method of mapping the linear-address space is referred to as virtual memory or demand-paged virtual memory.

Paging divides the linear address space into fixed-size pages that can be mapped into the physical-address space and/or external storage. When a program (or task) references a linear address, the processor uses paging to translate the linear address into a corresponding physical address if such an address is defined.

If the page containing the linear address is not currently mapped into the physical-address space, the processor generates a page-fault exception as described in Section 4.7. The handler for page-fault exceptions typically directs the operating system or executive to load data for the unmapped page from external storage into physical memory (perhaps writing a different page from physical memory out to external storage in the process) and to map it using paging (by updating the paging structures). When the page has been loaded into physical memory, a return from the exception handler causes the instruction that generated the exception to be restarted.

Paging differs from segmentation through its use of fixed-size pages. Unlike segments, which usually are the same size as the code or data structures they hold, pages have a fixed size. If segmentation is the only form of address translation used, a data structure present in physical memory will have all of its parts in memory. If paging is used, a data structure can be partly in memory and partly in disk storage.

4.13 MAPPING SEGMENTS TO PAGES

The segmentation and paging mechanisms provide support for a wide variety of approaches to memory management. When segmentation and paging are combined, segments can be mapped to pages in several ways. To implement a flat (unsegmented) addressing environment, for example, all the code, data, and stack modules can be mapped to one or more large segments (up to 4-GBytes) that share same range of linear addresses (see Figure 3-2 in Section 3.2.2). Here, segments are essentially invisible to applications and the operating-system or executive. If paging is used, the paging mechanism can map a single linear-address space (contained in a single segment) into virtual memory. Alternatively, each program (or task) can have its own large linear-address space (contained in its own segment), which is mapped into virtual memory through its own paging structures.

Segments can be smaller than the size of a page. If one of these segments is placed in a page which is not shared with another segment, the extra memory is wasted. For example, a small data structure, such as a 1-Byte sema-

phore, occupies 4 KBytes if it is placed in a page by itself. If many semaphores are used, it is more efficient to pack them into a single page.

The Intel-64 and IA-32 architectures do not enforce correspondence between the boundaries of pages and segments. A page can contain the end of one segment and the beginning of another. Similarly, a segment can contain the end of one page and the beginning of another.

Memory-management software may be simpler and more efficient if it enforces some alignment between page and segment boundaries. For example, if a segment which can fit in one page is placed in two pages, there may be twice as much paging overhead to support access to that segment.

One approach to combining paging and segmentation that simplifies memory-management software is to give each segment its own page table, as shown in Figure 4-13. This convention gives the segment a single entry in the page directory, and this entry provides the access control information for paging the entire segment.

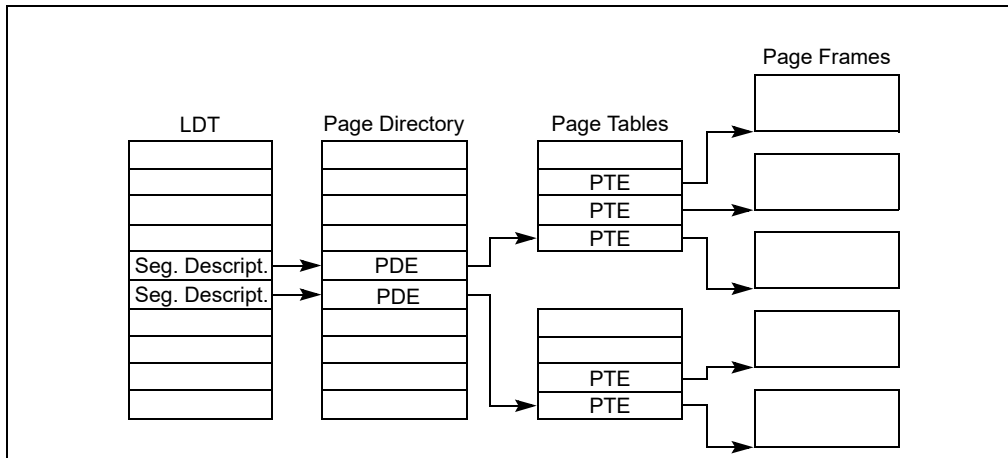


Figure 4-13. Memory Management Convention That Assigns a Page Table to Each Segment

9. Updates to Chapter 14, Volume 3B

Change bars and green text show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter: Updates to various locations in section 14.6, "Hardware Feedback Interface and Intel® Thread Director", to correct typos and add support for Energy Efficiency capability.

This chapter describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

14.1 ENHANCED INTEL SPEEDSTEP® TECHNOLOGY

Enhanced Intel SpeedStep® Technology was introduced in the Pentium M processor. The technology enables the management of processor power consumption via performance state transitions. These states are defined as discrete operating points associated with different voltages and frequencies.

Enhanced Intel SpeedStep Technology differs from previous generations of Intel SpeedStep® Technology in two ways:

- Centralization of the control mechanism and software interface in the processor by using model-specific registers.
- Reduced hardware overhead; this permits more frequent performance state transitions.

Previous generations of the Intel SpeedStep Technology require processors to be a deep sleep state, holding off bus master transfers for the duration of a performance state transition. Performance state transitions under the Enhanced Intel SpeedStep Technology are discrete transitions to a new target frequency.

Support is indicated by CPUID, using ECX feature bit 07. Enhanced Intel SpeedStep Technology is enabled by setting IA32_MISC_ENABLE MSR, bit 16. On reset, bit 16 of IA32_MISC_ENABLE MSR is cleared.

14.1.1 Software Interface For Initiating Performance State Transitions

State transitions are initiated by writing a 16-bit value to the IA32_PERF_CTL register, see Figure 14-2. If a transition is already in progress, transition to a new value will subsequently take effect.

Reads of IA32_PERF_CTL determine the last targeted operating point. The current operating point can be read from IA32_PERF_STATUS. IA32_PERF_STATUS is updated dynamically.

The 16-bit encoding that defines valid operating points is model-specific. Applications and performance tools are not expected to use either IA32_PERF_CTL or IA32_PERF_STATUS and should treat both as reserved. Performance monitoring tools can access model-specific events and report the occurrences of state transitions.

14.2 P-STATE HARDWARE COORDINATION

The Advanced Configuration and Power Interface (ACPI) defines performance states (P-states) that are used to facilitate system software's ability to manage processor power consumption. Different P-states correspond to different performance levels that are applied while the processor is actively executing instructions. Enhanced Intel SpeedStep Technology supports P-states by providing software interfaces that control the operating frequency and voltage of a processor.

With multiple processor cores residing in the same physical package, hardware dependencies may exist for a subset of logical processors on a platform. These dependencies may impose requirements that impact the coordination of P-state transitions. As a result, multi-core processors may require an OS to provide additional software support for coordinating P-state transitions for those subsets of logical processors.

ACPI firmware can choose to expose P-states as dependent and hardware-coordinated to OS power management (OSPM) policy. To support OSPMs, multi-core processors must have additional built-in support for P-state hardware coordination and feedback.

Intel 64 and IA-32 processors with dependent P-states amongst a subset of logical processors permit hardware coordination of P-states and provide a hardware-coordination feedback mechanism using IA32_MPERF MSR and

IA32_APERF MSR. See Figure 14-1 for an overview of the two 64-bit MSRs and the bullets below for a detailed description.

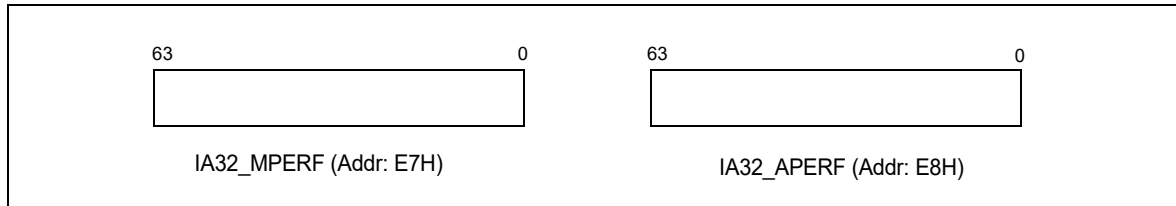


Figure 14-1. IA32_MPERF MSR and IA32_APERF MSR for P-state Coordination

- Use CPUID to check the P-State hardware coordination feedback capability bit. CPUID.06H.ECX[Bit 0] = 1 indicates IA32_MPERF MSR and IA32_APERF MSR are present.
- IA32_MPERF MSR (E7H) increments in proportion to a fixed frequency, which is configured when the processor is booted.
- IA32_APERF MSR (E8H) increments in proportion to actual performance, while accounting for hardware coordination of P-state and TM1/TM2; or software initiated throttling.
- The MSRs are per logical processor; they measure performance only when the targeted processor is in the C0 state.
- Only the IA32_APERF/IA32_MPERF ratio is architecturally defined; software should not attach meaning to the content of the individual of IA32_APERF or IA32_MPERF MSRs.
- When either MSR overflows, both MSRs are reset to zero and continue to increment.
- Both MSRs are full 64-bits counters. Each MSR can be written to independently. However, software should follow the guidelines illustrated in Example 14-1.

If P-states are exposed by the BIOS as hardware coordinated, software is expected to confirm processor support for P-state hardware coordination feedback and use the feedback mechanism to make P-state decisions. The OSPM is expected to either save away the current MSR values (for determination of the delta of the counter ratio at a later time) or reset both MSRs (execute WRMSR with 0 to these MSRs individually) at the start of the time window used for making the P-state decision. When not resetting the values, overflow of the MSRs can be detected by checking whether the new values read are less than the previously saved values.

Example 14-1 demonstrates steps for using the hardware feedback mechanism provided by IA32_APERF MSR and IA32_MPERF MSR to determine a target P-state.

Example 14-1. Determine Target P-state From Hardware Coordinated Feedback

```
DWORD PercentBusy; // Percentage of processor time not idle.
// Measure "PercentBusy" during previous sampling window.
// Typically, "PercentBusy" is measure over a time scale suitable for
// power management decisions
//
// RDMSR of MCNT and ACNT should be performed without delay.
// Software needs to exercise care to avoid delays between
// the two RDMSRs (for example, interrupts).
MCNT = RDMSR(IA32_MPERF);
ACNT = RDMSR(IA32_APERF);

// PercentPerformance indicates the percentage of the processor
// that is in use. The calculation is based on the PercentBusy,
// that is the percentage of processor time not idle and the P-state
// hardware coordinated feedback using the ACNT/MCNT ratio.
// Note that both values need to be calculated over the same
```



```

// time window.
    PercentPerformance = PercentBusy * (ACNT/MCNT);

// This example does not cover the additional logic or algorithms
// necessary to coordinate multiple logical processors to a target P-state.

TargetPstate = FindPstate(PercentPerformance);

if (TargetPstate ≠ currentPstate) {
    SetPState(TargetPstate);
}
// WRMSR of MCNT and ACNT should be performed without delay.
// Software needs to exercise care to avoid delays between
// the two WRMSRs (for example, interrupts).
WRMSR(IA32_MPERF, 0);
WRMSR(IA32_APERF, 0);

```

14.3 SYSTEM SOFTWARE CONSIDERATIONS AND OPPORTUNISTIC PROCESSOR PERFORMANCE OPERATION

An Intel 64 processor may support a form of processor operation that takes advantage of design headroom to opportunistically increase performance. The Intel[®] Turbo Boost Technology can convert thermal headroom into higher performance across multi-threaded and single-threaded workloads. The Intel[®] Dynamic Acceleration Technology feature can convert thermal headroom into higher performance if only one thread is active.

14.3.1 Intel[®] Dynamic Acceleration Technology

The Intel Core 2 Duo processor T 7700 introduces Intel Dynamic Acceleration Technology. Intel Dynamic Acceleration Technology takes advantage of thermal design headroom and opportunistically allows a single core to operate at a higher performance level when the operating system requests increased performance.

14.3.2 System Software Interfaces for Opportunistic Processor Performance Operation

Opportunistic processor performance operation, applicable to Intel Dynamic Acceleration Technology and Intel[®] Turbo Boost Technology, has the following characteristics:

- A transition from a normal state of operation (e.g., Intel Dynamic Acceleration Technology/Turbo mode disengaged) to a target state is not guaranteed, but may occur opportunistically after the corresponding enable mechanism is activated, the headroom is available and certain criteria are met.
- The opportunistic processor performance operation is generally transparent to most application software.
- System software (BIOS and Operating system) must be aware of hardware support for opportunistic processor performance operation and may need to temporarily disengage opportunistic processor performance operation when it requires more predictable processor operation.
- When opportunistic processor performance operation is engaged, the OS should use hardware coordination feedback mechanisms to prevent un-intended policy effects if it is activated during inappropriate situations.

14.3.2.1 Discover Hardware Support and Enabling of Opportunistic Processor Performance Operation

If an Intel 64 processor has hardware support for opportunistic processor performance operation, the power-on default state of IA32_MISC_ENABLE[38] indicates the presence of such hardware support. For Intel 64 processors that support opportunistic processor performance operation, the default value is 1, indicating its presence. For processors that do not support opportunistic processor performance operation, the default value is 0. The power-

on default value of IA32_MISC_ENABLE[38] allows BIOS to detect the presence of hardware support of opportunistic processor performance operation.

IA32_MISC_ENABLE[38] is shared across all logical processors in a physical package. It is written by BIOS during platform initiation to enable/disable opportunistic processor performance operation in conjunction of OS power management capabilities, see Section 14.3.2.2. BIOS can set IA32_MISC_ENABLE[38] with 1 to disable opportunistic processor performance operation; it must clear the default value of IA32_MISC_ENABLE[38] to 0 to enable opportunistic processor performance operation. OS and applications must use CPUID leaf 06H if it needs to detect processors that have opportunistic processor performance operation enabled.

When CPUID is executed with EAX = 06H on input, Bit 1 of EAX in Leaf 06H (i.e., CPUID.06H:EAX[1]) indicates opportunistic processor performance operation, such as Intel Dynamic Acceleration Technology, has been enabled by BIOS.

Opportunistic processor performance operation can be disabled by setting bit 38 of IA32_MISC_ENABLE. This mechanism is intended for BIOS only. If IA32_MISC_ENABLE[38] is set, CPUID.06H:EAX[1] will return 0.

14.3.2.2 OS Control of Opportunistic Processor Performance Operation

There may be phases of software execution in which system software cannot tolerate the non-deterministic aspects of opportunistic processor performance operation. For example, when calibrating a real-time workload to make a CPU reservation request to the OS, it may be undesirable to allow the possibility of the processor delivering increased performance that cannot be sustained after the calibration phase.

System software can temporarily disengage opportunistic processor performance operation by setting bit 32 of the IA32_PERF_CTL MSR (0199H), using a read-modify-write sequence on the MSR. The opportunistic processor performance operation can be re-engaged by clearing bit 32 in IA32_PERF_CTL MSR, using a read-modify-write sequence. The DISENAGE bit in IA32_PERF_CTL is not reflected in bit 32 of the IA32_PERF_STATUS MSR (0198H), and it is not shared between logical processors in a physical package. In order for OS to engage Intel Dynamic Acceleration Technology/Turbo mode, the BIOS must:

- Enable opportunistic processor performance operation, as described in Section 14.3.2.1.
- Expose the operating points associated with Intel Dynamic Acceleration Technology/Turbo mode to the OS.

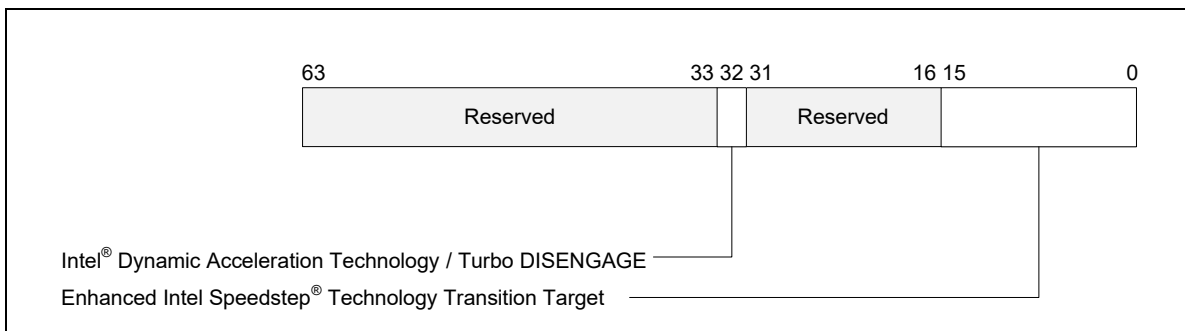


Figure 14-2. IA32_PERF_CTL Register

14.3.2.3 Required Changes to OS Power Management P-State Policy

Intel Dynamic Acceleration Technology and Intel Turbo Boost Technology can provide opportunistic performance greater than the performance level corresponding to the Processor Base frequency of the processor (see CPUID's processor frequency information). System software can use a pair of MSRs to observe performance feedback. Software must query for the presence of IA32_APERF and IA32_MPERF (see Section 14.2). The ratio between IA32_APERF and IA32_MPERF is architecturally defined and a value greater than unity indicates performance increase occurred during the observation period due to Intel Dynamic Acceleration Technology. Without incorporating such performance feedback, the target P-state evaluation algorithm can result in a non-optimal P-state target.

There are other scenarios under which OS power management may want to disable Intel Dynamic Acceleration Technology, some of these are listed below:

- When engaging ACPI defined passive thermal management, it may be more effective to disable Intel Dynamic Acceleration Technology for the duration of passive thermal management.
- When the user has indicated a policy preference of power savings over performance, OS power management may want to disable Intel Dynamic Acceleration Technology while that policy is in effect.

14.3.3 Intel® Turbo Boost Technology

Intel Turbo Boost Technology is supported in Intel Core i7 processors and Intel Xeon processors based on Nehalem microarchitecture. It uses the same principle of leveraging thermal headroom to dynamically increase processor performance for single-threaded and multi-threaded/multi-tasking environment. The programming interface described in Section 14.3.2 also applies to Intel Turbo Boost Technology.

14.3.4 Performance and Energy Bias Hint Support

Intel 64 processors may support additional software hint to guide the hardware heuristic of power management features to favor increasing dynamic performance or conserve energy consumption.

Software can detect the processor's capability to support the performance-energy bias preference hint by examining bit 3 of ECX in CPUID leaf 6. The processor supports this capability if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H).

Software can program the lowest four bits of IA32_ENERGY_PERF_BIAS MSR with a value from 0 - 15. The values represent a sliding scale, where a value of 0 (the default reset value) corresponds to a hint preference for highest performance and a value of 15 corresponds to the maximum energy savings. A value of 7 roughly translates into a hint to balance performance with energy consumption.

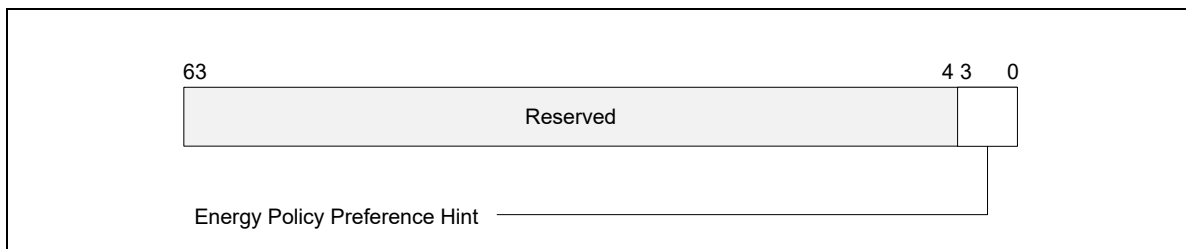


Figure 14-3. IA32_ENERGY_PERF_BIAS Register

The layout of IA32_ENERGY_PERF_BIAS is shown in Figure 14-3. The scope of IA32_ENERGY_PERF_BIAS is per logical processor, which means that each of the logical processors in the package can be programmed with a different value. This may be especially important in virtualization scenarios, where the performance / energy requirements of one logical processor may differ from the other. Conflicting "hints" from various logical processors at higher hierarchy level will be resolved in favor of performance over energy savings.

Software can use whatever criteria it sees fit to program the MSR with an appropriate value. However, the value only serves as a hint to the hardware and the actual impact on performance and energy savings is model specific.

14.4 HARDWARE-CONTROLLED PERFORMANCE STATES (HWP)

Intel processors may contain support for Hardware-Controlled Performance States (HWP), which autonomously selects performance states while utilizing OS supplied performance guidance hints. The Enhanced Intel Speed-Step® Technology provides a means for the OS to control and monitor discrete frequency-based operating points via the IA32_PERF_CTL and IA32_PERF_STATUS MSRs.

In contrast, HWP is an implementation of the ACPI-defined Collaborative Processor Performance Control (CPPC), which specifies that the platform enumerates a continuous, abstract unit-less, performance value scale that is not tied to a specific performance state / frequency by definition. While the enumerated scale is roughly linear in terms of a delivered integer workload performance result, the OS is required to characterize the performance value range to comprehend the delivered performance for an applied workload.

When HWP is enabled, the processor autonomously selects performance states as deemed appropriate for the applied workload and with consideration of constraining hints that are programmed by the OS. These OS-provided hints include minimum and maximum performance limits, preference towards energy efficiency or performance, and the specification of a relevant workload history observation time window. The means for the OS to override HWP's autonomous selection of performance state with a specific desired performance target is also provided, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations.

14.4.1 HWP Programming Interfaces

The programming interfaces provided by HWP include the following:

- The CPUID instruction allows software to discover the presence of HWP support in an Intel processor. Specifically, execute CPUID instruction with EAX=06H as input will return 5 bit flags covering the following aspects in bits 7 through 11 of CPUID.06H:EAX:
 - Availability of HWP baseline resource and capability, CPUID.06H:EAX[bit 7]: If this bit is set, HWP provides several new architectural MSRs: IA32_PM_ENABLE, IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS.
 - Availability of HWP Notification upon dynamic Guaranteed Performance change, CPUID.06H:EAX[bit 8]: If this bit is set, HWP provides IA32_HWP_INTERRUPT MSR to enable interrupt generation due to dynamic Performance changes and excursions.
 - Availability of HWP Activity window control, CPUID.06H:EAX[bit 9]: If this bit is set, HWP allows software to program activity window in the IA32_HWP_REQUEST MSR.
 - Availability of HWP energy/performance preference control, CPUID.06H:EAX[bit 10]: If this bit is set, HWP allows software to set an energy/performance preference hint in the IA32_HWP_REQUEST MSR.
 - Availability of HWP package level control, CPUID.06H:EAX[bit 11]: If this bit is set, HWP provides the IA32_HWP_REQUEST_PKG MSR to convey OS Power Management's control hints for all logical processors in the physical package.

Table 14-1. Architectural and Non-Architectural MSRs Related to HWP

Address	Architectural	Register Name	Description
770H	Y	IA32_PM_ENABLE	Enable/Disable HWP.
771H	Y	IA32_HWP_CAPABILITIES	Enumerates the HWP performance range (static and dynamic).
772H	Y	IA32_HWP_REQUEST_PKG	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for all logical processor in the physical package.
773H	Y	IA32_HWP_INTERRUPT	Controls HWP native interrupt generation (Guaranteed Performance changes, excursions).
774H	Y	IA32_HWP_REQUEST	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for a single logical processor.
775H	Y	IA32_HWP_PECI_REQUEST_INFO	Conveys embedded system controller requests to override some of the OS HWP Request settings via the PECI mechanism.
777H	Y	IA32_HWP_STATUS	Status bits indicating changes to Guaranteed Performance and excursions to Minimum Performance.
19CH	Y	IA32_THERM_STATUS[bits 15:12]	Conveys reasons for performance excursions.
64EH	N	MSR_PPERF	Productive Performance Count.

- Additionally, HWP may provide a non-architectural MSR, MSR_PPERF, which provides a quantitative metric to software of hardware's view of workload scalability. This hardware's view of workload scalability is implementation specific.

14.4.2 Enabling HWP

The layout of the IA32_PM_ENABLE MSR is shown in Figure 14-4. The bit fields are described below:

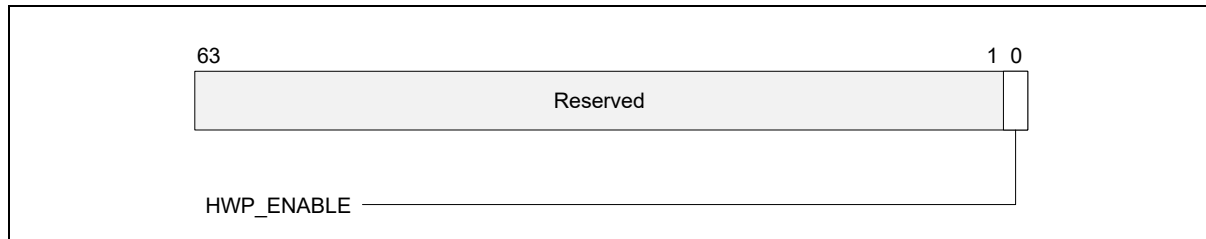


Figure 14-4. IA32_PM_ENABLE MSR

- **HWP_ENABLE (bit 0, R/W1Once)** — Software sets this bit to enable HWP with autonomous selection of processor P-States. When set, the processor will disregard input from the legacy performance control interface (IA32_PERF_CTL). Note this bit can only be enabled once from the default value. Once set, writes to the HWP_ENABLE bit are ignored. Only RESET will clear this bit. Default = zero (0).
- Bits 63:1 are reserved and must be zero.

After software queries CPUID and verifies the processor's support of HWP, system software can write 1 to IA32_PM_ENABLE.HWP_ENABLE (bit 0) to enable hardware controlled performance states. The default value of IA32_PM_ENABLE MSR at power-on is 0, i.e., HWP is disabled.

Additional MSRs associated with HWP may only be accessed after HWP is enabled, with the exception of IA32_HWP_INTERRUPT and MSR_PPERF. Accessing the IA32_HWP_INTERRUPT MSR requires only HWP is present as enumerated by CPUID but does not require enabling HWP.

IA32_PM_ENABLE is a package level MSR, i.e., writing to it from any logical processor within a package affects all logical processors within that package.

14.4.3 HWP Performance Range and Dynamic Capabilities

The OS reads the IA32_HWP_CAPABILITIES MSR to comprehend the limits of the HWP-managed performance range as well as the dynamic capability, which may change during processor operation. The enumerated performance range values reported by IA32_HWP_CAPABILITIES directly map to initial frequency targets (prior to workload-specific frequency optimizations of HWP). However the mapping is processor family specific.

The layout of the IA32_HWP_CAPABILITIES MSR is shown in Figure 14-5. The bit fields are described below:

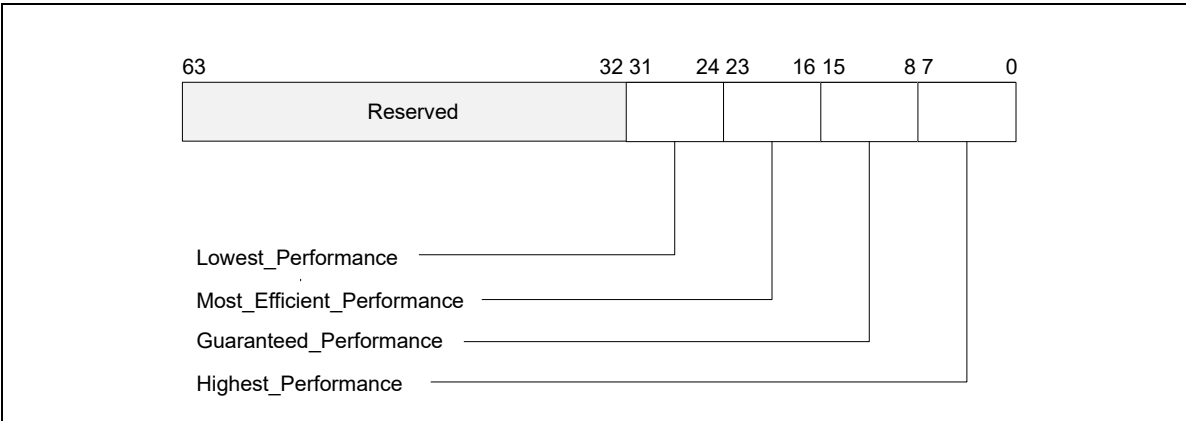


Figure 14-5. IA32_HWP_CAPABILITIES Register

- **Highest_Performance (bits 7:0, RO)** — Value for the maximum non-guaranteed performance level.
- **Guaranteed_Performance (bits 15:8, RO)** — Current value for the guaranteed performance level. This value can change dynamically as a result of internal or external constraints, e.g., thermal or power limits.
- **Most_Efficient_Performance (bits 23:16, RO)** — Current value of the most efficient performance level. This value can change dynamically as a result of workload characteristics.
- **Lowest_Performance (bits 31:24, RO)** — Value for the lowest performance level that software can program to IA32_HWP_REQUEST.
- Bits 63:32 are reserved and must be zero.

The value returned in the **Guaranteed_Performance** field is hardware's best-effort approximation of the available performance given current operating constraints. Changes to the Guaranteed_Performance value will primarily occur due to a shift in operational mode. This includes a power or other limit applied by an external agent, e.g., RAPL (see Figure 14.10.1), or the setting of a Configurable TDP level (see model-specific controls related to Programmable TDP Limit in Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.*). Notification of a change to the Guaranteed_Performance occurs via interrupt (if configured) and the IA32_HWP_Status MSR. Changes to Guaranteed_Performance are indicated when a macroscopically meaningful change in performance occurs i.e., sustained for greater than one second. Consequently, notification of a change in Guaranteed Performance will typically occur no more frequently than once per second. Rapid changes in platform configuration, e.g., docking / undocking, with corresponding changes to a Configurable TDP level could potentially cause more frequent notifications.

The value returned by the **Most_Efficient_Performance** field provides the OS with an indication of the practical lower limit for the IA32_HWP_REQUEST. The processor may not honor IA32_HWP_REQUEST.Maximum Performance settings below this value.

14.4.4 Managing HWP

14.4.4.1 IA32_HWP_REQUEST MSR (Address: 774H Logical Processor Scope)

Typically, the operating system controls HWP operation for each logical processor via the writing of control hints / constraints to the IA32_HWP_REQUEST MSR. The layout of the IA32_HWP_REQUEST MSR is shown in Figure 14-6. The bit fields are described below Figure 14-6.

Operating systems can control HWP by writing both IA32_HWP_REQUEST and IA32_HWP_REQUEST_PKG MSRs (see Section 14.4.4.2). Five valid bits within the IA32_HWP_REQUEST MSR let the operating system flexibly select which of its five hint / constraint fields should be derived by the processor from the IA32_HWP_REQUEST MSR and which should be derived from the IA32_HWP_REQUEST_PKG MSR. These five valid bits are supported if CPUID[6].EAX[17] is set.

When the IA32_HWP_REQUEST MSR Package Control bit is set, any valid bit that is NOT set indicates to the processor to use the respective field value from the IA32_HWP_REQUEST_PKG MSR. Otherwise, the values are derived from the IA32_HWP_REQUEST MSR. The valid bits are ignored when the IA32_HWP_REQUEST MSR Package Control bit is zero.

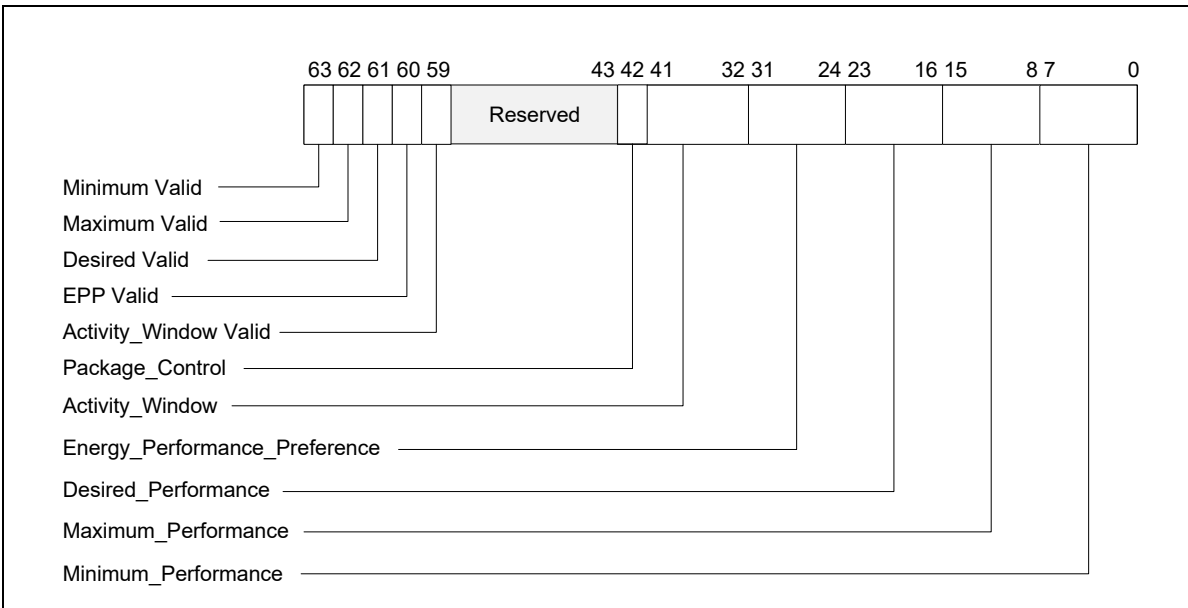


Figure 14-6. IA32_HWP_REQUEST Register

- **Minimum_Performance (bits 7:0, RW)** — Conveys a hint to the HWP hardware. The OS programs the minimum performance hint to achieve the required quality of service (QoS) or to meet a service level agreement (SLA) as needed. Note that an excursion below the level specified is possible due to hardware constraints. The default value of this field is IA32_HWP_CAPABILITIES.Lowest_Performance.
- **Maximum_Performance (bits 15:8, RW)** — Conveys a hint to the HWP hardware. The OS programs this field to limit the maximum performance that is expected to be supplied by the HWP hardware. Excursions above the limit requested by OS are possible due to hardware coordination between the processor cores and other components in the package. The default value of this field is IA32_HWP_CAPABILITIES.Highest_Performance.
- **Desired_Performance (bits 23:16, RW)** — Conveys a hint to the HWP hardware. When set to zero, hardware autonomous selection determines the performance target. When set to a non-zero value (between the range of Lowest_Performance and Highest_Performance of IA32_HWP_CAPABILITIES) conveys an explicit performance request hint to the hardware; effectively disabling HW Autonomous selection. The Desired_Performance input is non-constraining in terms of Performance and Energy Efficiency optimizations, which are independently controlled. The default value of this field is 0.
- **Energy_Performance_Preference (bits 31:24, RW)** — Conveys a hint to the HWP hardware. The OS may write a range of values from 0 (performance preference) to 0FFH (energy efficiency preference) to influence the rate of performance increase /decrease and the result of the hardware's energy efficiency and performance optimizations. The default value of this field is 80H. Note: If CPUID.06H:EAX[bit 10] indicates that this field is not supported, HWP uses the value of the IA32_ENERGY_PERF_BIAS MSR to determine the energy efficiency / performance preference.
- **Activity_Window (bits 41:32, RW)** — Conveys a hint to the HWP hardware specifying a moving workload history observation window for performance/frequency optimizations. If 0, the hardware will determine the appropriate window size. When writing a non-zero value to this field, this field is encoded in the format of bits 38:32 as a 7-bit mantissa and bits 41:39 as a 3-bit exponent value in powers of 10. The resultant value is in microseconds. Thus, the minimal/maximum activity window size is 1 microsecond/1270 seconds. Combined with the Energy_Performance_Preference input, Activity_Window influences the rate of performance increase

/ decrease. This non-zero hint only has meaning when Desired_Performance = 0. The default value of this field is 0.

- **Package_Control (bit 42, RW)** — When set, causes this logical processor's IA32_HWP_REQUEST control inputs to be derived from the IA32_HWP_REQUEST_PKG MSR.
- Bits 58:43 are reserved and must be zero.
- **Activity_Window_Valid (bit 59, RW)** — When set, indicates to the processor to derive the Activity Window field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **EPP_Valid (bit 60, RW)** — When set, indicates to the processor to derive the EPP field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Desired_Valid (bit 61, RW)** — When set, indicates to the processor to derive the Desired Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Maximum_Valid (bit 62, RW)** — When set, indicates to the processor to derive the Maximum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Minimum_Valid (bit 63, RW)** — When set, indicates to the processor to derive the Minimum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.

The HWP hardware clips and resolves the field values as necessary to the valid range. Reads return the last value written not the clipped values.

Processors may support a subset of IA32_HWP_REQUEST fields as indicated by CPUID. Reads of non-supported fields will return 0. Writes to non-supported fields are ignored.

The OS may override HWP's autonomous selection of performance state with a specific performance target by setting the Desired_Performance field to a non-zero value, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations, which are influenced by the Energy Performance Preference field.

Software may disable all hardware optimizations by setting Minimum_Performance = Maximum_Performance (subject to package coordination).

Note: The processor may run below the Minimum_Performance level due to hardware constraints including: power, thermal, and package coordination constraints. The processor may also run below the Minimum_Performance level for short durations (few milliseconds) following C-state exit, and when Hardware Duty Cycling (see Section 14.5) is enabled.

When the IA32_HWP_REQUEST MSR is set to fast access mode, writes of this MSR are posted, i.e., the WRMSR instruction retires before the data reaches its destination within the processor. It may retire even before all preceding IA stores are globally visible, i.e., it is not an architecturally serializing instruction anymore (no store fence). A new CPUID bit indicates this new characteristic of the IA32_HWP_REQUEST MSR (see Section 14.4.8 for additional details).

14.4.4.2 IA32_HWP_REQUEST_PKG MSR (Address: 772H Package Scope)

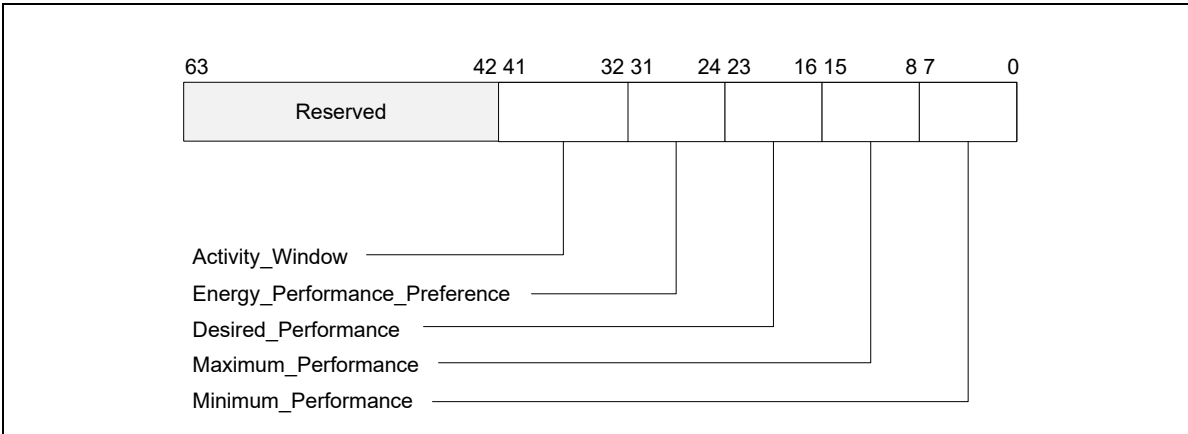


Figure 14-7. IA32_HWP_REQUEST_PKG Register

The structure of the IA32_HWP_REQUEST_PKG MSR (package-level) is identical to the IA32_HWP_REQUEST MSR with the exception of the the Package Control bit field and the five valid bit fields, which do not exist in the IA32_HWP_REQUEST_PKG MSR. Field values written to this MSR apply to all logical processors within the physical package with the exception of logical processors whose IA32_HWP_REQUEST.Package Control field is clear (zero). Single P-state Control mode is only supported when IA32_HWP_REQUEST_PKG is not supported.

14.4.4.3 IA32_HWP_PECI_REQUEST_INFO MSR (Address 775H Package Scope)

When an embedded system controller is integrated in the platform, it can override some of the OS HWP Request settings via the PECI mechanism. PECI initiated settings take precedence over the relevant fields in the IA32_HWP_REQUEST MSR and in the IA32_HWP_REQUEST_PKG MSR, irrespective of the Package Control bit or the Valid Bit values described above. PECI can independently control each of: Minimum Performance, Maximum Performance and EPP fields. This MSR contains both the PECI induced values and the control bits that indicate whether the embedded controller actually set the processor to use the respective value.

PECI override is supported if CPUID[6].EAX[16] is set.

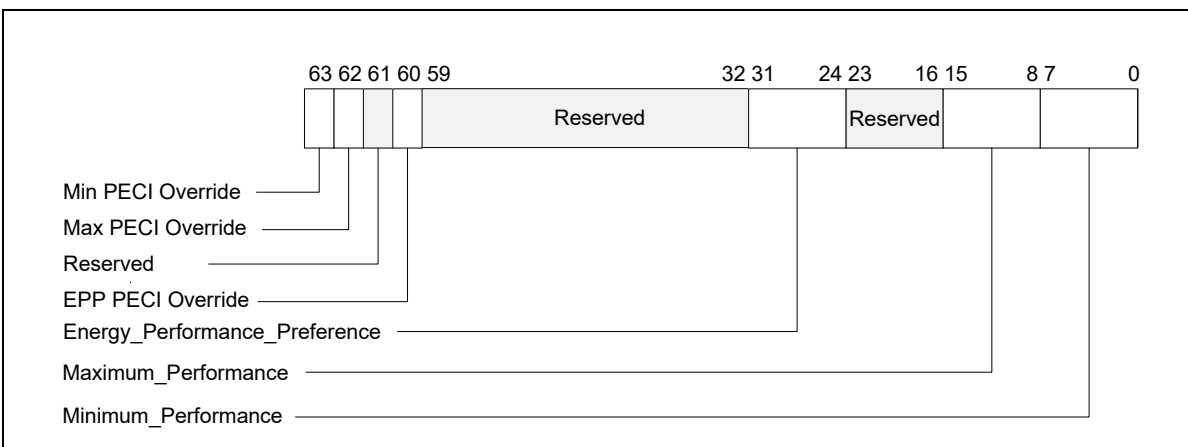


Figure 14-8. IA32_HWP_PECI_REQUEST_INFO MSR

The layout of the IA32_HWP_PECI_REQUEST_INFO MSR is shown in Figure 14-8. This MSR is writable by the embedded controller but is read-only by software executing on the CPU. This MSR has Package scope. The bit fields are described below:

- **Minimum_Performance (bits 7:0, RO)** — Used by the OS to read the latest value of Peci minimum performance input.
- **Maximum_Performance (bits 15:8, RO)** — Used by the OS to read the latest value of Peci maximum performance input.
- Bits 23:16 are reserved and must be zero.
- **Energy_Performance_Preference (bits 31:24, RO)** — Used by the OS to read the latest value of Peci energy performance preference input.
- Bits 59:32 are reserved and must be zero.
- **EPP_Peci_Override (bit 60, RO)** — Indicates whether Peci is currently overriding the Energy Performance Preference input. If set(1), Peci is overriding the Energy Performance Preference input. If clear(0), OS has control over Energy Performance Preference input.
- Bit 61 is reserved and must be zero.
- **Max_Peci_Override (bit 62, RO)** — Indicates whether Peci is currently overriding the Maximum Performance input. If set(1), Peci is overriding the Maximum Performance input. If clear(0), OS has control over Maximum Performance input.
- **Min_Peci_Override (bit 63, RO)** — Indicates whether Peci is currently overriding the Minimum Performance input. If set(1), Peci is overriding the Minimum Performance input. If clear(0), OS has control over Minimum Performance input.

HWP Request Field Hierarchical Resolution

HWP Request field resolution is fed by three MSRs: IA32_HWP_REQUEST, IA32_HWP_REQUEST_PKG and IA32_HWP_PECI_REQUEST_INFO. The flow that the processor goes through to resolve which field value is chosen is shown below.

For each of the two HWP Request fields; Desired and Activity Window:

```
If IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
    Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
Else
    Resolved Field Value = IA32_HWP_REQUEST.<field>
```

For each of the three HWP Request fields; Min, Max and EPP:

```
If IA32_HWP_PECI_REQUEST_INFO.<field> Peci Override bit = 1
    Resolved Field Value = IA32_HWP_PECI_REQUEST_INFO.<field>
Else if IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
    Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
Else
    Resolved Field Value = IA32_HWP_REQUEST.<field>
```

14.4.4.4 IA32_HWP_CTL MSR (Address: 776H Logical Processor Scope)

IA32_HWP_CTL[0] controls the behavior of IA32_HWP_REQUEST Package Control [bit 42]. This control bit allows the IA32_HWP_REQUEST MSR to stay in INIT mode most of the time (Control Bit is equal to its RESET value of 0) thus avoiding actual saving/restoring of the MSR contents when the OS adds it to the register set saved and restored by XSAVES/XRSTORS.

- When IA32_HWP_CTL[0] = 0:
 - If IA32_HWP_REQUEST[42] = 0, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
 - If IA32_HWP_REQUEST[42] = 1, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according

to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 14.4.4.1 for additional details.

- When IA32_HWP_CTL[0] = 1, the behavior is reversed:
 - If IA32_HWP_REQUEST[42] = 1, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
 - If IA32_HWP_REQUEST[42] = 0, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 14.4.4.1 for additional details.

Section 14-2 summarizes the IA32_HWP_CTL MSR bit 0 control behavior.

Table 14-2. IA32_HWP_CTL MSR Bit 0 Behavior

Field	Description		
Thread request PKG CTL meaning	Defines which HWP Request MSR is used, whether thread level or package level. When the package MSR is used, the thread MSR valid bits define which thread MSR fields override the package (default 0).		
	IA32_HWP_CTL[PKG_CTL_PLR]	IA32_HWP_REQUEST[PKG_CTL]	HWP Request MSR Used
	0	0	IA32_HWP_REQUEST MSR
	0	1	IA32_HWP_REQUEST_PKG MSR
	1	0	IA32_HWP_REQUEST_PKG MSR
1	1	IA32_HWP_REQUEST MSR	

This MSR is supported if CPUID[6].EAX[22] is set.

If the IA32_PM_ENABLE[HWP_ENABLE] (bit 0) is not set, access to this MSR will generate a #GP fault.

14.4.5 HWP Feedback

The processor provides several types of feedback to the OS during HWP operation.

The IA32_MPERF MSR and IA32_APERF MSR mechanism (see Section 14.2) allows the OS to calculate the resultant effective frequency delivered over a time period. Energy efficiency and performance optimizations directly impact the resultant effective frequency delivered.

The layout of the IA32_HWP_STATUS MSR is shown in Figure 14-9. It provides feedback regarding changes to IA32_HWP_CAPABILITIES.Guaranteed_Performance, IA32_HWP_CAPABILITIES.Highest_Performance, excursions to IA32_HWP_CAPABILITIES.Minimum_Performance, and PECI_Override entry/exit events. The bit fields are described below:

- **Guaranteed_Performance_Change (bit 0, RWC0)** — If set (1), a change to Guaranteed_Performance has occurred. Software should query IA32_HWP_CAPABILITIES.Guaranteed_Performance value to ascertain the new Guaranteed Performance value and to assess whether to re-adjust HWP hints via IA32_HWP_REQUEST. Software must clear this bit by writing a zero (0).
- Bit 1 is reserved and must be zero.
- **Excursion_To_Minimum (bit 2, RWC0)** — If set (1), an excursion to Minimum_Performance of IA32_HWP_REQUEST has occurred. Software must clear this bit by writing a zero (0).
- **Highest_Change (bit 3, RWC0)** — If set (1), a change to Highest Performance has occurred. Software should query IA32_HWP_CAPABILITIES to ascertain the new Highest Performance value. Software must clear this bit by writing a zero (0). Interrupts upon Highest Performance change are supported if CPUID[6].EAX[15] is set.
- **PECI_Override_Entry (bit 4, RWC0)** — If set (1), an embedded/management controller has started a PECI override of one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are now overridden via the PECI mechanism and what their values are (see Section 14.4.4.3 for

additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override entry are supported if CPUID[6].EAX[16] is set.

- **PECI_Override_Exit (bit 5, RWC0)** — If set (1), an embedded/management controller has stopped overriding one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are still overridden via the PECI mechanism and which fields are now back under software control (see Section 14.4.4.3 for additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override exit are supported if CPUID[6].EAX[16] is set.
- Bits 63:6 are reserved and must be zero.

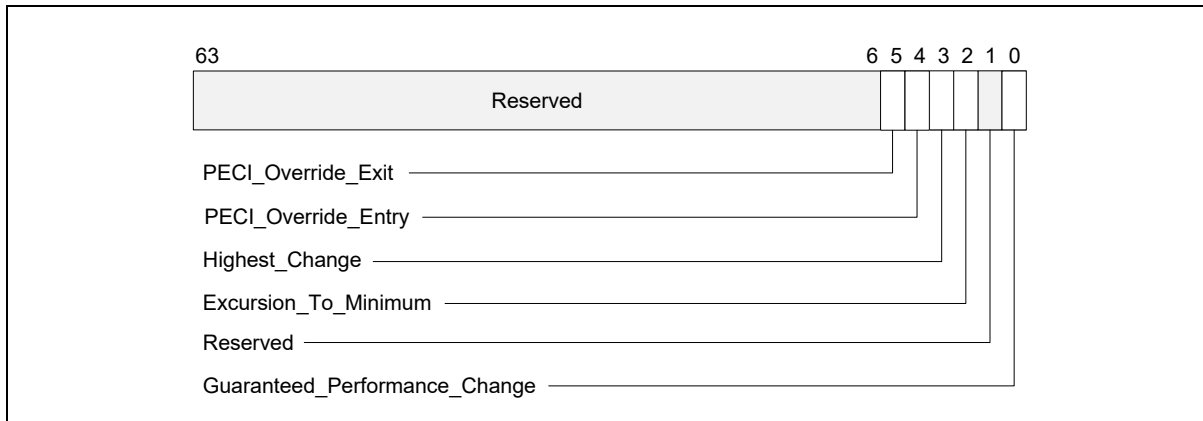


Figure 14-9. IA32_HWP_STATUS MSR

The status bits of IA32_HWP_STATUS must be cleared (0) by software so that a new status condition change will cause the hardware to set the bit again and issue the notification. Status bits are not set for “normal” excursions, e.g., running below Minimum Performance for short durations during C-state exit. Changes to Guaranteed_Performance, Highest_Performance, excursions to Minimum_Performance, or PECI_Override entry/exit will occur no more than once per second.

The OS can determine the specific reasons for a Guaranteed_Performance change or an excursion to Minimum_Performance in IA32_HWP_REQUEST by examining the associated status and log bits reported in the IA32_THERM_STATUS MSR. The layout of the IA32_HWP_STATUS MSR that HWP uses to support software query of HWP feedback is shown in Figure 14-10. The bit fields of IA32_THERM_STATUS associated with HWP feedback are described below (Bit fields of IA32_THERM_STATUS unrelated to HWP can be found in Section 14.8.5.2).

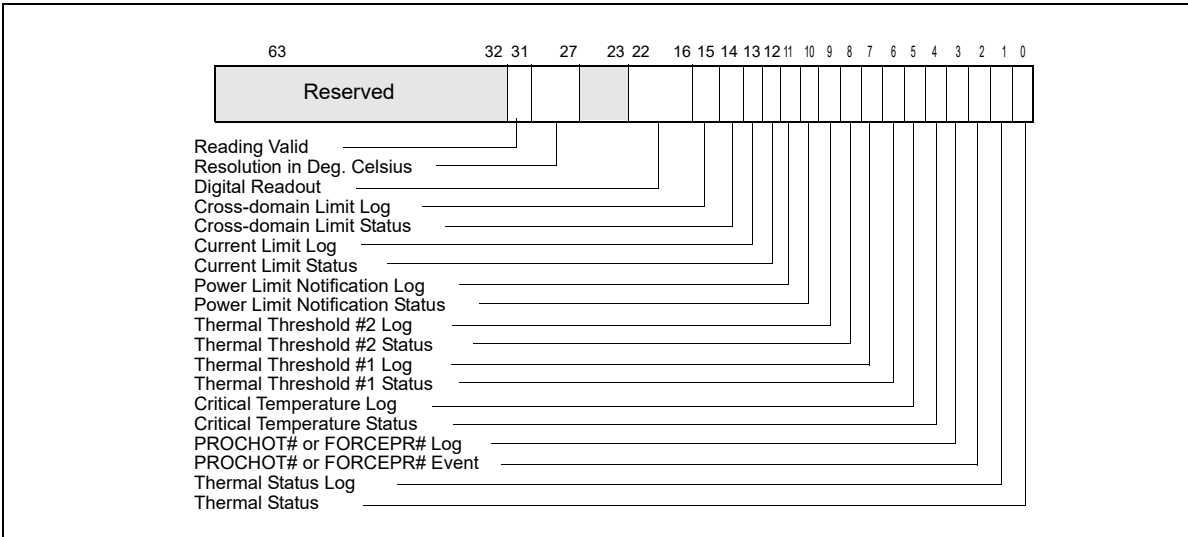


Figure 14-10. IA32_THERM_STATUS Register With HWP Feedback

- Bits 11:0, See Section 14.8.5.2.
- **Current Limit Status (bit 12, RO)** — If set (1), indicates an electrical current limit (e.g., Electrical Design Point/IccMax) is being exceeded and is adversely impacting energy efficiency optimizations.
- **Current Limit Log (bit 13, RWC0)** — If set (1), an electrical current limit has been exceeded that has adversely impacted energy efficiency optimizations since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- **Cross-domain Limit Status (bit 14, RO)** — If set (1), indicates another hardware domain (e.g., processor graphics) is currently limiting energy efficiency optimizations in the processor core domain.
- **Cross-domain Limit Log (bit 15, RWC0)** — If set (1), indicates another hardware domain (e.g., processor graphics) has limited energy efficiency optimizations in the processor core domain since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- Bits 63:16, See Section 14.8.5.2.

14.4.5.1 Non-Architectural HWP Feedback

The Productive Performance (MSR_PPERF) MSR (non-architectural) provides hardware's view of workload scalability, which is a rough assessment of the relationship between frequency and workload performance, to software. The layout of the MSR_PPERF is shown in Figure 14-11.

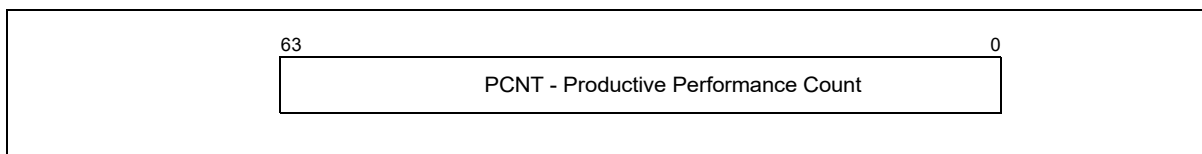


Figure 14-11. MSR_PPERF MSR

- **PCNT (bits 63:0, RO)** — Similar to IA32_APERF but only counts cycles perceived by hardware as contributing to instruction execution (e.g., unhalted and unstalled cycles). This counter increments at the same rate as IA32_APERF, where the ratio of ($\Delta PCNT / \Delta ACNT$) is an indicator of workload scalability (0% to 100%). Note that values in this register are valid even when HWP is not enabled.

14.4.6 HWP Notifications

Processors may support interrupt-based notification of changes to HWP status as indicated by CPUID. If supported, the IA32_HWP_INTERRUPT MSR is used to enable interrupt-based notifications. Notification events, when enabled, are delivered using the existing thermal LVT entry. The layout of the IA32_HWP_INTERRUPT is shown in Figure 14-12. The bit fields are described below:

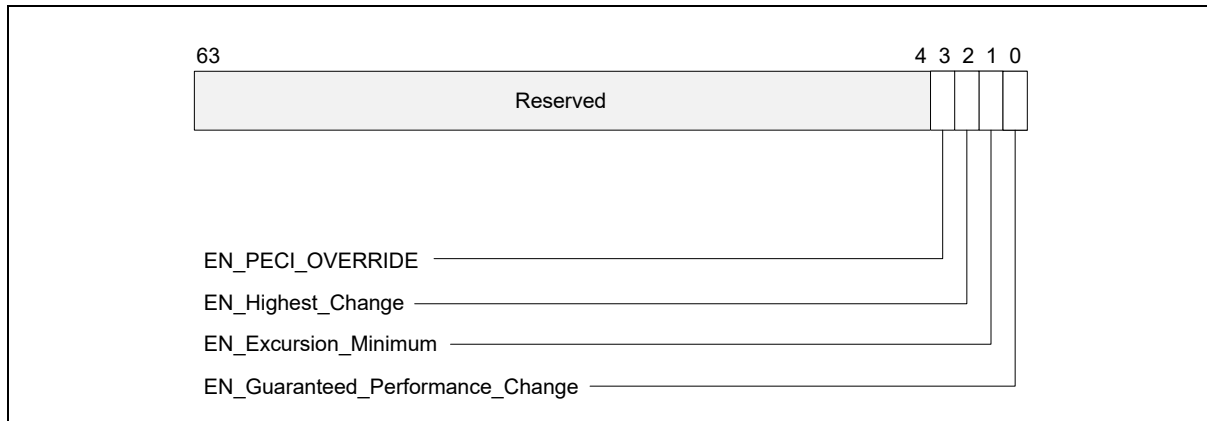


Figure 14-12. IA32_HWP_INTERRUPT MSR

- **EN_Guaranteed_Performance_Change (bit 0, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Guaranteed_Performance occurs. The default value is 0 (Interrupt generation is disabled).
- **EN_Excursion_Minimum (bit 1, RW)** — When set (1), an HWP Interrupt will be generated whenever the HWP hardware is unable to meet the IA32_HWP_REQUEST.Minimum_Performance setting. The default value is 0 (Interrupt generation is disabled).
- **EN_Highest_Change (bit 2, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Highest_Performance occurs. The default value is 0 (interrupt generation is disabled). Interrupts upon Highest Performance change are supported if CPUID[6].EAX[15] is set.
- **EN_PECI_OVERRIDE (bit 3, RW)** — When set (1), an HWP Interrupt will be generated whenever PECI starts or stops overriding any of the three HWP fields described in Section 14.4.4.3. The default value is 0 (interrupt generation is disabled). See Section 14.4.5 and Section 14.4.4.3 for details on how the OS learns what is the current set of HWP fields that are overridden by PECI. Interrupts upon PECI override change are supported if CPUID[6].EAX[16] is set.
- Bits 63:4 are reserved and must be zero.

14.4.7 Idle Logical Processor Impact on Core Frequency

Intel processors use one of two schemes for setting core frequency:

1. All cores share same frequency.
2. Each physical core is set to a frequency of its own.

In both cases the two logical processors that share a single physical core are set to the same frequency, so the processor accounts for the IA32_HWP_REQUEST MSR fields of both logical processors when defining the core frequency or the whole package frequency.

When **CPUID[6].EAX[20]** is set and only one logical processor of the two is active, while the other is idle (in any **C1 sub-state** or in a deeper sleep state), only the **active logical processor's** IA32_HWP_REQUEST MSR fields are considered, i.e., the HWP Request fields of a logical processor in the C1E sub-state or in a deeper sleep state are ignored.

Note: when a logical processor is in **C1 state** its HWP Request fields are accounted for.

14.4.8 Fast Write of Uncore MSR (Model Specific Feature)

There are a few logical processor scope MSRs whose values need to be observed outside the logical processor. The WRMSR instruction takes over 1000 cycles to complete (retire) for those MSRs. This overhead forces operating systems to avoid writing them too often whereas in many cases it is preferable that the OS writes them quite frequently for optimal power/performance operation of the processor.

The model specific “Fast Write MSR” feature reduces this overhead by an order of magnitude to a level of 100 cycles for a selected subset of MSRs.

Note: Writes to Fast Write MSRs are posted, i.e., when the WRMSR instruction completes, the data may still be “in transit” within the processor. Software can check the status by querying the processor to ensure data is already visible outside the logical processor (see Section 14.4.8.3 for additional details). Once the data is visible outside the logical processor, software is ensured that later writes by the same logical processor to the same MSR will be visible later (will not bypass the earlier writes).

MSRs that are selected for Fast Write are specified in a special capability MSR (see Section 14.4.8.1). Architectural MSRs that existed prior to the introduction of this feature and are selected for Fast Write, thus turning from slow to fast write MSRs, will be noted as such via a new CPUID bit. New MSRs that are fast upon introduction will be documented as such without an additional CPUID bit.

Three model specific MSRs are associated with the feature itself. They enable *enumerating, controlling and monitoring* it. All three are logical processor scope.

14.4.8.1 FAST_UNCORE_MSRS_CAPABILITY (Address: 0x65F, Logical Processor Scope)

Operating systems or BIOS can read the FAST_UNCORE_MSRS_CAPABILITY MSR to enumerate those MSRs that are Fast Write MSRs.

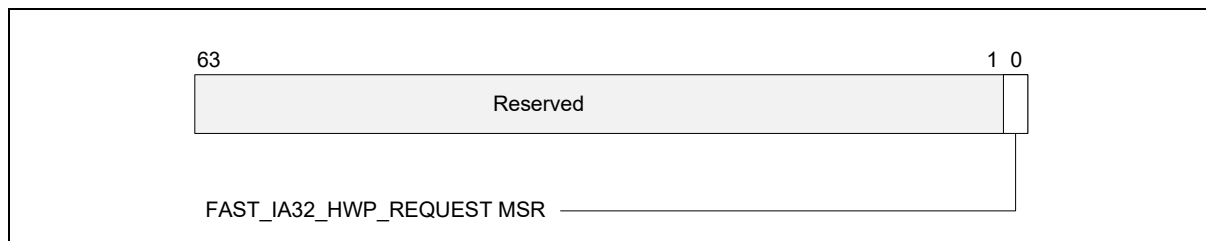


Figure 14-13. FAST_UNCORE_MSRS_CAPABILITY MSR

- **FAST_IA32_HWP_REQUEST MSR (bit 0, RO)** — When set (1), indicates that the IA32_HWP_REQUEST MSR is supported as a Fast Write MSR. A value of 0 indicates the IA32_HWP_REQUEST MSR is not supported as a Fast Write MSR.
- Bits 63:1 are reserved and must be zero.

14.4.8.2 FAST_UNCORE_MSRS_CTL (Address: 0x657, Logical Processor Scope)

Operating Systems or BIOS can use the FAST_UNCORE_MSRS_CTL MSR to opt-in or opt-out for fast write of specific MSRs that are enabled for Fast Write by the processor.

Note: Not all MSRs that are selected for this feature will necessarily have this opt-in/opt-out option. They may be supported in fast write mode only.

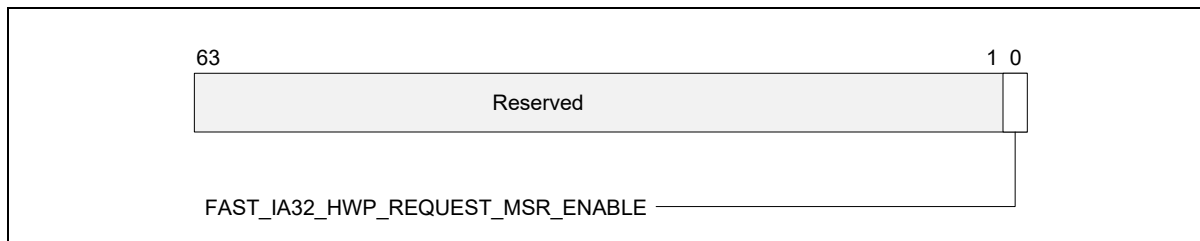


Figure 14-14. FAST_UNCORE_MSRS_CTL MSR

- **FAST_IA32_HWP_REQUEST_MSR_ENABLE (bit 0, RW)** — When set (1), enables fast access mode for the IA32_HWP_REQUEST MSR and sets the low latency, posted IA32_HWP_REQUEST MSR' CPUID[6].EAX[18]. The default value is 0. Note that this bit can only be enabled once from the default value. Once set, writes to this bit are ignored. Only RESET will clear this bit.
- Bits 63:1 are reserved and must be zero.

14.4.8.3 FAST_UNCORE_MSRS_STATUS (Address: 0x65E, Logical Processor Scope)

Software that executes the WRMSR instruction of a Fast Write MSR can check whether the data is already visible outside the logical processor by reading the FAST_UNCORE_MSRS_STATUS MSR. For each Fast Write MSR there is a status bit that indicates whether the data is already visible outside the logical processor or is still in “transit”.

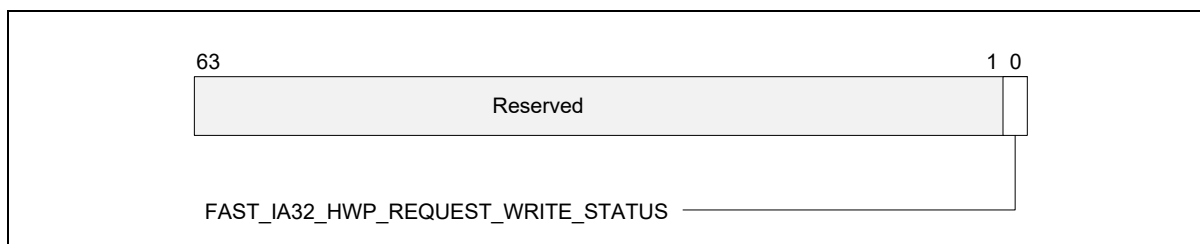


Figure 14-15. FAST_UNCORE_MSRS_STATUS MSR

- **FAST_IA32_HWP_REQUEST_WRITE_STATUS (bit 0, RO)** — Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor.
- Bits 63:1 are reserved and must be zero.

14.4.9 Fast_IA32_HWP_REQUEST CPUID

IA32_HWP_REQUEST is an architectural MSR that exists in processors whose CPUID[6].EAX[7] is set (HWP BASE is enabled). This MSR has logical processor scope, but after its contents are written the contents become visible outside the logical processor. When the FAST_IA32_HWP_REQUEST CPUID[6].EAX[18] bit is set, writes to the IA32_HWP_REQUEST MSR are visible outside the logical processor via the “Fast Write” feature described in Section 14.4.8.

14.4.10 Recommendations for OS use of HWP Controls

Common Cases of Using HWP

The default HWP control field values are expected to be suitable for many applications. The OS can enable autonomous HWP for these common cases by

- Setting IA32_HWP_REQUEST.Desired_Performance = 0 (hardware autonomous selection determines the performance target). Set IA32_HWP_REQUEST.Activity_Window = 0 (enable HW dynamic selection of window size).

To maximize HWP benefit for the common cases, the OS should set

- IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance and
- IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance.

Setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance is functionally equivalent to using of the IA32_PERF_CTL interface and is therefore not recommended (bypassing HWP).

Calibrating HWP for Application-Specific HWP Optimization

In some applications, the OS may have Quality of Service requirements that may not be met by the default values. The OS can characterize HWP by:

- keeping IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance to prevent non-linearity in the characterization process,
- utilizing the range values enumerated from the IA32_HWP_CAPABILITIES MSR to program IA32_HWP_REQUEST while executing workloads of interest and observing the power and performance result.

The power and performance result of characterization is also influenced by the IA32_HWP_REQUEST.Energy_Performance_Preference field, which must also be characterized.

Characterization can be used to set IA32_HWP_REQUEST.Minimum_Performance to achieve the required QOS in terms of performance. If IA32_HWP_REQUEST.Minimum_Performance is set higher than IA32_HWP_CAPABILITIES.Guaranteed_Performance then notification of excursions to Minimum Performance may be continuous.

If autonomous selection does not deliver the required workload performance, the OS should assess the current delivered effective frequency and for the duration of the specific performance requirement set IA32_HWP_REQUEST.Desired_Performance \neq 0 and adjust IA32_HWP_REQUEST.Energy_Performance_Preference as necessary to achieve the required workload performance. The MSR_PPERF.PCNT value can be used to better comprehend the potential performance result from adjustments to IA32_HWP_REQUEST.Desired_Performance. The OS should set IA32_HWP_REQUEST.Desired_Performance = 0 to re-enable autonomous selection.

Tuning for Maximum Performance or Lowest Power Consumption

Maximum performance will be delivered by setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance and setting IA32_HWP_REQUEST.Energy_Performance_Preference = 0 (performance preference).

Lowest power will be achieved by setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance and setting IA32_HWP_REQUEST.Energy_Performance_Preference = 0FFH (energy efficiency preference).

Mixing Logical Processor and Package Level HWP Field Settings

Using the IA32_HWP_REQUEST.Package_Control bit and the five valid bits in that MSR, the OS can mix and match between selecting the Logical Processor scope fields and the Package level fields. For example, the OS can set all logical cores' IA32_HWP_REQUEST.Package_Control bit to '1', and for those logical processors if it prefers a different EPP value than the one set in the IA32_HWP_REQUEST_PKG MSR, the OS can set the desired EPP value and the EPP valid bit. This overrides the package EPP value for only a subset of the logical processors in the package.

Additional Guidelines

Set IA32_HWP_REQUEST.Energy_Performance_Preference as appropriate for the platform's current mode of operation. For example, a mobile platforms' setting may be towards performance preference when on AC power and more towards energy efficiency when on DC power.

The use of the Running Average Power Limit (RAPL) processor capability (see section 14.7.1) is highly recommended when HWP is enabled. Use of IA32_HWP_Request.Maximum_Performance for thermal control is subject to limitations and can adversely impact the performance of other processor components, e.g., graphics

If default values deliver undesirable performance latency in response to events, the OS should set IA32_HWP_REQUEST.Activity_Window to a low (non-zero) value and IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) for the event duration.

Similarly, for “real-time” threads, set IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) and IA32_HWP_REQUEST.Activity_Window to a low value, e.g., 01H, for the duration of their execution.

When executing low priority work that may otherwise cause the hardware to deliver high performance, set IA32_HWP_REQUEST.Activity_Window to a longer value and reduce the IA32_HWP_Request.Maximum_Performance value as appropriate to control energy efficiency. Adjustments to IA32_HWP_REQUEST.Energy_Performance_Preference may also be necessary.

14.5 HARDWARE DUTY CYCLING (HDC)

Intel processors may contain support for Hardware Duty Cycling (HDC), which enables the processor to autonomously force its components inside the physical package into idle state. For example, the processor may selectively force only the processor cores into an idle state.

HDC is disabled by default on processors that support it. System software can dynamically enable or disable HDC to force one or more components into an idle state or wake up those components previously forced into an idle state. Forced Idling (and waking up) of multiple components in a physical package can be done with one WRMSR to a packaged-scope MSR from any logical processor within the same package.

HDC does not delay events such as timer expiration, but it may affect the latency of short (less than 1 msec) software threads, e.g., if a thread is forced to idle state just before completion and entering a “natural idle”.

HDC forced idle operation can be thought of as operating at a lower effective frequency. The effective average frequency computed by software will include the impact of HDC forced idle.

The primary use of HDC is enable system software to manage low active workloads to increase the package level C6 residency. Additionally, HDC can lower the effective average frequency in case of power or thermal limitation.

When HDC forces a logical processor, a processor core or a physical package to enter an idle state, its C-State is set to C3 or deeper. The deep “C-states” referred to in this section are processor-specific C-states.

14.5.1 Hardware Duty Cycling Programming Interfaces

The programming interfaces provided by HDC include the following:

- The CPUID instruction allows software to discover the presence of HDC support in an Intel processor. Specifically, execute CPUID instruction with EAX=06H as input, bit 13 of EAX indicates the processor’s support of the following aspects of HDC.
 - Availability of HDC baseline resource, CPUID.06H:EAX[bit 13]: If this bit is set, HDC provides the following architectural MSRs: IA32_PKG_HDC_CTL, IA32_PM_CTL1, and the IA32_THREAD_STALL MSRs.
- Additionally, HDC may provide several non-architectural MSR.

Table 14-3. Architectural and non-Architecture MSRs Related to HDC

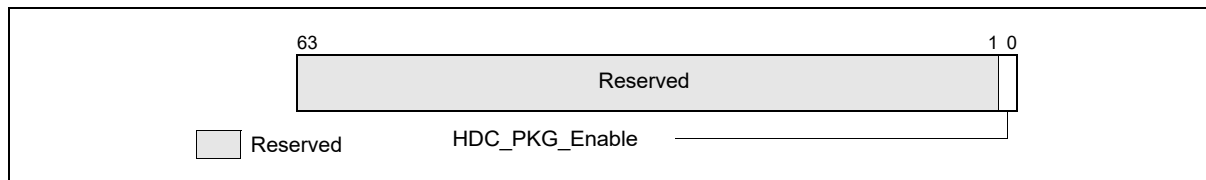
Address	Architectural	Register Name	Description
DB0H	Y	IA32_PKG_HDC_CTL	Package Enable/Disable HDC.
DB1H	Y	IA32_PM_CTL1	Per-logical-processor select control to allow/block HDC forced idling.
DB2H	Y	IA32_THREAD_STALL	Accumulate stalled cycles on this logical processor due to HDC forced idling.
653H	N	MSR_CORE_HDC_RESIDENCY	Core level stalled cycle counter due to HDC forced idling on one or more logical processor.
655H	N	MSR_PKG_HDC_SHALLOW_RESIDENCY	Accumulate the cycles the package was in C2 ¹ state and at least one logical processor was in forced idle
656H	N	MSR_PKG_HDC_DEEP_RESIDENCY	Accumulate the cycles the package was in the software specified Cx ¹ state and at least one logical processor was in forced idle. Cx is specified in MSR_PKG_HDC_CONFIG_CTL.
652H	N	MSR_PKG_HDC_CONFIG_CTL	HDC configuration controls

NOTES:

1. The package “C-states” referred to in this section are processor-specific C-states.

14.5.2 Package level Enabling HDC

The layout of the IA32_PKG_HDC_CTL MSR is shown in Figure 14-16. IA32_PKG_HDC_CTL is a writable MSR from any logical processor in a package. The bit fields are described below:

**Figure 14-16. IA32_PKG_HDC_CTL MSR**

- **HDC_PKG_Enable (bit 0, R/W)** — Software sets this bit to enable HDC operation by allowing the processor to force to idle all “HDC-allowed” (see Figure 14.5.3) logical processors in the package. Clearing this bit disables HDC operation in the package by waking up all the processor cores that were forced into idle by a previous ‘0’-to-‘1’ transition in IA32_PKG_HDC_CTL.HDC_PKG_Enable. This bit is writable only if CPUID.06H:EAX[bit 13] = 1. Default = zero (0).
- Bits 63:1 are reserved and must be zero.

After processor support is determined via CPUID, system software can enable HDC operation by setting IA32_PKG_HDC_CTL.HDC_PKG_Enable to 1. At reset, IA32_PKG_HDC_CTL.HDC_PKG_Enable is cleared to 0. A ‘0’-to-‘1’ transition in HDC_PKG_Enable allows the processor to force to idle all HDC-allowed (indicated by the non-zero state of IA32_PM_CTL1[bit 0]) logical processors in the package. A ‘1’-to-‘0’ transition wakes up those HDC force-idled logical processors.

Software can enable or disable HDC using this package level control multiple times from any logical processor in the package. Note the latency of writing a value to the package-visible IA32_PKG_HDC_CTL.HDC_PKG_Enable is longer than the latency of a WRMSR operation to a Logical Processor MSR (as opposed to package level MSR) such as: IA32_PM_CTL1 (described in Section 14.5.3). Propagation of the change in IA32_PKG_HDC_CTL.HDC_PKG_Enable and reaching all HDC idled logical processor to be woken up may take on the order of core C6 exit latency.

14.5.3 Logical-Processor Level HDC Control

The layout of the IA32_PM_CTL1 MSR is shown in Figure 14-17. Each logical processor in a package has its own IA32_PM_CTL1 MSR. The bit fields are described below:

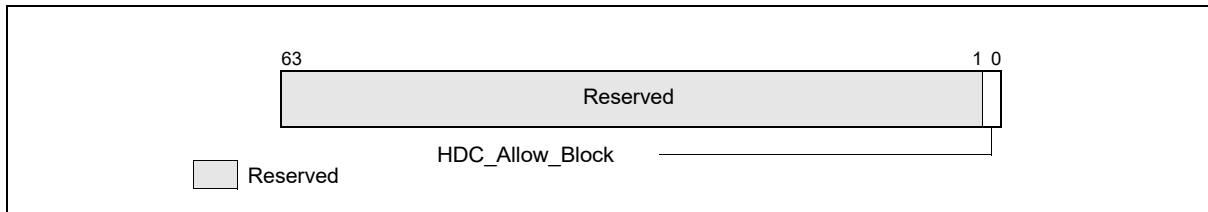


Figure 14-17. IA32_PM_CTL1 MSR

- **HDC_Allow_Block (bit 0, R/W)** — Software sets this bit to allow this logical processors to honor the package-level IA32_PKG_HDC_CTL.HDC_PKG_Enable control. Clearing this bit prevents this logical processor from using the HDC. This bit is writable only if CPUID.06H:EAX[bit 13] = 1. Default = one (1).
- Bits 63:1 are reserved and must be zero.

Fine-grain OS control of HDC operation at the granularity of per-logical-processor is provided by IA32_PM_CTL1. At RESET, all logical processors are allowed to participate in HDC operation such that OS can manage HDC using the package-level IA32_PKG_HDC_CTL.

Writes to IA32_PM_CTL1 complete with the latency that is typical to WRMSR to a Logical Processor level MSR. When the OS chooses to manage HDC operation at per-logical-processor granularity, it can write to IA32_PM_CTL1 on one or more logical processors as desired. Each write to IA32_PM_CTL1 must be done by code that executes on the logical processor targeted to be allowed into or blocked from HDC operation.

Blocking one logical processor for HDC operation may have package level impact. For example, the processor may decide to stop duty cycling of all other Logical Processors as well.

The propagation of IA32_PKG_HDC_CTL.HDC_PKG_Enable in a package takes longer than a WRMSR to IA32_PM_CTL1. The last completed write to IA32_PM_CTL1 on a logical processor will be honored when a '0'-to-'1' transition of IA32_PKG_HDC_CTL.HDC_PKG_Enable arrives to a logical processor.

14.5.4 HDC Residency Counters

There is a collection of counters available for software to track various residency metrics related to HDC operation. In general, HDC residency time is defined as the time in HDC forced idle state at the granularity of per-logical-processor, per-core, or package. At the granularity of per-core/package-level HDC residency, at least one of the logical processor in a core/package must be in the HDC forced idle state.

14.5.4.1 IA32_THREAD_STALL

Software can track per-logical-processor HDC residency using the architectural MSR IA32_THREAD_STALL. The layout of the IA32_THREAD_STALL MSR is shown in Figure 14-18. Each logical processor in a package has its own IA32_THREAD_STALL MSR. The bit fields are described below:

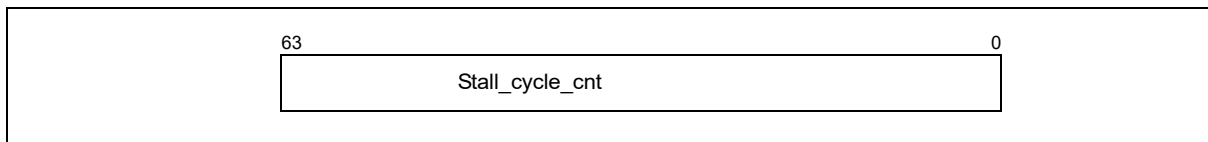


Figure 14-18. IA32_THREAD_STALL MSR

- **Stall_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after the logical processor exits from the forced idled C-state. At each update, the number of cycles that the logical processor was stalled due to forced-idle will be added to the counter. This counter is available only if CPUID.06H:EAX[bit 13] = 1. Default = zero (0).

A value of zero in IA32_THREAD_STALL indicates either HDC is not supported or the logical processor never serviced any forced HDC idle. A non-zero value in IA32_THREAD_STALL indicates the HDC forced-idle residency times of the logical processor. It also indicates the forced-idle cycles due to HDC that could appear as C0 time to traditional OS accounting mechanisms (e.g., time-stamping OS idle/exit events).

Software can read IA32_THREAD_STALL irrespective of the state of IA32_PKG_HDC_CTL and IA32_PM_CTL1, as long as CPUID.06H:EAX[bit 13] = 1.

14.5.4.2 Non-Architectural HDC Residency Counters

Processors that support HDC operation may provide the following model-specific HDC residency counters.

MSR_CORE_HDC_RESIDENCY

Software can track per-core HDC residency using the counter MSR_CORE_HDC_RESIDENCY. This counter increments when the core is in C3 state or deeper (all logical processors in this core are idle due to either HDC or other mechanisms) and at least one of the logical processors is in HDC forced idle state. The layout of the MSR_CORE_HDC_RESIDENCY is shown in Figure 14-19. Each processor core in a package has its own MSR_CORE_HDC_RESIDENCY MSR. The bit fields are described below:

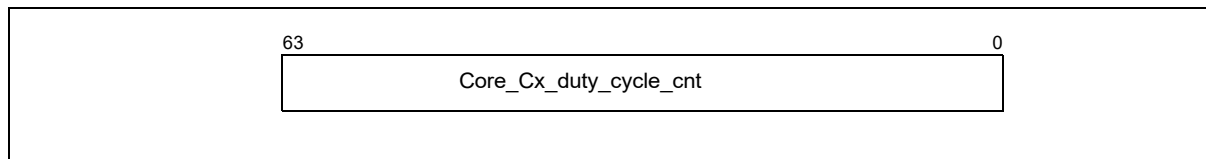


Figure 14-19. MSR_CORE_HDC_RESIDENCY MSR

- **Core_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after core C-state exit from a forced idled C-state. At each update, the increment counts cycles when the core is in a Cx state (all its logical processor are idle) and at least one logical processor in this core was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR will cause a #GP fault. Default = zero (0).

A value of zero in MSR_CORE_HDC_RESIDENCY indicates either HDC is not supported or this processor core never serviced any forced HDC idle.

MSR_PKG_HDC_SHALLOW_RESIDENCY

The counter MSR_PKG_HDC_SHALLOW_RESIDENCY allows software to track HDC residency time when the package is in C2 state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. The layout of the MSR_PKG_HDC_SHALLOW_RESIDENCY is shown in Figure 14-20. There is one MSR_PKG_HDC_SHALLOW_RESIDENCY per package. The bit fields are described below:

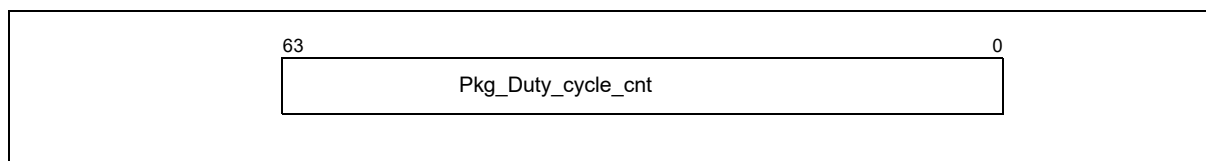


Figure 14-20. MSR_PKG_HDC_SHALLOW_RESIDENCY MSR

- Pkg_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. Package shallow residency may be implementation specific. In the initial implementation, the threshold is package C2-state. The count is updated only after package C2-state exit from a forced idled C-state. At each update, the increment counts cycles when the package is in C2 state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_DEEP_RESIDENCY

The counter MSR_PKG_HDC_DEEP_RESIDENCY allows software to track HDC residency time when the package is in a software-specified package Cx state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. Selection of a specific package Cx state can be configured using MSR_PKG_HDC_CONFIG. The layout of the MSR_PKG_HDC_DEEP_RESIDENCY is shown in Figure 14-21. There is one MSR_PKG_HDC_DEEP_RESIDENCY per package. The bit fields are described below:

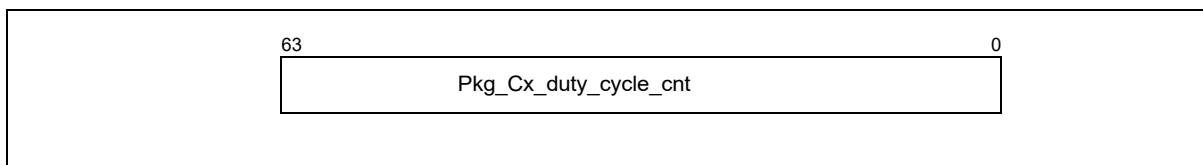


Figure 14-21. MSR_PKG_HDC_DEEP_RESIDENCY MSR

- Pkg_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after package C-state exit from a forced idle state. At each update, the increment counts cycles when the package is in the software-configured Cx state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_CONFIG

MSR_PKG_HDC_CONFIG allows software to configure the package Cx state that the counter MSR_PKG_HDC_DEEP_RESIDENCY monitors. The layout of the MSR_PKG_HDC_CONFIG is shown in Figure 14-22. There is one MSR_PKG_HDC_CONFIG per package. The bit fields are described below:

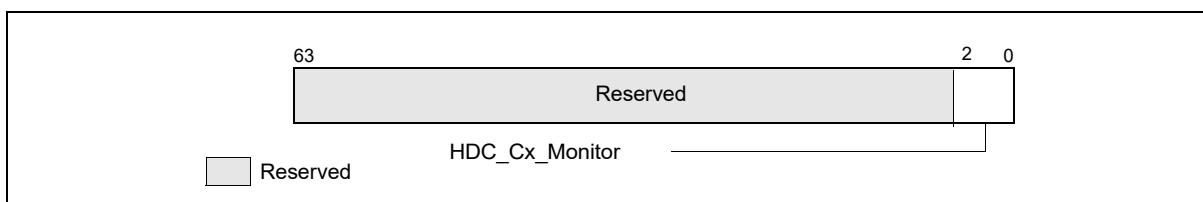


Figure 14-22. MSR_PKG_HDC_CONFIG MSR

- Pkg_Cx_Monitor (bits 2:0, R/W)** — Selects which package C-state the MSR_HDC_DEEP_RESIDENCY counter will monitor. The encoding of the HDC_Cx_Monitor field are: **0**: no-counting; **1**: count package C2 only; **2**: count package C3 and deeper; **3**: count package C6 and deeper; **4**: count package C7 and deeper; other encodings are reserved. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).
- Bits 63:3 are reserved and must be zero.

14.5.5 MPERF and APERF Counters Under HDC

HDC operation can be thought of as an average effective frequency drop due to all or some of the Logical Processors enter an idle state period.

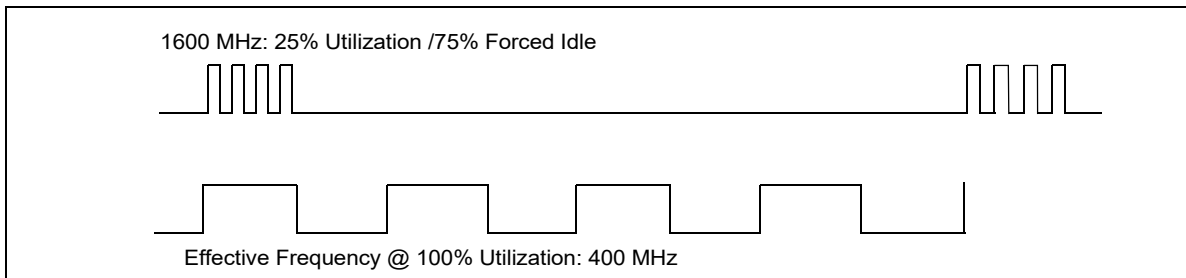


Figure 14-23. Example of Effective Frequency Reduction and Forced Idle Period of HDC

By default, the IA32_MPERF counter counts during forced idle periods as if the logical processor was active. The IA32_APERF counter does not count during forced idle state. This counting convention allows the OS to compute the average effective frequency of the Logical Processor between the last MWAIT exit and the next MWAIT entry (OS visible C0) by $\Delta\text{ACNT}/\Delta\text{MCNT} * \text{TSC Frequency}$.

14.6 HARDWARE FEEDBACK INTERFACE AND INTEL® THREAD DIRECTOR

Intel processors that enumerate CPUID.06H.0H:EAX.HW_FEEDBACK[bit 19] as 1 support Hardware Feedback Interface (HFI). Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a hardware feedback interface structure in memory. Details on this table structure are described in Section 14.6.1.

Intel processors that enumerate CPUID.06H.0H:EAX[bit 23] as 1 support Intel® Thread Director. Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a memory resident table and software thread specific index (Class ID) that points into that table and selects which data to use for that software thread. Details on this table structure are described in Section 14.6.2.

14.6.1 Hardware Feedback Interface Table Structure

This structure has a global header that is 16 bytes in size. Following this global header, there is one 8 byte entry per logical processor in the socket. The structure is designed as follows.

Table 14-4. Hardware Feedback Interface Structure

Byte Offset	Size (Bytes)	Description
0	16	Global Header
16	8	Per Logical Processor Entry
24	8	Per Logical Processor Entry
...
16 + n*8	8	Per Logical Processor Entry

The global header is structured as shown in Table 14-5.

Table 14-5. Hardware Feedback Interface Global Header Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	8	Timestamp	Timestamp of when the table was last updated by hardware. This is a timestamp in crystal clock units. Initialized by the OS to 0.
8	1	Performance Capability Changed	If set to a non-zero value , indicates the performance capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
9	1	Energy Efficiency Capability Changed	If set to a non-zero value , indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
10	6	Reserved	Initialized by the OS to 0.

The per logical processor scheduler feedback entry is structured as follows. The operating system can determine the index of the logical processor feedback entry for a logical processor using CPUID.06H.0H:EDX[31:16] by executing CPUID on that logical processor.

Table 14-6. Hardware Feedback Interface Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.0H:EDX[0] enumerates support for Performance capability reporting.
1	1	Energy Efficiency Capability	Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.0H:EDX[1] enumerates support for Energy Efficiency capability reporting.
2	6	Reserved	The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback.

14.6.2 Intel® Thread Director Table Structure

This structure has a global header that is at least 16 bytes in size. Its size depends on the number of classes and capabilities enumerated by the CPUID instruction (see notes below Table 14-7). Following this global header there are multiple Logical Processor related entries. The structure is designed as follows.

Table 14-7. Intel® Thread Director Table Structure

Byte Offset ^{1,2,3}	Size (Bytes)	Description
0	$8 + CP^4 * CL^4 + R8^5$	Global Header
$8 + CP * CL + R8$	$CL * CP + R8$	Per Logical Processor Entry ₀ ⁶
$8 + 2 * (CP * CL + R8)$	$CL * CP + R8$	Per Logical Processor Entry ₁
...
$8 + (N^7 - 1) * (CP * CL + R8)$	$CL * CP + R8$	Logical Processor Entry _{N-1}

NOTES:

1. Byte offset of Capability_{cp} of Class_{cl} change indication: $8 + CP * cl + cp$.
2. Byte offset of LP Entry_i: $8 + (i+1) * (CP * CL + R8)$.
3. Byte offset of capability_{cp} of class_{cl} of LP Entry_i: $8 + (i+1) * (CP * CL + R8) + CP * cl + cp$.
4. Both upper case CL and CP denote total number of classes and capabilities defined for the processor. Lower case cl and cp denote one instance of a class or capability. cl and cp are counted starting at zero. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for the number of classes (CL) and the number of supported capabilities (CP). CP (# of capabilities): number of enumerated bits in CPUID.06H.0H.EDX[7:0] and CL (# of classes): CPUID.06H.0H.ECX[15:8].
5. R8 is the number of bytes necessary to round up the Capability Change Indication array and the Logical Processor Entry to whole multiple of 8 bytes.
6. Table size: $8 + (N+1) * (CP * CL + R8)$.
7. N is the number of Logical Processor Entries in the table. It is not greater than the number of Logical Processors on the socket, but may be lower.
8. The Operating System can determine the index for the Logical Processor Entry within the Intel Thread Director table using CPUID.06H.0H:EDX[31:16] by executing the CPUID instruction on that Logical Processor.
9. The Operating System should allocate space to accommodate for one such structure per socket in the system.
10. The Intel Thread Director table structure extends the Hardware Feedback Interface table structure without breaking backward compatibility. The Hardware Feedback Interface can be viewed as having two capabilities and a single class.

The global header is structured as shown in Table 14-8.

Table 14-8. Intel® Thread Director Global Header Structure

Byte Offset	Size (Bytes)	Description	
0	8	Time-stamp of when the table was last updated by hardware. This is a time-stamp in crystal clock units. Initialized by the OS to 0.	
8	1	Class 0 Performance Capability Changed	If set to a non-zero value , indicates the performance capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
8 + 1	1	Class 0 Energy Efficiency Capability Changed	If set to a non-zero value , indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
...			
$8 + CP - 1$	1	Class 0 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.

Table 14-8. Intel® Thread Director Global Header Structure (Contd.)

Byte Offset	Size (Bytes)	Description	
8 + CP	1	Class 1 Performance Capability Changed	If set to a non-zero value, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
8 + CP + 1	1	Class 1 Energy Efficiency Capability Changed	If set to a non-zero value, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
...			
8 + 2*CP - 1	1	Class 1 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
...			Change indication for Capabilities of additional Classes if exist.
8 + (CL-1)*CP	1	Class #(CL-1) Performance Capability Changed	If set to a non-zero value, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
8 + (CL-1)*CP + 1	1	Class #(CL-1) Energy Efficiency Capability Changed	If set to a non-zero value, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by the OS to 0.
...			
8 + CL*CP - 1	1	Class #(CL-1) change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
8 + CL*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.

The logical processor capability entry in the Intel Thread Director table is structured as follows.

Table 14-9. Intel® Thread Director Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Class 0 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
1	1	Energy Efficiency Capability	Class 0 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
CP - 1	1	Capability #(CP-1)	Class 0 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.

Table 14-9. Intel® Thread Director Logical Processor Entry Structure (Contd.)

Byte Offset	Size (Bytes)	Field Name	Description
CP + R8	1	Performance Capability	Class 1 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
CP + 1	1	Energy Efficiency Capability	Class 1 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
2*CP - 1	1	Capability #(CP-1)	Class 1 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
2*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.
...			
(CL-1)*CP	1	Performance Capability	Class #(CL-1) Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
(CL-1)*CP + 1	1	Energy Efficiency Capability	Class #(CL-1) Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
CL*CP - 1	1	Capability #(CP-1)	Class #(CL-1) Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CL*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.

14.6.3 Intel® Thread Director Usage Model

When the OS Scheduler needs to decide which one of multiple free logical processors to assign to a software thread that is ready to execute, it can choose one of the following options:

1. The free logical processor with the highest performance value of that software thread class, if the system is scheduling for performance.

- The free logical processor with the highest energy efficiency value of that software thread class, if the system is scheduling for energy efficiency.

When the OS Scheduler needs to decide which of two logical processors (i, j) to assign to which of two software threads whose Class IDs are k_1 and k_2 , it can compute the two performance ratios: $\text{Perf Ratio}_1 = \text{Perf}_{i,k_1} / \text{Perf}_{j,k_1}$ and $\text{Perf Ratio}_2 = \text{Perf}_{i,k_2} / \text{Perf}_{j,k_2}$, or two energy efficiency ratios: $\text{Energy Eff. Ratio}_1 = \text{Energy Eff}_{i,k_1} / \text{Energy Eff}_{j,k_1}$ and $\text{Energy Eff. Ratio}_2 = \text{Energy Eff}_{i,k_2} / \text{Energy Eff}_{j,k_2}$ between the two logical processors for each of the two classes, depending on whether the OS is scheduling for performance or for energy efficiency.

For example, assume that the system is scheduling for performance and that $\text{Perf Ratio}_1 > \text{Perf Ratio}_2$. The OS Scheduler will assign the software thread whose Class ID is k_1 to logical processor i , and the one whose Class ID is k_2 to logical processor j .

When the two software threads in question belong to the same Class ID, the OS Scheduler can schedule to higher performance logical processors within that class when scheduling for performance and to higher energy efficiency logical processors within that class when scheduling for energy efficiency.

The highest to lowest ordering may be different between classes across cores and between the performance column and the energy efficiency column of the same class across cores.

14.6.4 Hardware Feedback Interface Pointer

The physical address of the HFI/Intel Thread Director structure is programmed by the OS into a package scoped MSR named IA32_HW_FEEDBACK_PTR. The MSR is structured as follows:

- Bits 63:MAXPHYADDR¹ – Reserved.
- Bits MAXPHYADDR-1:12 – ADDR. This is the physical address of the page frame of the first page of this structure.
- Bits 11:1 – Reserved.
- Bit 0 – Valid. When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.

The address of this MSR is 17D0H. This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

CPUID.06H.0H:EDX[11:8] enumerates the size of memory that must be allocated by the OS for this structure.

14.6.5 Hardware Feedback Interface Configuration

The operating system enables HFI/Intel Thread Director using a package scoped MSR named IA32_HW_FEEDBACK_CONFIG (address 17D1H). This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

The MSR is structured as follows:

- Bits 63:2 – Reserved.
- Bit 1 – Enable Intel Thread Director (or multi-class support). Both bits 0 and 1 must be set for Intel Thread Director to be enabled. The extra class columns in the Intel Thread Director table are updated by hardware immediately following setting those two bits, as well as during run time as necessary.
- Bit 0 – Enable. When set to 1, enables HFI.

Before enabling HFI, the OS must set a valid hardware feedback interface structure using IA32_HW_FEEDBACK_PTR.

When the OS sets bit 0 only, the hardware populates class 0 capabilities only in the HFI structure. When bit 1 is set after or together with bit 0, the Intel Thread Director multi-class structure is populated.

When either the HFI structure or the Intel Thread Director structure are ready to use by the OS, the hardware sets IA32_PACKAGE_THERM_STATUS[bit 26]. An interrupt is generated by the hardware if IA32_PACKAGE_THERM_INTERRUPT[bit 25] is set.

1. MAXPHYADDR is reported in CPUID.80000008H:EAX[7:0].

When the OS clears bit 1 but leaves bit 0 set, Intel Thread Director is disabled, but HFI is kept operational. IA32_PACKAGE_THERM_STATUS[bit 26] is NOT set in this case.

Clearing bit 0 disables both HFI and Intel Thread Director, independent of the bit 1 state. Setting bit 1 to '1' while keeping bit 0 at '0' is an invalid combination which is quietly ignored.

When the OS clears bit 0, hardware sets the IA32_PACKAGE_THERM_STATUS[bit 26] to 1 to acknowledge disabling of the interface. The OS should wait for this bit to be set to 1 to reclaim the memory of the Intel Thread Director structure, as by setting IA32_PACKAGE_THERM_STATUS[bit 26] hardware guarantees not to write into the Intel Thread Director structure anymore.

The OS may clear bit 0 only after receiving an indication from the hardware that the structure initialization is complete via the same IA32_PACKAGE_THERM_STATUS[bit 26], following enabling of HFI/Intel Thread Director, thus avoiding a race condition between OS and hardware.

Bit 1 is valid only if CPUID[6].EAX[bit 23] is set. When setting this bit while support is not enumerated, the hardware generates #GP.

Table 14-10 summarizes the control options described above.

See Section 14.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

Table 14-10. IA32_HW_FEEDBACK_CONFIG Control Options

Pre-Bit 1	Pre-Bit 0	Post-Bit 1	Post-Bit 0	Action	IA32_PACKAGE_THERM_STATUS [bit 26] and Interrupt
0	0	0	0	Reset value.	Both Hardware Feedback Interface and Intel Thread Director are disabled, no status bit set, no interrupt is generated.
0	0	0	1	Enable HFI structure.	Set the status bit and generate interrupt if enabled.
0	0	1	0	Invalid option; quietly ignored by the hardware.	No action (no update in the table).
0	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	0	0	Disable HFI support.	Set the status bit and generate interrupt if enabled.
0	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	1	1	Enable Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	0	0	0	No action; keeps HFI and Intel Thread Director disabled.	No action (no update in the table).
1	0	0	1	Enable HFI.	Set the status bit and generate interrupt if enabled.
1	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	1	Disable Intel Thread Director; keep HFI enabled.	No action (no update in the table).
1	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.

14.6.6 Hardware Feedback Interface Notifications

The IA32_PACKAGE_THERM_STATUS MSR is extended with a new bit, hardware feedback interface structure change status (bit 26, R/WC0), to indicate that the hardware has updated the HFI/Intel Thread Director structure. This is a sticky bit and once set, indicates that the OS should read the structure to determine the change and adjust its scheduling decisions. Once set, the hardware will not generate any further updates to this structure until the OS clears this bit by writing 0.

The OS can enable interrupt-based notifications when the structure is updated by hardware through a new enable bit, hardware feedback interrupt enable (bit 25, R/W), in the IA32_PACKAGE_THERM_INTERRUPT MSR. When this bit is set to 1, it enables the generation of an interrupt when the HFI/Intel Thread Director structure is updated by hardware. When the enable bit transitions from 0 to 1, hardware will generate an initial notify, with the IA32_PACKAGE_THERM_STATUS bit 26 set to 1, to indicate that the OS should read the current HFI/Intel Thread Director structure.

14.6.7 Hardware Feedback Interface and Intel® Thread Director Structure Dynamic Update

The HFI/Intel Thread Director structure can be updated dynamically during run time. Changes to the structure may occur to one or more of its cells. Such changes may occur for one or more logical processors. The hardware sets a non-zero value in the “capability change” field of the HFI/Intel Thread Director structure as an indication for the OS to read that capability for all logical processors. A thermal interrupt is delivered to indicate to the OS that the structure has just changed. Section 14.6.6 contains more details on this notification mechanism. The hardware clears all “capability change” fields after the OS resets IA32_PACKAGE_THERM_STATUS[bit 26].

Zeroing a performance or energy efficiency cell hints to the OS that it is beneficial not to schedule software threads of that class on the associated logical processor for performance or energy efficiency reasons, respectively. If SMT is supported, it may be the case that the hardware zeroes one of the core's logical processors only. Zeroing the performance and energy efficiency cells of all classes for a logical processor implies that the hardware provides a hint to the OS to completely avoid scheduling work on that logical processor.

When EE=255 is set on one or more logical processors, it represents a request that the OS attempt to consolidate work to those logical processors with EE=255. These requests are made when the SOC has knowledge that consolidating the work to a subset of cores will result in significantly better platform energy efficiency. Examples of consolidating work would include, but not limited to, delaying less important work as needed to provide compute bandwidth for more important work, and routing interrupts to the logical processors with EE=255. When the cumulative workload requires performance greater than that which is available on the subset of cores with EE=255, it is expected that the OS will scale the work out to additional logical processors.

A few example reasons for runtime changes in the HGS/Intel Thread Director Table:

- Over clocking run time update that changes the capability values.
- Change in run time physical constraints.
- Run time performance or energy efficiency optimization.
- Change in core frequency, voltage, or power budget.

14.6.8 Logical Processor Scope Intel® Thread Director Configuration

The operating system enables Intel Thread Director at the logical processor scope using a logical processor scope MSR named IA32_HW_FEEDBACK_THREAD_CONFIG (address 17D4H).

The MSR is read/write and is structured as follows:

- Bits 63:1 – Reserved.
- Bit 0 – Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).

Bit 0 of the logical processor scope configuration MSR can be cleared or set regardless of the state of the HFI/Intel Thread Director package configuration MSR state. Even when bit 0 of all logical processor configuration MSRs is clear, the processor can still update the Intel Thread Director structure if it is still enabled in the IA32_HW_FEEDBACK_CONFIG package scope MSR. When the operating system clears

IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0], hardware clears the history accumulated on that logical processor which otherwise drives assigning the Class ID to the software thread that executes on that logical processor. As long as IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set, the Class ID is available for the operating system to read, independent of the state of the package scope IA32_HW_FEEDBACK_CONFIG[1:0] bits.

See Section 14.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

14.6.9 Implicit Reset of Package and Logical Processor Scope Configuration MSRs

HFI/Intel Thread Director enable bits are reset by hardware in the following scenarios:

1. When GETSEC[SENDER] is executed:
 - a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on all sockets (packages) in the system.
 - b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors in the system across all sockets.
 - c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR across all sockets.
2. When GETSEC[ENTERACCS] is executed:
 - a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.
 - b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors on the socket where the GETSEC[ENTERACCS] instruction was executed.
 - c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.
3. When an INIT or a wait-for-SIPI state are processed by a logical processor:
 - a. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on that logical processor, whether the signal was in the context of GETSEC[ENTERACCS] or not.

If the OS requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

14.6.10 Logical Processor Scope Intel® Thread Director Run Time Characteristics

The processor provides the operating system with run time feedback about the execution characteristics of the software thread executing on logical processors whose IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set.

The run time feedback is communicated via a read-only MSR named IA32_THREAD_FEEDBACK_CHAR. This is a logical processor scope MSR whose address is 17D2H. This MSR is structured as follows:

- Bit 63 – Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.
- Bits 62:8 – Reserved.
- Bits 7:0 – Application Class ID, pointing into the Intel Thread Director structure described in Table 14-8.

This MSR is valid only if CPUID.06H:EAX[bit 23] is set.

The valid bit is cleared by the hardware in the following cases:

- The hardware does not have enough information to provide the operating system with a reliable Class ID.

- The operating system cleared the logical processor's IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] bit. The HRESET instruction is executed while configured to reset the Intel Thread Director history.

14.6.11 Logical Processor Scope History

The operating system can reset the Intel Thread Director related history accumulated on the current logical processor it is executing on by issuing the HRESET instruction. See "CPUID—CPU Identification" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A* for enumeration of the HRESET instruction. See also the "HRESET — History Reset" instruction description in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

14.6.11.1 Enabling Intel® Thread Director History Reset

The IA32_HRESET_ENABLE MSR is a read/write MSR and is structured as follows:

- Bits 63:32 – Reserved.
- Bits 31:1 – Reserved for other capabilities that can be reset by the HRESET instruction.
- Bit 0 – Enable reset of the Intel Thread Director history.

The operating system should set IA32_HRESET_ENABLE[bit 0] to enable Intel Thread Director history reset via the HRESET instruction.

14.6.11.2 Implicit Intel® Thread Director History Reset

The Intel Thread Director history is implicitly reset in the following scenarios:

1. When the processor enters or exits SMM mode and IA32_DEBUGCTL MSR.FREEZE_WHILE_SMM (bit 14) is set, the Intel Thread Director history is implicitly reset by the processor.
2. When GETSEC[SENDER] is executed, the processor resets the Intel Thread Director history on all logical processors in the system, including logical processors on other sockets (other than the one GETSEC[SENDER] is executed).
3. When GETSEC[ENTERACCS] is executed, the processor resets the Intel Thread Director history on the logical processor it is executed on.
4. When an INIT or a wait-for-SIPI state are processed by a logical processor, the Intel Thread Director history is reset whether the signal was a result of GETSEC[ENTERACCS] or not.

If the operating system requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

14.7 MWAIT EXTENSIONS FOR ADVANCED POWER MANAGEMENT

IA-32 processors may support a number of C-states¹ that reduce power consumption for inactive states. Intel Core Solo and Intel Core Duo processors support both deeper C-state and MWAIT extensions that can be used by OS to implement power management policy.

Software should use CPUID to discover if a target processor supports the enumeration of MWAIT extensions. If CPUID.05H.ECX[Bit 0] = 1, the target processor supports MWAIT extensions and their enumeration (see Chapter 4, "Instruction Set Reference, M-U," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*).

If CPUID.05H.ECX[Bit 1] = 1, the target processor supports using interrupts as break-events for MWAIT, even when interrupts are disabled. Use this feature to measure C-state residency as follows:

1. The processor-specific C-states defined in MWAIT extensions can map to ACPI defined C-state types (C0, C1, C2, C3). The mapping relationship depends on the definition of a C-state by processor implementation and is exposed to OSPM by the BIOS using the ACPI defined _CST table.

- Software can write to bit 0 in the MWAIT Extensions register (ECX) when issuing an MWAIT to enter into a processor-specific C-state or sub C-state.
- When a processor comes out of an inactive C-state or sub C-state, software can read a timestamp before an interrupt service routine (ISR) is potentially executed.

CPUID.05H.EDX allows software to enumerate processor-specific C-states and sub C-states available for use with MWAIT extensions. IA-32 processors may support more than one C-state of a given C-state type. These are called sub C-states. Numerically higher C-state have higher power savings and latency (upon entering and exiting) than lower-numbered C-state.

At CPL = 0, system software can specify desired C-state and sub C-state by using the MWAIT hints register (EAX). Processors will not go to C-state and sub C-state deeper than what is specified by the hint register. If CPL > 0 and if MONITOR/MWAIT is supported at CPL > 0, the processor will only enter C1-state (regardless of the C-state request in the hints register).

Executing MWAIT generates an exception on processors operating at a privilege level where MONITOR/MWAIT are not supported.

NOTE

If MWAIT is used to enter a C-state (including sub C-state) that is numerically higher than C1, a store to the address range armed by MONITOR instruction will cause the processor to exit MWAIT if the store was originated by other processor agents. A store from non-processor agent may not cause the processor to exit MWAIT.

14.8 THERMAL MONITORING AND PROTECTION

The IA-32 architecture provides the following mechanisms for monitoring temperature and controlling thermal power:

1. The **catastrophic shutdown detector** forces processor execution to stop if the processor's core temperature rises above a preset limit.
2. **Automatic and adaptive thermal monitoring mechanisms** force the processor to reduce its power consumption in order to operate within predetermined temperature limits.
3. The **software controlled clock modulation mechanism** permits operating systems to implement power management policies that reduce power consumption; this is in addition to the reduction offered by automatic thermal monitoring mechanisms.
4. **On-die digital thermal sensor and interrupt mechanisms** permit the OS to manage thermal conditions natively without relying on BIOS or other system board components.

The first mechanism is not visible to software. The other three mechanisms are visible to software using processor feature information returned by executing CPUID with EAX = 1.

The second mechanism includes:

- **Automatic thermal monitoring** provides two modes of operation. One mode modulates the clock duty cycle; the second mode changes the processor's frequency. Both modes are used to control the core temperature of the processor.
- **Adaptive thermal monitoring** can provide flexible thermal management on processors made of multiple cores.

The third mechanism modulates the clock duty cycle of the processor. As shown in Figure 14-24, the phrase 'duty cycle' does not refer to the actual duty cycle of the clock signal. Instead it refers to the time period during which the clock signal is allowed to drive the processor chip. By using the stop clock mechanism to control how often the processor is clocked, processor power consumption can be modulated.

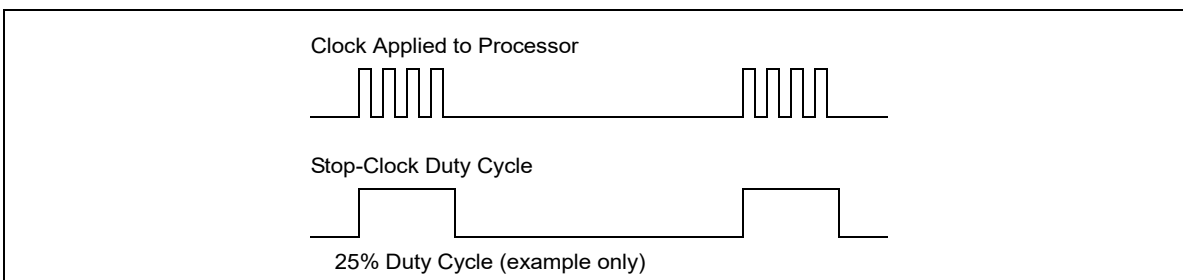


Figure 14-24. Processor Modulation Through Stop-Clock Mechanism

For previous automatic thermal monitoring mechanisms, software controlled mechanisms that changed processor operating parameters to impact changes in thermal conditions. Software did not have native access to the native thermal condition of the processor; nor could software alter the trigger condition that initiated software program control.

The fourth mechanism (listed above) provides access to an on-die digital thermal sensor using a model-specific register and uses an interrupt mechanism to alert software to initiate digital thermal monitoring.

14.8.1 Catastrophic Shutdown Detector

P6 family processors introduced a thermal sensor that acts as a catastrophic shutdown detector. This catastrophic shutdown detector was also implemented in Pentium 4, Intel Xeon and Pentium M processors. It is always enabled. When processor core temperature reaches a factory preset level, the sensor trips and processor execution is halted until after the next reset cycle.

14.8.2 Thermal Monitor

Pentium 4, Intel Xeon and Pentium M processors introduced a second temperature sensor that is factory-calibrated to trip when the processor's core temperature crosses a level corresponding to the recommended thermal design envelop. The trip-temperature of the second sensor is calibrated below the temperature assigned to the catastrophic shutdown detector.

14.8.2.1 Thermal Monitor 1

The Pentium 4 processor uses the second temperature sensor in conjunction with a mechanism called Thermal Monitor 1 (TM1) to control the core temperature of the processor. TM1 controls the processor's temperature by modulating the duty cycle of the processor clock. Modulation of duty cycles is processor model specific. Note that the processors STPCLK# pin is not used here; the stop-clock circuitry is controlled internally.

Support for TM1 is indicated by `CPUID.1:EDX.TM[bit 29] = 1`.

TM1 is enabled by setting the thermal-monitor enable flag (bit 3) in `IA32_MISC_ENABLE` [see Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*]. Following a power-up or reset, the flag is cleared, disabling TM1. BIOS is required to enable only one automatic thermal monitoring modes. Operating systems and applications must not disable the operation of these mechanisms.

14.8.2.2 Thermal Monitor 2

An additional automatic thermal protection mechanism, called Thermal Monitor 2 (TM2), was introduced in the Intel Pentium M processor and also incorporated in newer models of the Pentium 4 processor family. Intel Core Duo and Solo processors, and Intel Core 2 Duo processor family all support TM1 and TM2. TM2 controls the core temperature of the processor by reducing the operating frequency and voltage of the processor and offers a higher performance level for a given level of power reduction than TM1.

TM2 is triggered by the same temperature sensor as TM1. The mechanism to enable TM2 may be implemented differently across various IA-32 processor families with different CPUID signatures in the family encoding value, but will be uniform within an IA-32 processor family.

Support for TM2 is indicated by CPUID.1:ECX.TM2[bit 8] = 1.

14.8.2.3 Two Methods for Enabling TM2

On processors with CPUID family/model/stepping signature encoded as 0x69n or 0x6Dn (early Pentium M processors), TM2 is enabled if the TM_SELECT flag (bit 16) of the MSR_THERM2_CTL register is set to 1 (Figure 14-25) and bit 3 of the IA32_MISC_ENABLE register is set to 1.

Following a power-up or reset, the TM_SELECT flag may be cleared. BIOS is required to enable either TM1 or TM2. Operating systems and applications must not disable mechanisms that enable TM1 or TM2. If bit 3 of the IA32_MISC_ENABLE register is set and TM_SELECT flag of the MSR_THERM2_CTL register is cleared, TM1 is enabled.

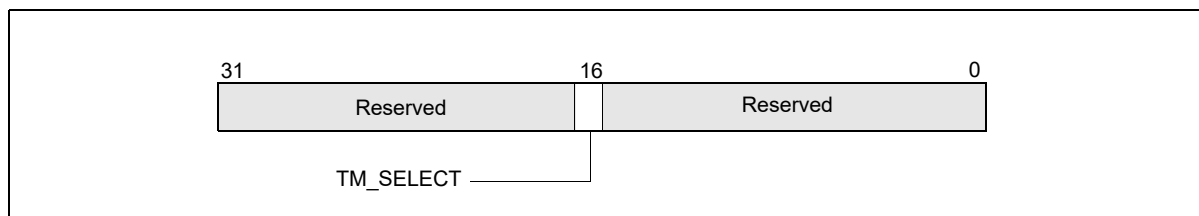


Figure 14-25. MSR_THERM2_CTL Register On Processors with CPUID Family/Model/Stepping Signature Encoded as 0x69n or 0x6Dn

On processors introduced after the Pentium 4 processor (this includes most Pentium M processors), the method used to enable TM2 is different. TM2 is enabled by setting bit 13 of IA32_MISC_ENABLE register to 1. This applies to Intel Core Duo, Core Solo, and Intel Core 2 processor family.

The target operating frequency and voltage for the TM2 transition after TM2 is triggered is specified by the value written to MSR_THERM2_CTL, bits 15:0 (Figure 14-26). Following a power-up or reset, BIOS is required to enable at least one of these two thermal monitoring mechanisms. If both TM1 and TM2 are supported, BIOS may choose to enable TM2 instead of TM1. Operating systems and applications must not disable the mechanisms that enable TM1 or TM2; and they must not alter the value in bits 15:0 of the MSR_THERM2_CTL register.

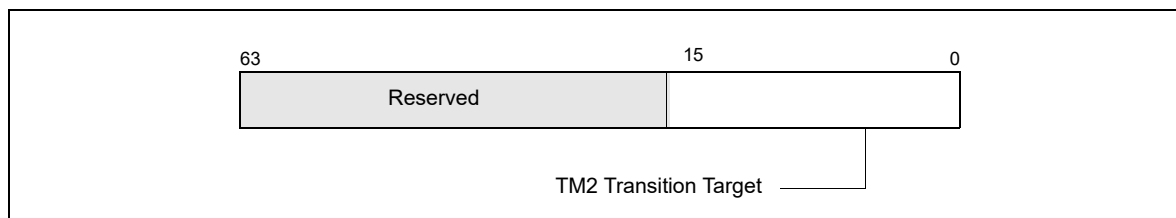


Figure 14-26. MSR_THERM2_CTL Register for Supporting TM2

14.8.2.4 Performance State Transitions and Thermal Monitoring

If the thermal control circuitry (TCC) for thermal monitor (TM1/TM2) is active, writes to the IA32_PERF_CTL will effect a new target operating point as follows:

- If TM1 is enabled and the TCC is engaged, the performance state transition can commence before the TCC is disengaged.

- If TM2 is enabled and the TCC is engaged, the performance state transition specified by a write to the IA32_PERF_CTL will commence after the TCC has disengaged.

14.8.2.5 Thermal Status Information

The status of the temperature sensor that triggers the thermal monitor (TM1/TM2) is indicated through the thermal status flag and thermal status log flag in the IA32_THERM_STATUS MSR (see Figure 14-27).

The functions of these flags are:

- **Thermal Status flag, bit 0** — When set, indicates that the processor core temperature is currently at the trip temperature of the thermal monitor and that the processor power consumption is being reduced via either TM1 or TM2, depending on which is enabled. When clear, the flag indicates that the core temperature is below the thermal monitor trip temperature. This flag is read only.
- **Thermal Status Log flag, bit 1** — When set, indicates that the thermal sensor has tripped since the last power-up or reset or since the last time that software cleared this flag. This flag is a sticky bit; once set it remains set until cleared by software or until a power-up or reset of the processor. The default state is clear.

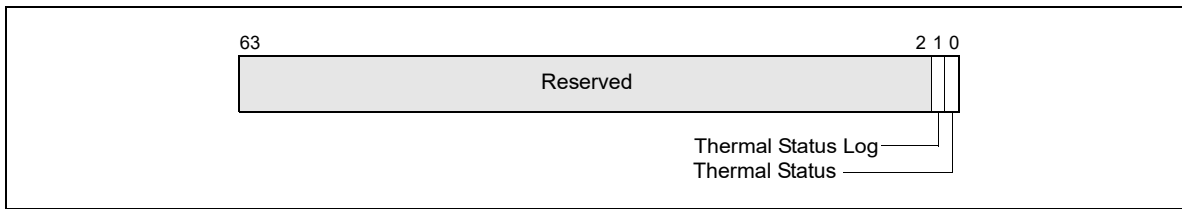


Figure 14-27. IA32_THERM_STATUS MSR

After the second temperature sensor has been tripped, the thermal monitor (TM1/TM2) will remain engaged for a minimum time period (on the order of 1 ms). The thermal monitor will remain engaged until the processor core temperature drops below the preset trip temperature of the temperature sensor, taking hysteresis into account.

While the processor is in a stop-clock state, interrupts will be blocked from interrupting the processor. This holding off of interrupts increases the interrupt latency, but does not cause interrupts to be lost. Outstanding interrupts remain pending until clock modulation is complete.

The thermal monitor can be programmed to generate an interrupt to the processor when the thermal sensor is tripped; this is called a thermal interrupt. The delivery mode, mask and vector for this interrupt can be programmed through the thermal entry in the local APIC’s LVT (see Section 10.5.1, “Local Vector Table”). The low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR (see Figure 14-28) control when the interrupt is generated; that is, on a transition from a temperature below the trip point to above and/or vice-versa.

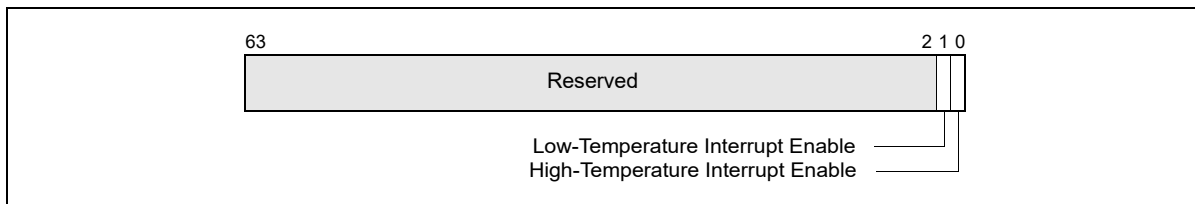


Figure 14-28. IA32_THERM_INTERRUPT MSR

- **High-Temperature Interrupt Enable flag, bit 0** — Enables an interrupt to be generated on the transition from a low-temperature to a high-temperature when set; disables the interrupt when clear.(R/W).
- **Low-Temperature Interrupt Enable flag, bit 1** — Enables an interrupt to be generated on the transition from a high-temperature to a low-temperature when set; disables the interrupt when clear.

The thermal interrupt can be masked by the thermal LVT entry. After a power-up or reset, the low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR are cleared

(interrupts are disabled) and the thermal LVT entry is set to mask interrupts. This interrupt should be handled either by the operating system or system management mode (SMM) code.

Note that the operation of the thermal monitoring mechanism has no effect upon the clock rate of the processor's internal high-resolution timer (time stamp counter).

14.8.2.6 Adaptive Thermal Monitor

The Intel Core 2 Duo processor family supports enhanced thermal management mechanism, referred to as Adaptive Thermal Monitor (Adaptive TM).

Unlike TM2, Adaptive TM is not limited to one TM2 transition target. During a thermal trip event, Adaptive TM (if enabled) selects an optimal target operating point based on whether or not the current operating point has effectively cooled the processor.

Similar to TM2, Adaptive TM is enable by BIOS. The BIOS is required to test the TM1 and TM2 feature flags and enable all available thermal control mechanisms (including Adaptive TM) at platform initiation.

Adaptive TM is available only to a subset of processors that support TM2.

In each chip-multiprocessing (CMP) silicon die, each core has a unique thermal sensor that triggers independently. These thermal sensor can trigger TM1 or TM2 transitions in the same manner as described in Section 14.8.2.1 and Section 14.8.2.2. The trip point of the thermal sensor is not programmable by software since it is set during the fabrication of the processor.

Each thermal sensor in a processor core may be triggered independently to engage thermal management features. In Adaptive TM, both cores will transition to a lower frequency and/or lower voltage level if one sensor is triggered.

Triggering of this sensor is visible to software via the thermal interrupt LVT entry in the local APIC of a given core.

14.8.3 Software Controlled Clock Modulation

Pentium 4, Intel Xeon and Pentium M processors also support software-controlled clock modulation. This provides a means for operating systems to implement a power management policy to reduce the power consumption of the processor. Here, the stop-clock duty cycle is controlled by software through the IA32_CLOCK_MODULATION MSR (see Figure 14-29).

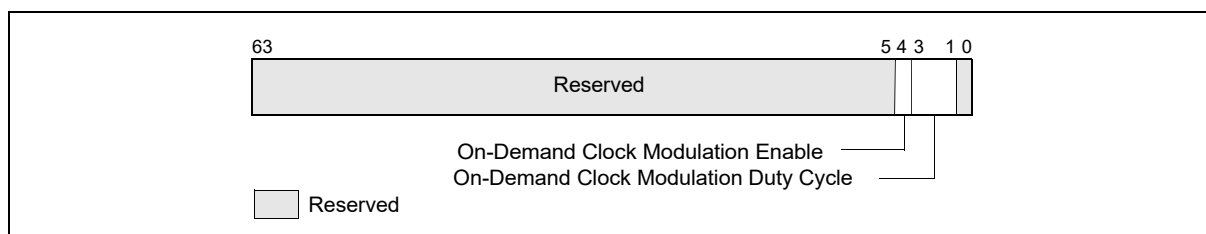


Figure 14-29. IA32_CLOCK_MODULATION MSR

The IA32_CLOCK_MODULATION MSR contains the following flag and field used to enable software-controlled clock modulation and to select the clock modulation duty cycle:

- **On-Demand Clock Modulation Enable, bit 4** — Enables on-demand software controlled clock modulation when set; disables software-controlled clock modulation when clear.
- **On-Demand Clock Modulation Duty Cycle, bits 1 through 3** — Selects the on-demand clock modulation duty cycle (see Table 14-11). This field is only active when the on-demand clock modulation enable flag is set.

Note that the on-demand clock modulation mechanism (like the thermal monitor) controls the processor's stop-clock circuitry internally to modulate the clock signal. The STPCLK# pin is not used in this mechanism.

Table 14-11. On-Demand Clock Modulation Duty Cycle Field Encoding

Duty Cycle Field Encoding	Duty Cycle
---------------------------	------------

Table 14-11. On-Demand Clock Modulation Duty Cycle Field Encoding

000B	Reserved
001B	12.5% (Default)
010B	25.0%
011B	37.5%
100B	50.0%
101B	63.5%
110B	75%
111B	87.5%

The on-demand clock modulation mechanism can be used to control processor power consumption. Power management software can write to the IA32_CLOCK_MODULATION MSR to enable clock modulation and to select a modulation duty cycle. If on-demand clock modulation and TM1 are both enabled and the thermal status of the processor is hot (bit 0 of the IA32_THERM_STATUS MSR is set), clock modulation at the duty cycle specified by TM1 takes precedence, regardless of the setting of the on-demand clock modulation duty cycle.

For Hyper-Threading Technology enabled processors, the IA32_CLOCK_MODULATION register is duplicated for each logical processor. In order for the On-demand clock modulation feature to work properly, the feature must be enabled on all the logical processors within a physical processor. If the programmed duty cycle is not identical for all the logical processors, the processor core clock will modulate to the highest duty cycle programmed for processors with any of the following CPUID DisplayFamily_DisplayModel signatures (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*): 06_1A, 06_1C, 06_1E, 06_1F, 06_25, 06_26, 06_27, 06_2C, 06_2E, 06_2F, 06_35, 06_36, and 0F_xx. For all other processors, if the programmed duty cycle is not identical for all logical processors in the same core, the processor core will modulate at the lowest programmed duty cycle.

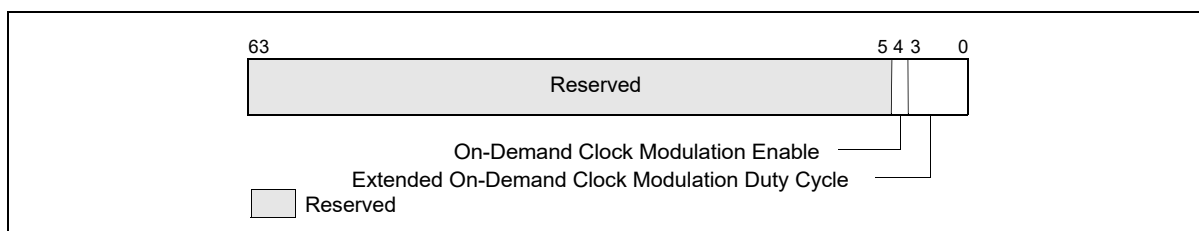
For multiple processor cores in a physical package, each processor core can modulate to a programmed duty cycle independently.

For the P6 family processors, on-demand clock modulation was implemented through the chipset, which controlled clock modulation through the processor's STPCLK# pin.

14.8.3.1 Extension of Software Controlled Clock Modulation

Extension of the software controlled clock modulation facility supports on-demand clock modulation duty cycle with 4-bit dynamic range (increased from 3-bit range). Granularity of clock modulation duty cycle is increased to 6.25% (compared to 12.5%).

Four bit dynamic range control is provided by using bit 0 in conjunction with bits 3:1 of the IA32_CLOCK_MODULATION MSR (see Figure 14-30).

**Figure 14-30. IA32_CLOCK_MODULATION MSR with Clock Modulation Extension**

Extension to software controlled clock modulation is supported only if CPUID.06H:EAX[Bit 5] = 1. If CPUID.06H:EAX[Bit 5] = 0, then bit 0 of IA32_CLOCK_MODULATION is reserved.

14.8.4 Detection of Thermal Monitor and Software Controlled Clock Modulation Facilities

The ACPI flag (bit 22) of the CPUID feature flags indicates the presence of the IA32_THERM_STATUS, IA32_THERM_INTERRUPT, IA32_CLOCK_MODULATION MSRs, and the xAPIC thermal LVT entry.

The TM1 flag (bit 29) of the CPUID feature flags indicates the presence of the automatic thermal monitoring facilities that modulate clock duty cycles.

14.8.4.1 Detection of Software Controlled Clock Modulation Extension

Processor's support of software controlled clock modulation extension is indicated by CPUID.06H:EAX[Bit 5] = 1.

14.8.5 On Die Digital Thermal Sensors

On die digital thermal sensor can be read using an MSR (no I/O interface). In Intel Core Duo processors, each core has a unique digital sensor whose temperature is accessible using an MSR. The digital thermal sensor is the preferred method for reading the die temperature because (a) it is located closer to the hottest portions of the die, (b) it enables software to accurately track the die temperature and the potential activation of thermal throttling.

14.8.5.1 Digital Thermal Sensor Enumeration

The processor supports a digital thermal sensor if CPUID.06H:EAX[0] = 1. If the processor supports digital thermal sensor, EBX[bits 3:0] determine the number of thermal thresholds that are available for use.

Software sets thermal thresholds by using the IA32_THERM_INTERRUPT MSR. Software reads output of the digital thermal sensor using the IA32_THERM_STATUS MSR.

14.8.5.2 Reading the Digital Sensor

Unlike traditional analog thermal devices, the output of the digital thermal sensor is a temperature relative to the maximum supported operating temperature of the processor.

Temperature measurements returned by digital thermal sensors are always at or below TCC activation temperature. Critical temperature conditions are detected using the "Critical Temperature Status" bit. When this bit is set, the processor is operating at a critical temperature and immediate shutdown of the system should occur. Once the "Critical Temperature Status" bit is set, reliable operation is not guaranteed.

See Figure 14-31 for the layout of IA32_THERM_STATUS MSR. Bit fields include:

- **Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#). Bit 1 = 1 if PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **PROCHOT# or FORCEPR# Event (bit 2, RO)** — Indicates whether PROCHOT# or FORCEPR# is being asserted by another agent on the platform.

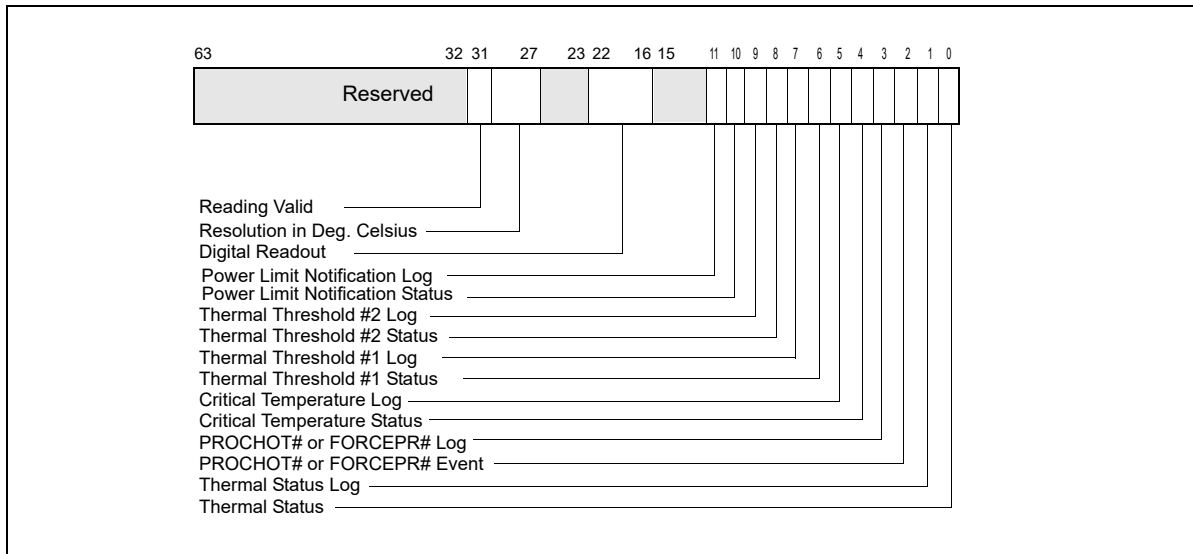


Figure 14-31. IA32_THERM_STATUS Register

- PROCHOT# or FORCEPR# Log (bit 3, R/WC0)** — Sticky bit that indicates whether PROCHOT# or FORCEPR# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, PROCHOT# or FORCEPR# has been externally asserted. Software may clear this bit by writing a zero. External PROCHOT# assertions are only acknowledged if the Bidirectional Prochot feature is enabled.
- Critical Temperature Status (bit 4, RO)** — Indicates whether the critical temperature detector output signal is currently active. If bit 4 = 1, the critical temperature detector output signal is currently active.
- Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual temperature is currently higher than or equal to the value set in Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to TT#1. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.
- Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Threshold #1 has been reached. Software may clear this bit by writing a zero.
- Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual temperature is currently higher than or equal to the value set in Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to TT#2. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.
- Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- Power Limitation Status (bit 10, RO)** — Indicates whether the processor is currently operating below OS-requested P-state (specified in IA32_PERF_CTL) or OS-requested clock modulation duty cycle (specified in IA32_CLOCK_MODULATION). This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification can be delivered independently to IA32_PACKAGE_THERM_STATUS MSR.
- Power Notification Log (bit 11, R/WC0)** — Sticky bit that indicates the processor went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET. This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification is indicated independently in IA32_PACKAGE_THERM_STATUS MSR.

- **Digital Readout (bits 22:16, RO)** — Digital temperature reading in 1 degree Celsius relative to the TCC activation temperature.
0: TCC Activation temperature,
1: (TCC Activation - 1) , etc. See the processor's data sheet for details regarding TCC activation.
A lower reading in the Digital Readout field (bits 22:16) indicates a higher actual temperature.
- **Resolution in Degrees Celsius (bits 30:27, RO)** — Specifies the resolution (or tolerance) of the digital thermal sensor. The value is in degrees Celsius. It is recommended that new threshold values be offset from the current temperature by at least the resolution + 1 in order to avoid hysteresis of interrupt generation.
- **Reading Valid (bit 31, RO)** — Indicates if the digital readout in bits 22:16 is valid. The readout is valid if bit 31 = 1.

Changes to temperature can be detected using two thresholds (see Figure 14-32); one is set above and the other below the current temperature. These thresholds have the capability of generating interrupts using the core's local APIC which software must then service. Note that the local APIC entries used by these thresholds are also used by the Intel® Thermal Monitor; it is up to software to determine the source of a specific interrupt.

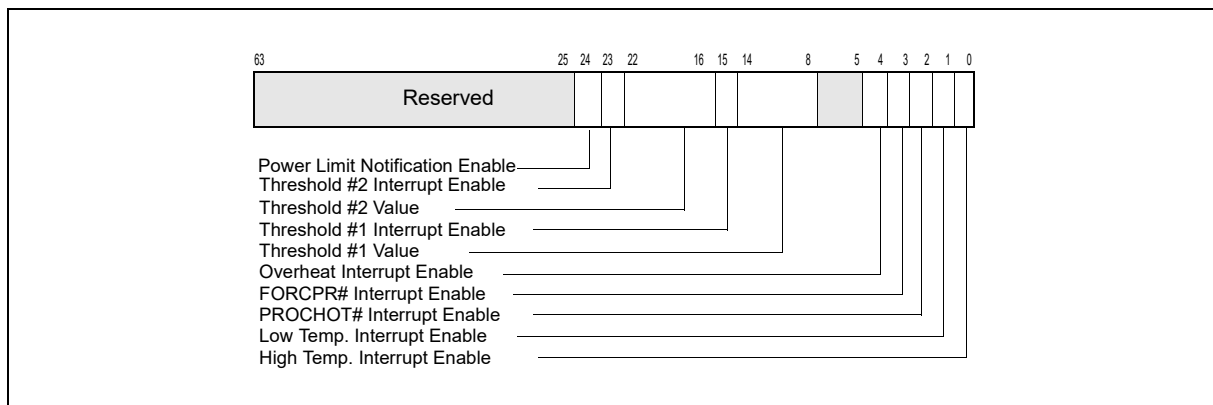


Figure 14-32. IA32_THERM_INTERRUPT Register

See Figure 14-32 for the layout of IA32_THERM_INTERRUPT MSR. Bit fields include:

- **High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.
- **PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **FORCEPR# Interrupt Enable (bit 3, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when FORCEPR# has been asserted by another agent on the platform. Bit 3 = 0 disables the interrupt; bit 3 = 1 enables the interrupt.
- **Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #1 Status and Log bits as well as the Threshold #1 thermal interrupt delivery.

- **Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Threshold #2 Value (bits 22:16, R/W)** — A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #2 Status and Log bits as well as the Threshold #2 thermal interrupt delivery.
- **Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #2 setting in any direction. Bit 23 = 1 enables the interrupt; bit 23 = 0 disables the interrupt.
- **Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of power notification events when the processor went below OS-requested P-state or OS-requested clock modulation duty cycle. This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification can be enabled independently by IA32_PACKAGE_THERM_INTERRUPT MSR.

14.8.6 Power Limit Notification

Platform firmware may be capable of specifying a power limit to restrict power delivered to a platform component, such as a physical processor package. This constraint imposed by platform firmware may occasionally cause the processor to operate below OS-requested P or T-state. A power limit notification event can be delivered using the existing thermal LVT entry in the local APIC.

Software can enumerate the presence of the processor's support for power limit notification by verifying CPUID.06H:EAX[bit 4] = 1.

If CPUID.06H:EAX[bit 4] = 1, then IA32_THERM_INTERRUPT and IA32_THERM_STATUS provides the following facility to manage power limit notification:

- Bits 10 and 11 in IA32_THERM_STATUS informs software of the occurrence of processor operating below OS-requested P-state or clock modulation duty cycle setting (see Figure 14-31).
- Bit 24 in IA32_THERM_INTERRUPT enables the local APIC to deliver a thermal event when the processor went below OS-requested P-state or clock modulation duty cycle setting (see Figure 14-32).

14.9 PACKAGE LEVEL THERMAL MANAGEMENT

The thermal management facilities like IA32_THERM_INTERRUPT and IA32_THERM_STATUS are often implemented with a processor core granularity. To facilitate software manage thermal events from a package level granularity, two architectural MSR is provided for package level thermal management. The IA32_PACKAGE_THERM_STATUS and IA32_PACKAGE_THERM_INTERRUPT MSRs use similar interfaces as IA32_THERM_STATUS and IA32_THERM_INTERRUPT, but are shared in each physical processor package.

Software can enumerate the presence of the processor's support for package level thermal management facility (IA32_PACKAGE_THERM_STATUS and IA32_PACKAGE_THERM_INTERRUPT) by verifying CPUID.06H:EAX[bit 6] = 1.

The layout of IA32_PACKAGE_THERM_STATUS MSR is shown in Figure 14-33.

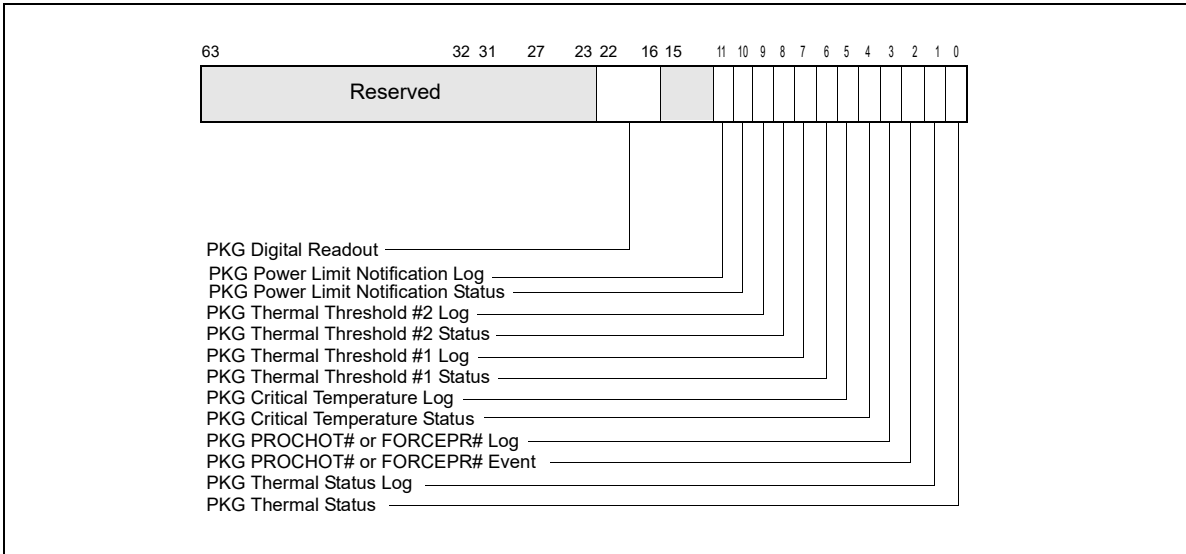


Figure 14-33. IA32_PACKAGE_THERM_STATUS Register

- **Package Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) for the package is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Package Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#) of the package. Bit 1 = 1 if package PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **Package PROCHOT# Event (bit 2, RO)** — Indicates whether package PROCHOT# is being asserted by another agent on the platform.
- **Package PROCHOT# Log (bit 3, R/WC0)** — Sticky bit that indicates whether package PROCHOT# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, package PROCHOT# has been externally asserted. Software may clear this bit by writing a zero.
- **Package Critical Temperature Status (bit 4, RO)** — Indicates whether the package critical temperature detector output signal is currently active. If bit 4 = 1, the package critical temperature detector output signal is currently active.
- **Package Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the package critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to PTT#1. Quantitative information of actual package temperature can be inferred from Package Digital Readout, bits 22:16.
- **Package Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Package Thermal Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to PTT#2. Quantitative information of actual temperature can be inferred from Package Digital Readout, bits 22:16.
- **Package Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Package Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.

- **Package Power Limitation Status (bit 10, RO)** — Indicates package power limit is forcing one or more processors to operate below OS-requested P-state. Note that package power limit violation may be caused by processor cores or by devices residing in the uncore. Software can examine IA32_THERM_STATUS to determine if the cause originates from a processor core (see Figure 14-31).
- **Package Power Notification Log (bit 11, R/WCO)** — Sticky bit that indicates any processor in the package went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET.
- **Package Digital Readout (bits 22:16, RO)** — Package digital temperature reading in 1 degree Celsius relative to the package TCC activation temperature.
 - 0: Package TCC Activation temperature,
 - 1: (PTCC Activation - 1) , etc. See the processor’s data sheet for details regarding PTCC activation.
 A lower reading in the Package Digital Readout field (bits 22:16) indicates a higher actual temperature.

The layout of IA32_PACKAGE_THERM_INTERRUPT MSR is shown in Figure 14-34.

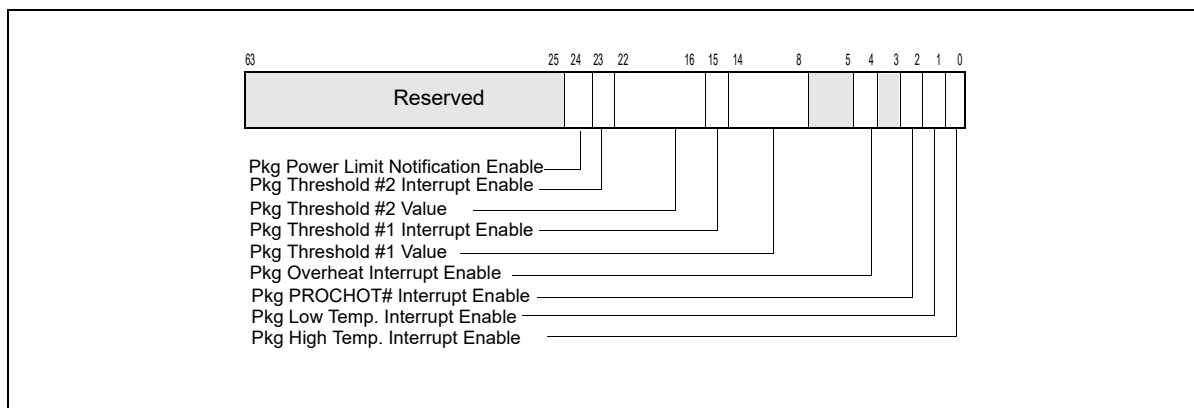


Figure 14-34. IA32_PACKAGE_THERM_INTERRUPT Register

- **Package High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a package high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Package Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.
- **Package PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when Package PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **Package Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Package Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Package Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the Package TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Package Digital Readout and is used to generate the Package Thermal Threshold #1 Status and Log bits as well as the Package Threshold #1 thermal interrupt delivery.
- **Package Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Package Threshold #2 Value (bits 22:16, R/W)** — A temperature threshold, encoded relative to the PTCC Activation temperature (using the same format as the Package Digital Readout). This threshold is compared

against the Package Digital Readout and is used to generate the Package Thermal Threshold #2 Status and Log bits as well as the Package Threshold #2 thermal interrupt delivery.

- **Package Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #2 setting in any direction. Bit 23 = 1 enables the interrupt; bit 23 = 0 disables the interrupt.
- **Package Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of package power notification events.

14.9.1 Support for Passive and Active cooling

Passive and active cooling may be controlled by the OS power management agent through ACPI control methods. On platforms providing package level thermal management facility described in the previous section, it is recommended that active cooling (FAN control) should be driven by measuring the package temperature using the IA32_PACKAGE_THERM_INTERRUPT MSR.

Passive cooling (frequency throttling) should be driven by measuring (a) the core and package temperatures, or (b) only the package temperature. If measured package temperature led the power management agent to choose which core to execute passive cooling, then all cores need to execute passive cooling. Core temperature is measured using the IA32_THERMAL_STATUS and IA32_THERMAL_INTERRUPT MSRs. The exact implementation details depend on the platform firmware and possible solutions include defining two different thermal zones (one for core temperature and passive cooling and the other for package temperature and active cooling).

14.10 PLATFORM SPECIFIC POWER MANAGEMENT SUPPORT

This section covers power management interfaces that are not architectural but addresses the power management needs of several platform specific components. Specifically, RAPL (Running Average Power Limit) interfaces provide mechanisms to enforce power consumption limit. Power limiting usages have specific usages in client and server platforms.

For client platform power limit control and for server platforms used in a data center, the following power and thermal related usages are desirable:

- Platform Thermal Management: Robust mechanisms to manage component, platform, and group-level thermals, either proactively or reactively (e.g., in response to a platform-level thermal trip point).
- Platform Power Limiting: More deterministic control over the system's power consumption, for example to meet battery life targets on rack-level or container-level power consumption goals within a datacenter.
- Power/Performance Budgeting: Efficient means to control the power consumed (and therefore the sustained performance delivered) within and across platforms.

The server and client usage models are addressed by RAPL interfaces, which expose multiple domains of power rationing within each processor socket. Generally, these RAPL domains may be viewed to include hierarchically:

- Package domain is the processor die.
- Memory domain includes the directly-attached DRAM; an additional power plane may constitute a separate domain.

In order to manage the power consumed across multiple sockets via RAPL, individual limits must be programmed for each processor complex. Programming specific RAPL domain across multiple sockets is not supported.

14.10.1 RAPL Interfaces

RAPL interfaces consist of non-architectural MSRs. Each RAPL domain supports the following set of capabilities, some of which are optional as stated below.

- Power limit - MSR interfaces to specify power limit, time window; lock bit, clamp bit etc.
- Energy Status - Power metering interface providing energy consumption information.

- Perf Status (Optional) - Interface providing information on the performance effects (regression) due to power limits. It is defined as a duration metric that measures the power limit effect in the respective domain. The meaning of duration is domain specific.
- Power Info (Optional) - Interface providing information on the range of parameters for a given domain, minimum power, maximum power etc.
- Policy (Optional) - 4-bit priority information that is a hint to hardware for dividing budget between sub-domains in a parent domain.

Each of the above capabilities requires specific units in order to describe them. Power is expressed in Watts, Time is expressed in Seconds, and Energy is expressed in Joules. Scaling factors are supplied to each unit to make the information presented meaningful in a finite number of bits. Units for power, energy, and time are exposed in the read-only MSR_RAPL_POWER_UNIT MSR.

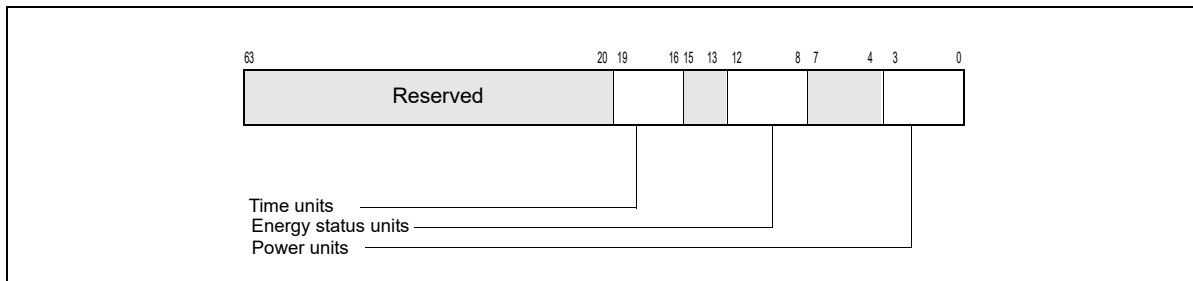


Figure 14-35. MSR_RAPL_POWER_UNIT Register

MSR_RAPL_POWER_UNIT (Figure 14-35) provides the following information across all RAPL domains:

- **Power Units** (bits 3:0): Power related information (in Watts) is based on the multiplier, $1/2^{\text{PU}}$; where PU is an unsigned integer represented by bits 3:0. Default value is 0011b, indicating power unit is in 1/8 Watts increment.
- **Energy Status Units** (bits 12:8): Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 10000b, indicating energy status unit is in 15.3 micro-Joules increment.
- **Time Units** (bits 19:16): Time related information (in Seconds) is based on the multiplier, $1/2^{\text{TU}}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating time unit is in 976 micro-seconds increment.

14.10.2 RAPL Domains and Platform Specificity

The specific RAPL domains available in a platform vary across product segments. Platforms targeting the client segment support the following RAPL domain hierarchy:

- Package
- Two power planes: PP0 and PP1 (PP1 may reflect to uncore devices)

Platforms targeting the server segment support the following RAPL domain hierarchy:

- Package
- Power plane: PPO
- DRAM

Each level of the RAPL hierarchy provides a respective set of RAPL interface MSRs. Table 14-12 lists the RAPL MSR interfaces available for each RAPL domain. The power limit MSR of each RAPL domain is located at offset 0 relative to an MSR base address which is non-architectural (see Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*). The energy status MSR of each domain is located at offset 1 relative to the MSR base address of respective domain.

Table 14-12. RAPL MSR Interfaces and RAPL Domains

Domain	Power Limit (Offset 0)	Energy Status (Offset 1)	Policy (Offset 2)	Perf Status (Offset 3)	Power Info (Offset 4)
PKG	MSR_PKG_POWER_LIMIT	MSR_PKG_ENERGY_STATUS	RESERVED	MSR_PKG_PERF_STATUS	MSR_PKG_POWER_INFO
DRAM	MSR_DRAM_POWER_LIMIT	MSR_DRAM_ENERGY_STATUS	RESERVED	MSR_DRAM_PERF_STATUS	MSR_DRAM_POWER_INFO
PP0	MSR_PP0_POWER_LIMIT	MSR_PP0_ENERGY_STATUS	MSR_PP0_POLICY	MSR_PP0_PERF_STATUS	RESERVED
PP1	MSR_PP1_POWER_LIMIT	MSR_PP1_ENERGY_STATUS	MSR_PP1_POLICY	RESERVED	RESERVED

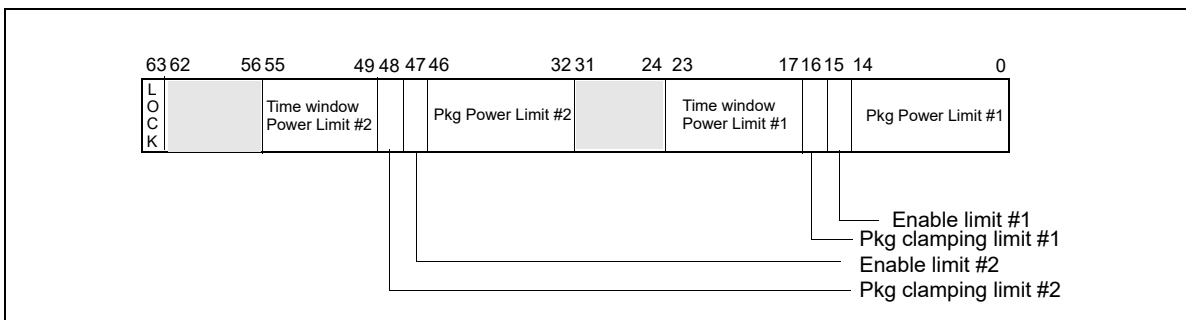
The presence of the optional MSR interfaces (the three right-most columns of Table 14-12) may be model-specific. See Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for details.

14.10.3 Package RAPL Domain

The MSR interfaces defined for the package RAPL domain are:

- MSR_PKG_POWER_LIMIT allows software to set power limits for the package and measurement attributes associated with each limit,
- MSR_PKG_ENERGY_STATUS reports measured actual energy usage,
- MSR_PKG_POWER_INFO reports the package power range information for RAPL usage.

MSR_PKG_PERF_STATUS can report the performance impact of power limiting, but its availability may be model-specific.

**Figure 14-36. MSR_PKG_POWER_LIMIT Register**

MSR_PKG_POWER_LIMIT allows a software agent to define power limitation for the package domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_PKG_POWER_LIMIT. Two power limits can be specified, corresponding to time windows of different sizes. Each power limit provides independent clamping control that would permit the processor cores to go below OS-requested state to meet the power limits. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_PKG_POWER_LIMIT (Figure 14-36) are:

- **Package Power Limit #1** (bits 14:0): Sets the average power usage limit of the package domain corresponding to time window # 1. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.

- **Enable Power Limit #1**(bit 15): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #1** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #1** (bits 23:17): Indicates the time window for power limit #1

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time_Unit}$$

Here "Y" is the unsigned integer value represented. by bits 21:17, "Z" is an unsigned integer represented by bits 23:22. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Package Power Limit #2**(bits 46:32): Sets the average power usage limit of the package domain corresponding to time window # 2. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #2**(bit 47): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #2** (bit 48): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #2** (bits 55:49): Indicates the time window for power limit #2

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time_Unit}$$

Here "Y" is the unsigned integer value represented. by bits 53:49, "Z" is an unsigned integer represented by bits 55:54. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT. This field may have a hard-coded value in hardware and ignores values written by software.
- **Lock** (bit 63): If set, all write attempts to this MSR are ignored until next RESET.

MSR_PKG_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the package domain. This MSR is updated every ~1msec. It has a wraparound time of around 60 secs when power consumption is high, and may be longer otherwise.

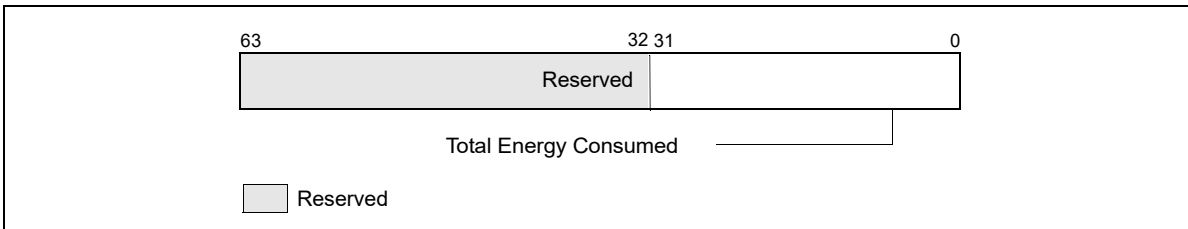


Figure 14-37. MSR_PKG_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PKG_POWER_INFO is a read-only MSR. It reports the package power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the package domain. It also provides the largest possible time window for software to program the RAPL interface.

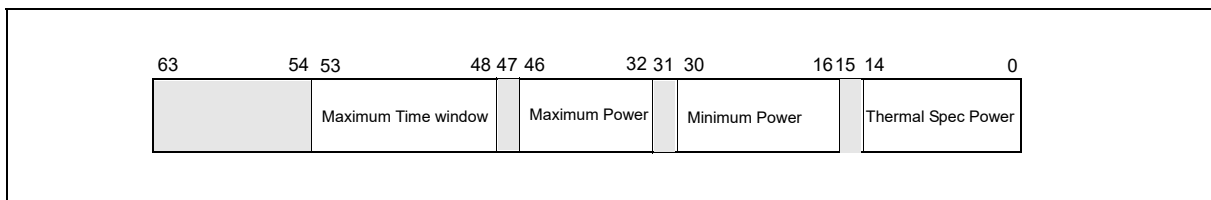


Figure 14-38. MSR_PKG_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.

- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the package domain. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the package domain. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_PKG_POWER_LIMIT. The unit of this field is specified by the “Time Units” field of MSR_RAPL_POWER_UNIT.

MSR_PKG_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific (see Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*).

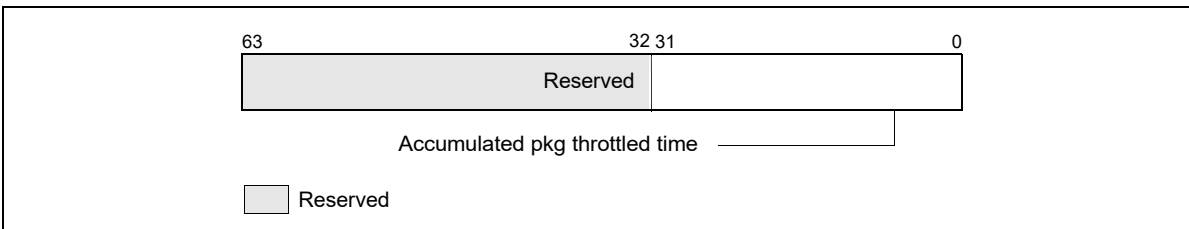


Figure 14-39. MSR_PKG_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the package has throttled. The unit of this field is specified by the “Time Units” field of MSR_RAPL_POWER_UNIT.

14.10.4 PP0/PP1 RAPL Domains

The MSR interfaces defined for the PP0 and PP1 domains are identical in layout. Generally, PP0 refers to the processor cores. The availability of PP1 RAPL domain interface is platform-specific. For a client platform, the PP1 domain refers to the power plane of a specific device in the uncore. For server platforms, the PP1 domain is not supported, but its PP0 domain supports the MSR_PP0_PERF_STATUS interface.

- MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow software to set power limits for the respective power plane domain.
- MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS report actual energy usage on a power plane.
- MSR_PP0_POLICY/MSR_PP1_POLICY allow software to adjust balance for respective power plane.

MSR_PP0_PERF_STATUS can report the performance impact of power limiting, but it is not available in client platforms.

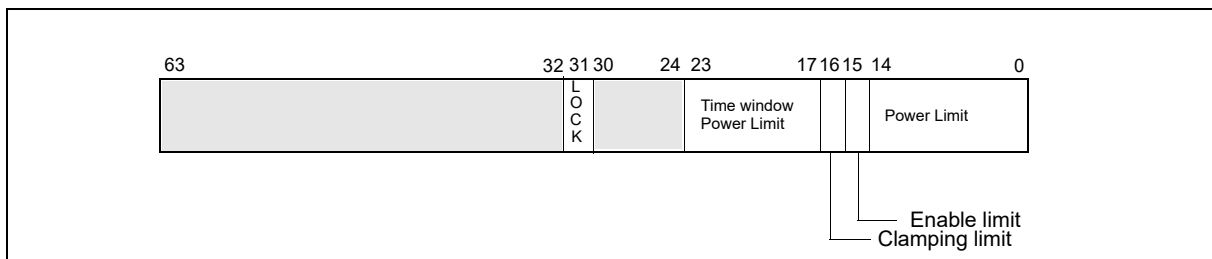


Figure 14-40. MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT Register

MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow a software agent to define power limitation for the respective power plane domain. A lock mechanism in each power plane domain allows the software agent to enforce power limit settings independently. Once a lock bit is set, the power limit settings in that power plane are static and un-modifiable until next RESET.

The bit fields of MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT (Figure 14-40) are:

- **Power Limit** (bits 14:0): Sets the average power usage limit of the respective power plane domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit** (bit 15): 0 = disabled; 1 = enabled.
- **Clamping Limitation** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit #1 will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to the MSR and corresponding policy MSR_PP0_POLICY/MSR_PP1_POLICY are ignored until next RESET.

MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS are read-only MSRs. They report the actual energy use for the respective power plane domains. These MSRs are updated every ~1msec.

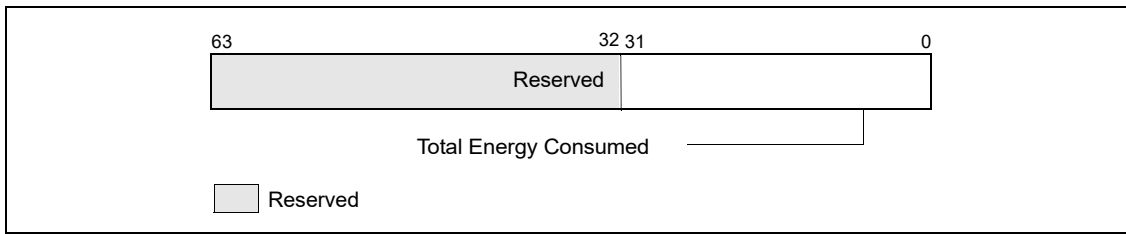


Figure 14-41. MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since the last time this register was cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PP0_POLICY/MSR_PP1_POLICY provide balance power policy control for each power plane by providing inputs to the power budgeting management algorithm. On platforms that support PP0 (IA cores) and PP1 (uncore graphic device), the default values give priority to the non-IA power plane. These MSRs enable the PCU to balance power consumption between the IA cores and uncore graphic device.

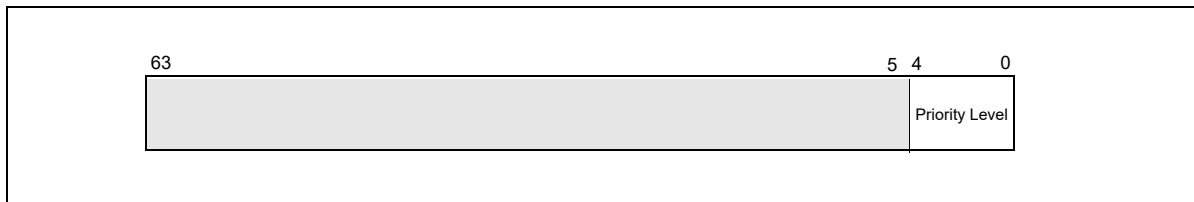


Figure 14-42. MSR_PP0_POLICY/MSR_PP1_POLICY Register

- **Priority Level** (bits 4:0): Priority level input to the PCU for respective power plane. PP0 covers the IA processor cores, PP1 covers the uncore graphic device. The value 31 is considered highest priority.

MSR_PP0_PERF_STATUS is a read-only MSR. It reports the total time for which the PP0 domain was throttled due to the power limits. This MSR is supported only in server platform. Throttling in this context is defined as going below the OS-requested P-state or T-state.

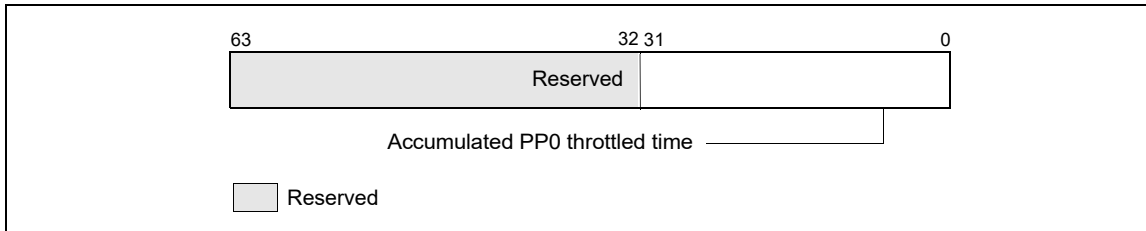


Figure 14-43. MSR_PPO_PERF_STATUS MSR

- **Accumulated PPO Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the PPO domain has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

14.10.5 DRAM RAPL Domain

The MSR interfaces defined for the DRAM domains are supported only in the server platform. The MSR interfaces are:

- MSR_DRAM_POWER_LIMIT allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.
- MSR_DRAM_ENERGY_STATUS reports measured actual energy usage.
- MSR_DRAM_POWER_INFO reports the DRAM domain power range information for RAPL usage.
- MSR_DRAM_PERF_STATUS can report the performance impact of power limiting.

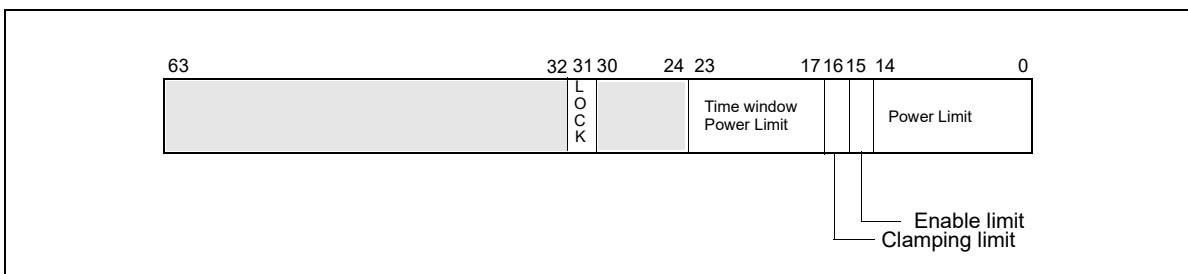


Figure 14-44. MSR_DRAM_POWER_LIMIT Register

MSR_DRAM_POWER_LIMIT allows a software agent to define power limitation for the DRAM domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_DRAM_POWER_LIMIT. A power limit can be specified along with a time window. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and unmodifiable until next RESET.

The bit fields of MSR_DRAM_POWER_LIMIT (Figure 14-44) are:

- **DRAM Power Limit #1** (bits 14:0): Sets the average power usage limit of the DRAM domain corresponding to time window # 1. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #1** (bit 15): 0 = disabled; 1 = enabled.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to this MSR are ignored until next RESET.

MSR_DRAM_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the DRAM domain. This MSR is updated every ~1msec.

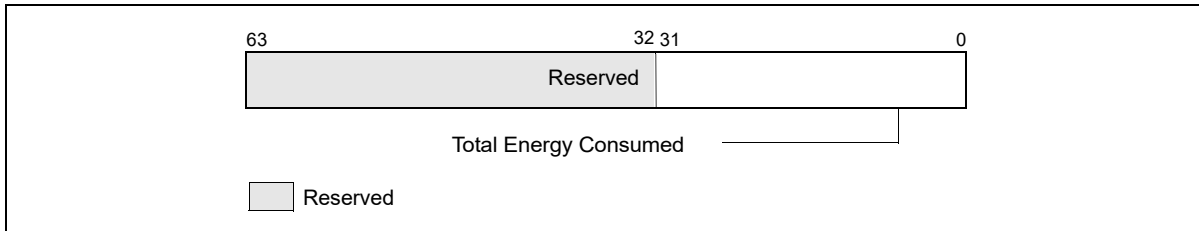


Figure 14-45. MSR_DRAM_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_POWER_INFO is a read-only MSR. It reports the DRAM power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the DRAM domain. It also provides the largest possible time window for software to program the RAPL interface.

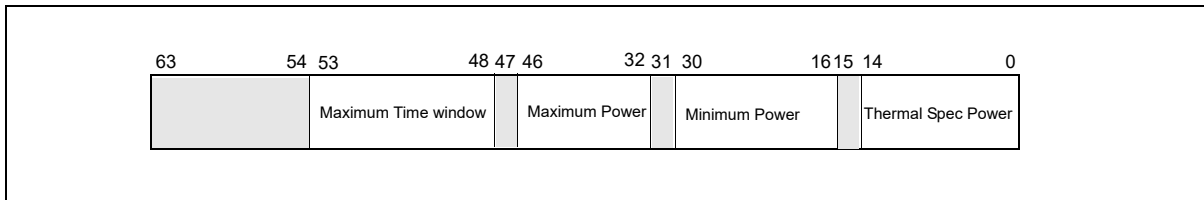


Figure 14-46. MSR_DRAM_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_DRAM_POWER_LIMIT. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific (see Chapter 2, "Model-Specific Registers (MSRs)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4).

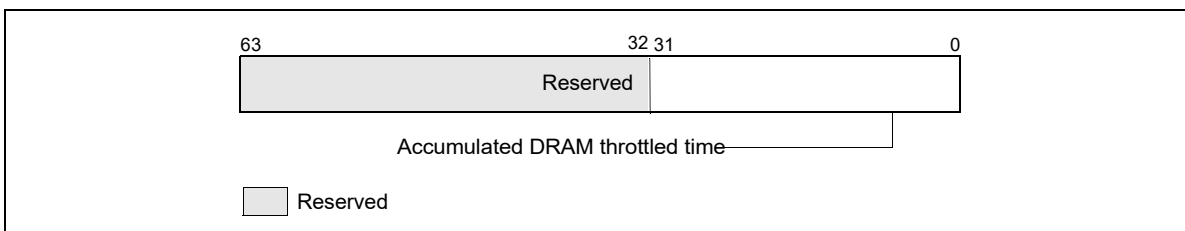


Figure 14-47. MSR_DRAM_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the DRAM domain has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

10. Updates to Chapter 17, Volume 3B

Change bars and green text show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter: Typo corrections in section 17.19.4.2, "Cache Allocation Technology: Resource Type and Capability Enumeration".

CHAPTER 17

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 18 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 17.17, “Time-Stamp Counter”.
- Features which allow monitoring of shared platform resources such as the L3 cache are described in Section 17.18, “Intel® Resource Director Technology (Intel® RDT) Monitoring Features”.
- Features which enable control over shared platform resources are described in Section 17.19, “Intel® Resource Director Technology (Intel® RDT) Allocation Features”.

17.1 OVERVIEW OF DEBUG SUPPORT FACILITIES

The following processor facilities support debugging and performance monitoring:

- **Debug exception (#DB)** — Transfers program control to a debug procedure or task when a debug event occurs.
- **Breakpoint exception (#BP)** — See breakpoint instruction (INT3) below.
- **Breakpoint-address registers (DR0 through DR3)** — Specifies the addresses of up to 4 breakpoints.
- **Debug status register (DR6)** — Reports the conditions that were in effect when a debug or breakpoint exception was generated.
- **Debug control register (DR7)** — Specifies the forms of memory or I/O access that cause breakpoints to be generated.
- **T (trap) flag, TSS** — Generates a debug exception (#DB) when an attempt is made to switch to a task with the T flag set in its TSS.
- **RF (resume) flag, EFLAGS register** — Suppresses multiple exceptions to the same instruction.
- **TF (trap) flag, EFLAGS register** — Generates a debug exception (#DB) after every execution of an instruction.
- **Breakpoint instruction (INT3)** — Generates a breakpoint exception (#BP) that transfers program control to the debugger procedure or task. This instruction is an alternative way to set instruction breakpoints. It is especially useful when more than four breakpoints are desired, or when breakpoints are being placed in the source code.

- **Last branch recording facilities** — Store branch records in the last branch record (LBR) stack MSR for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consist of a branch-from and a branch-to instruction address. Send branch records out on the system bus as branch trace messages (BTMs).

These facilities allow a debugger to be called as a separate task or as a procedure in the context of the current program or task. The following conditions can be used to invoke the debugger:

- Task switch to a specific task.
- Execution of the breakpoint instruction.
- Execution of any instruction.
- Execution of an instruction at a specified address.
- Read or write to a specified memory address/range.
- Write to a specified memory address/range.
- Input from a specified I/O address/range.
- Output to a specified I/O address/range.
- Attempt to change the contents of a debug register.

17.2 DEBUG REGISTERS

Eight debug registers (see Figure 17-1 for 32-bit operation and Figure 17-2 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.

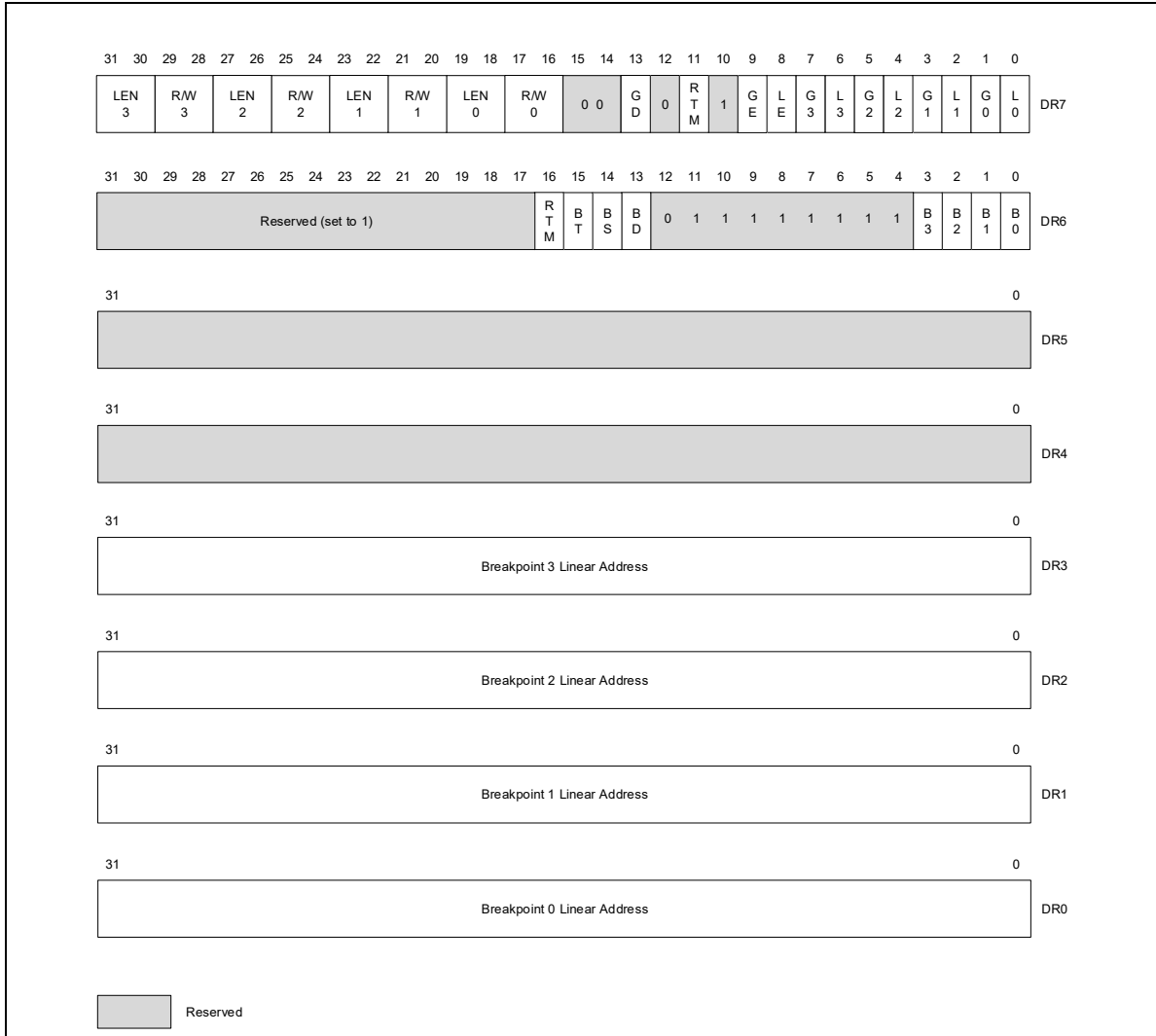


Figure 17-1. Debug Registers

Debug registers are privileged resources; a MOV instruction that accesses these registers can only be executed in real-address mode, in SMM or in protected mode at a CPL of 0. An attempt to read or write the debug registers from any other privilege level generates a general-protection exception (#GP).

The primary function of the debug registers is to set up and monitor from 1 to 4 breakpoints, numbered 0 through 3. For each breakpoint, the following information can be specified:

- The linear address where the breakpoint is to occur.
- The length of the breakpoint location: 1, 2, 4, or 8 bytes (refer to the notes in Section 17.2.4).
- 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.

17.2.1 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 17-1). Breakpoint comparisons are made before physical address translation occurs. The contents of debug register DR7 further specifies breakpoint conditions.

17.2.2 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.

17.2.3 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 17-1). 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.
- **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 17.2.4, “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 7.2.1, “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 17.3.3). 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.

17.2.4 Debug Control Register (DR7)

The debug control register (DR7) enables or disables breakpoints and sets breakpoint conditions (see Figure 17-1). 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 Ln 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 Gn 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 17.3.3). 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/Wn 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/Wn 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 R/Wn field in register DR7 is 00 (instruction execution), then the $LENn$ field should also be 00. The effect of using other lengths is undefined. See Section 17.2.5, “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 $LENn$ 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 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.

17.2.5 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 17-1 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 17-1. Breakpoint Examples

Debug Register Setup			
Debug Register	R/W _n	Breakpoint Address	LEN _n
DR0	R/W0 = 11 (Read/Write)	A0001H	LEN0 = 00 (1 byte)
DR1	R/W1 = 01 (Write)	A0002H	LEN1 = 00 (1 byte)
DR2	R/W2 = 11 (Read/Write)	B0002H	LEN2 = 01 (2 bytes)
DR3	R/W3 = 01 (Write)	C0000H	LEN3 = 11 (4 bytes)
Data Accesses			
Operation	Address	Access Length (In Bytes)	
Data operations that trap			
- Read or write	A0001H		1
- Read or write	A0001H		2
- Write	A0002H		1
- Write	A0002H		2
- Read or write	B0001H		4
- Read or write	B0002H		1
- Read or write	B0002H		2
- Write	C0000H		4
- Write	C0001H		2
- Write	C0003H		1
Data operations that do not trap			
- Read or write	A0000H		1
- Read	A0002H		1
- Read or write	A0003H		4
- Read or write	B0000H		2
- Read	C0000H		2
- Read or write	C0004H		4

17.2.6 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 17-2). 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). Break point conditions for 8-byte memory read/writes are supported in all modes.

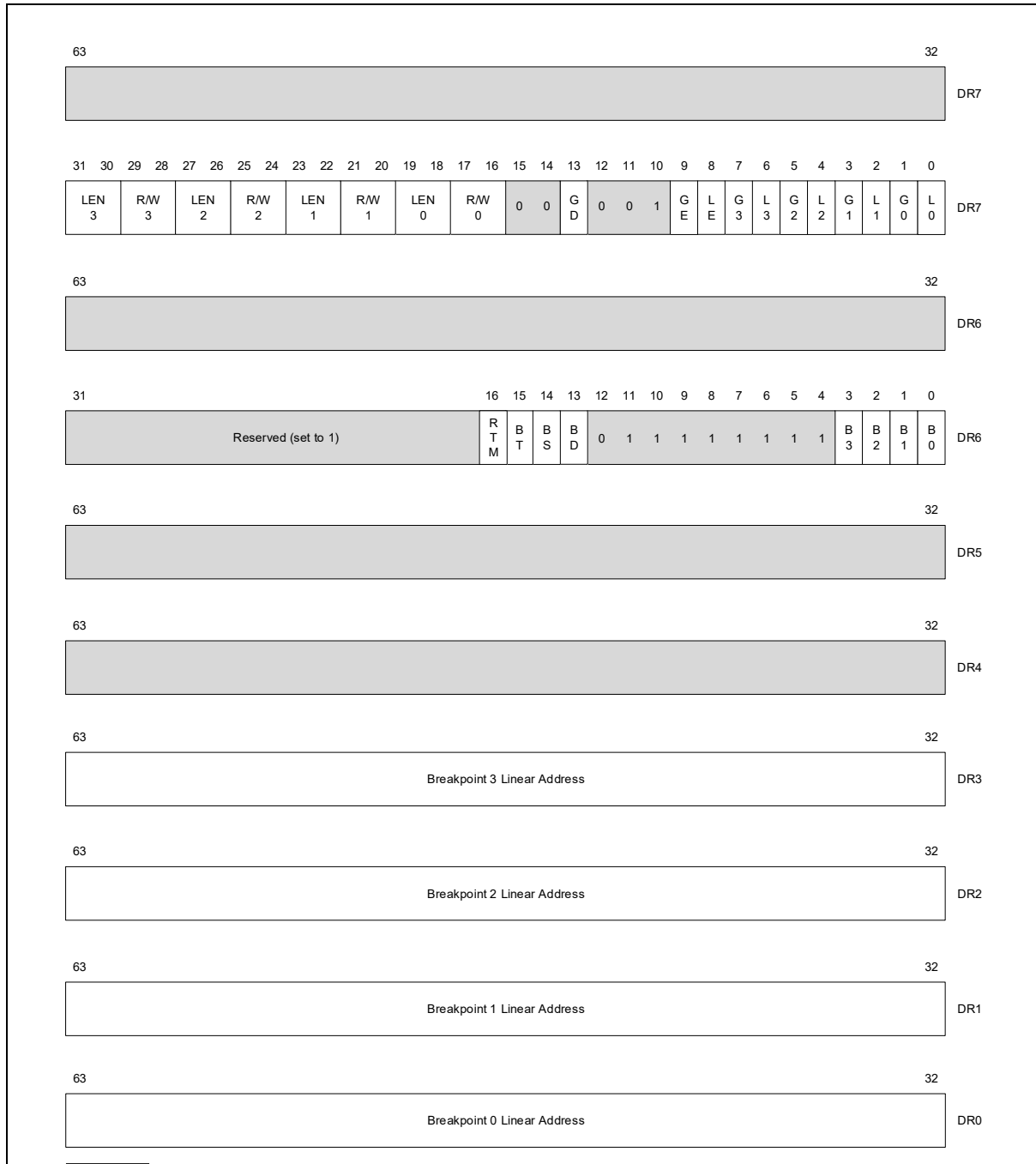


Figure 17-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

17.3 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.

17.3.1 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 17-2 shows the states of these flags following the generation of each kind of breakpoint condition.

Instruction-breakpoint and general-detect condition (see Section 17.3.1.3, “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*.

Table 17-2. Debug Exception Conditions

Debug or Breakpoint Condition	DR6 Flags Tested	DR7 Flags Tested	Exception Class
Single-step trap	BS = 1		Trap
Instruction breakpoint, at addresses defined by DR _n and LEN _n	B _n = 1 and (G _n or L _n = 1)	R/W _n = 0	Fault
Data write breakpoint, at addresses defined by DR _n and LEN _n	B _n = 1 and (G _n or L _n = 1)	R/W _n = 1	Trap
I/O read or write breakpoint, at addresses defined by DR _n and LEN _n	B _n = 1 and (G _n or L _n = 1)	R/W _n = 2	Trap
Data read or write (but not instruction fetches), at addresses defined by DR _n and LEN _n	B _n = 1 and (G _n or L _n = 1)	R/W _n = 3	Trap
General detect fault, resulting from an attempt to modify debug registers (usually in conjunction with in-circuit emulation)	BD = 1	None	Fault
Task switch	BT = 1	None	Trap
INT1 instruction	None	None	Trap

17.3.1.1 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.

17.3.1.2 Data Memory and I/O Breakpoint Exception Conditions

Data memory and I/O breakpoints are reported when the processor attempts to access a memory or I/O address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect data or I/O accesses (R/W flag is set to 1, 2, or 3). The processor generates the exception after it executes the instruction that made the access, so these breakpoint condition causes a trap-class exception to be generated.

Because data breakpoints are traps, an instruction that writes memory overwrites the original data before the debug exception generated by a data breakpoint is generated. If a debugger needs to save the contents of a write breakpoint location, it should save the original contents before setting the breakpoint. The handler can report the saved value after the breakpoint is triggered. The address in the debug registers can be used to locate the new value stored by the instruction that triggered the breakpoint.

If a data breakpoint is detected during an iteration of a string instruction executed with fast-string operation (see Section 7.3.9.3 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*), delivery of the resulting debug exception may be delayed until completion of the corresponding group of iterations.

Intel486 and later processors ignore the GE and LE flags in DR7. In Intel386 processors, exact data breakpoint matching does not occur unless it is enabled by setting the LE and/or the GE flags.

For repeated INS and OUTS instructions that generate an I/O-breakpoint debug exception, the processor generates the exception after the completion of the first iteration. Repeated INS and OUTS instructions generate a data-breakpoint debug exception after the iteration in which the memory address breakpoint location is accessed.

If an execution of the MOV or POP instruction loads the SS register and encounters a data breakpoint, the resulting debug exception is delivered after completion of the next instruction (the one after the MOV or POP).

Any pending data or I/O breakpoints are lost upon delivery of an exception. For example, if a machine-check exception (#MC) occurs following an instruction that encounters a data breakpoint (but before the resulting debug exception is delivered), the data breakpoint is lost. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, the data breakpoint is lost if the next instruction causes a fault.

Delivery of events due to INT *n*, INT3, or INTO does not cause a loss of data breakpoints. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, and the next instruction is software interrupt (INT *n*, INT3, or INTO), a debug exception (#DB) resulting from a data breakpoint will be delivered after the transition to the software-interrupt handler. The #DB handler should account for the fact that the #DB may have been delivered after a invocation of a software-interrupt handler, and in particular that the CPL may have changed between recognition of the data breakpoint and delivery of the #DB.

17.3.1.3 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.

17.3.1.4 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.

17.3.1.5 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.

17.3.2 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.

17.3.3 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]).

17.4 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 17.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)”
- Section 17.6, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Microarchitecture”
- Section 17.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture”

- Section 17.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture”
- Section 17.11, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Haswell Microarchitecture”
- Section 17.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture”
- Section 17.14, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors)”
- Section 17.15, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors)”
- Section 17.16, “Last Branch, Interrupt, and Exception Recording (P6 Family Processors)”

The following subsections of Section 17.4 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).

17.4.1 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 17-3 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 17.5.1, “LBR Stack” (Intel® Core™2 Duo and Intel Atom® processor family) and Section 17.9.1, “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 17.4.3, “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 17.4.4, “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 17.4.9, “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 17.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

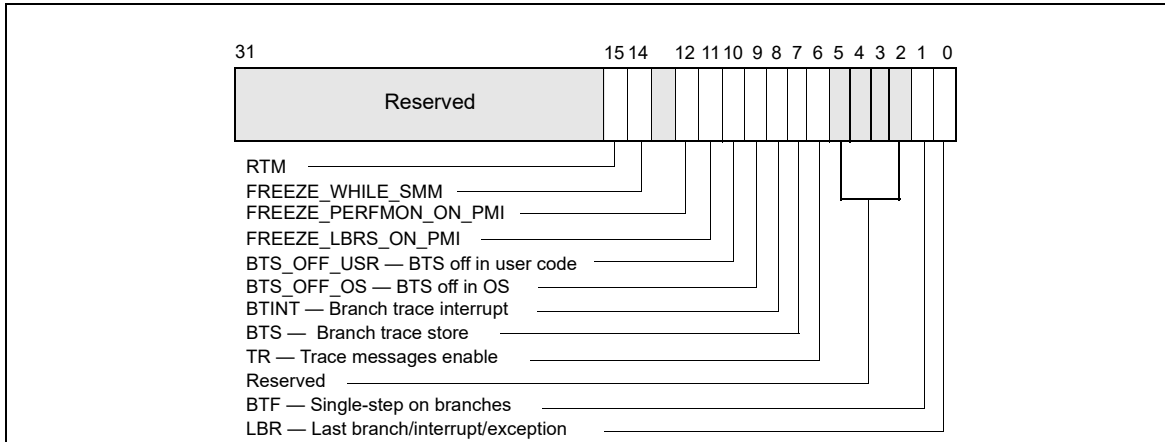


Figure 17-3. IA32_DEBUGCTL MSR for Processors based on Intel Core microarchitecture

- **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 17.13.2.
- **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 17.13.2.
- **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 17.4.7 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 17.4.7 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 14.6.11 for more details about Intel Thread Director enable, reset, and history reset operations).
Subsequently, the enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its service. If Intel Thread Director support is enabled when RSM is executed, then the processor resets the Intel Thread Director history.
Note that system software must check if the processor supports the IA32_DEBUGCTL.FREEZE_WHILE_SMM control bit. IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 19.8 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 17.3.3.

17.4.2 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.

17.4.3 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.

17.4.4 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 17.4.2, “Monitoring Branches, Exceptions, and Interrupts.”

For P6 processor family, Pentium M processor family, 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).

17.4.4.1 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.

17.4.5 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 17.4.9.2, “Setting Up the DS Save Area.” for additional details.

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 7.2.1, “Task-State Segment (TSS).”

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.

17.4.6 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 17.4.9.4. 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.

17.4.7 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 19-3). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 19.6.2.1, “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 19-1) or IA32_FIXED_CTR_CTRL MSR (see Figure 19-2).
- The PEBS buffer is almost full and reaches the interrupt threshold.
- The BTS buffer is almost full and reaches the interrupt threshold.

Table 17-3 compares the interaction of the processor with the PMI handler using the legacy versus streamlined Freeza_Perfmon_On_PMI interface.

Table 17-3. 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

17.4.8 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 17-4 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily_DisplayModel encoding (see CPUID instruction in Chapter 3 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*).

Table 17-4. 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 17.12.

The last branch recording mechanism tracks not only branch instructions (like JMP, Jcc, LOOP and CALL instructions), but also other operations that cause a change in the instruction pointer (like external interrupts, traps and faults). The branch recording mechanisms generally employ a set of MSRs, referred to as last branch record (LBR) stack. The size and exact locations of the LBR stack are generally model-specific (see Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for model-specific MSR addresses).

- **Last Branch Record (LBR) Stack** — The LBR consists of N pairs of MSRs (N is listed in the LBR stack size column of Table 17-4) that store source and destination address of recent branches (see Figure 17-3):
 - 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 17-4.

17.4.8.1 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.

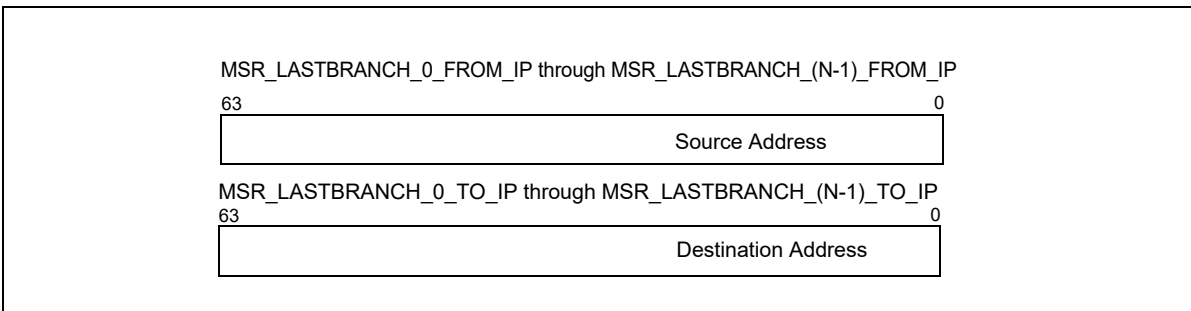


Figure 17-4. 64-bit Address Layout of LBR MSR

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, 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, 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 17.6).

- **000111B (64-bit LIP record format), Flags, 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).

17.4.8.2 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.

17.4.8.3 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.

17.4.9 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 17.4.5, "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 19.6.2.4, "Processor Event Based Sampling (PEBS)," and the relevant PEBS sub-sections across the core PMU sections in Chapter 19, "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 19.6.2.4.2 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 19.5.3.1 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 17-5). 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.

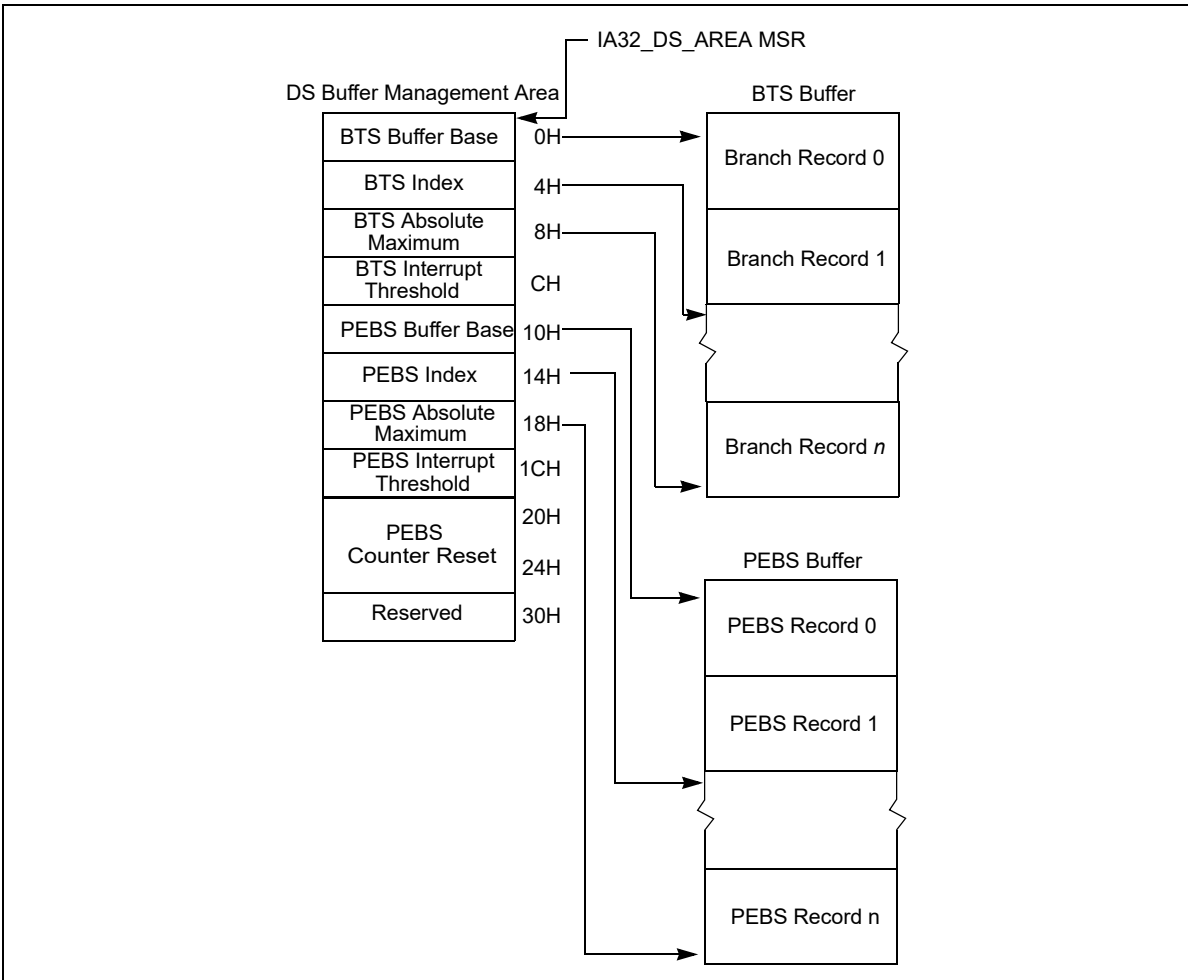


Figure 17-5. DS Save Area Example¹

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

Figure 17-6 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).

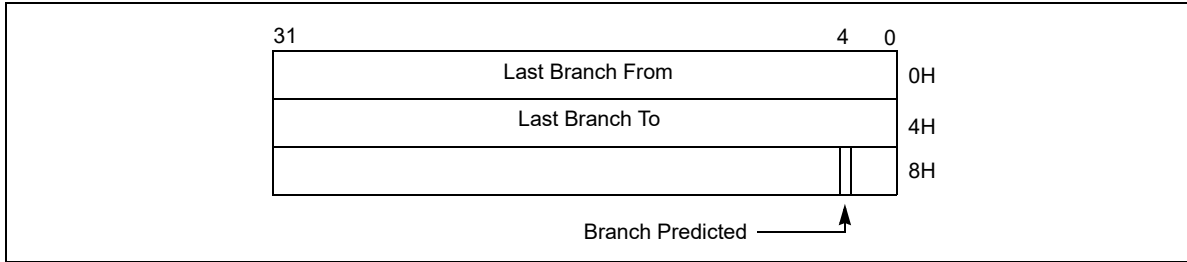


Figure 17-6. 32-bit Branch Trace Record Format

Figure 17-7 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.

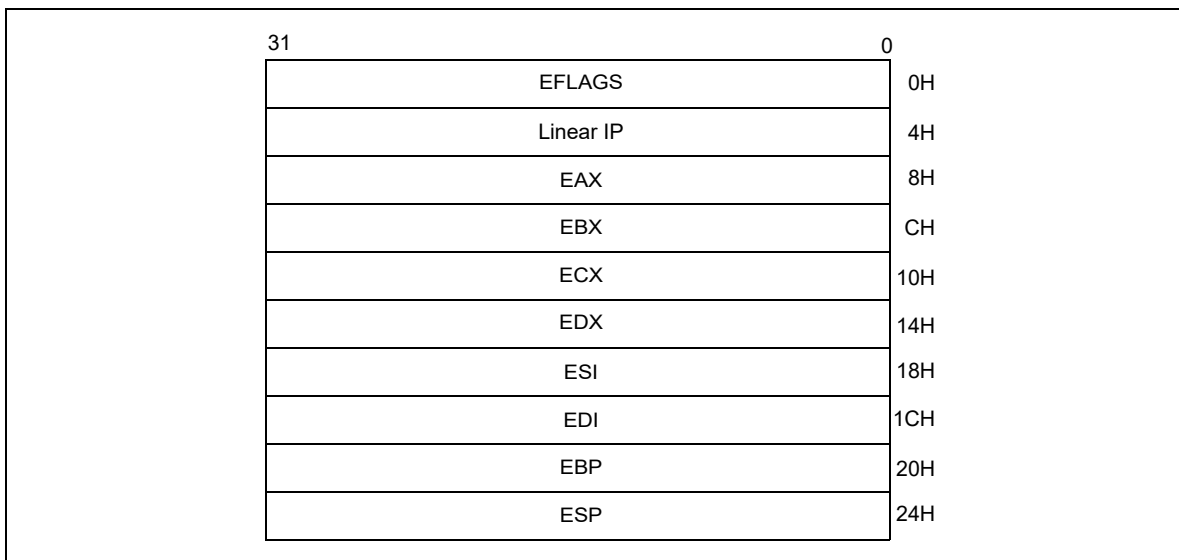


Figure 17-7. PEBS Record Format

17.4.9.1 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 17-8.

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 17-8. If IA-32e mode is not active the structure of the DS save area is as shown in Figure 17-5.

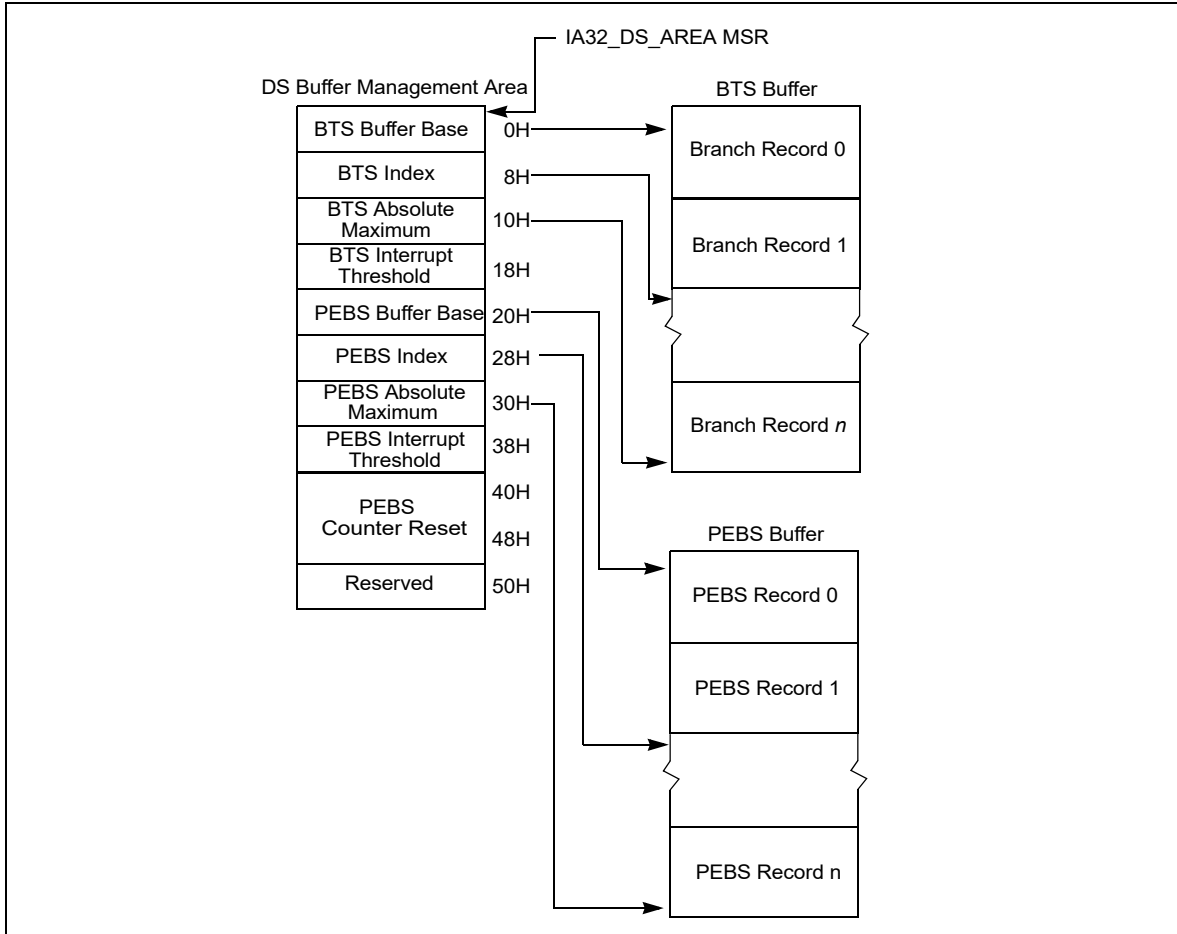


Figure 17-8. IA-32e Mode DS Save Area Example¹

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

The IA32_DS_AREA MSR holds the 64-bit linear address of the first byte of the DS buffer management area. The structure of a branch trace record is similar to that shown in Figure 17-6, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 17-9). The structure of a PEBS record is similar to that shown in Figure 17-7, 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 17-10).

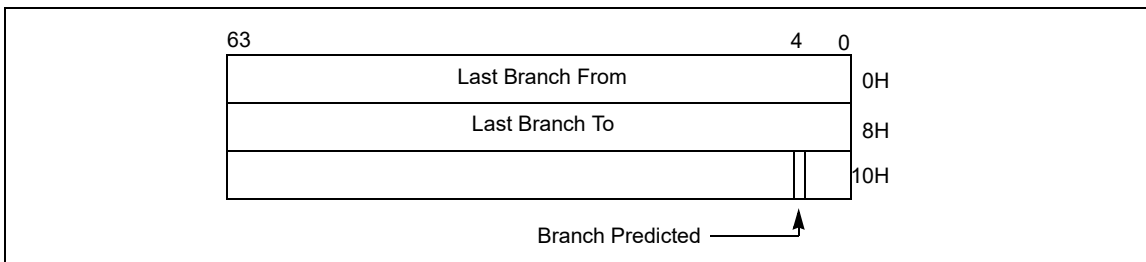


Figure 17-9. 64-bit Branch Trace Record Format

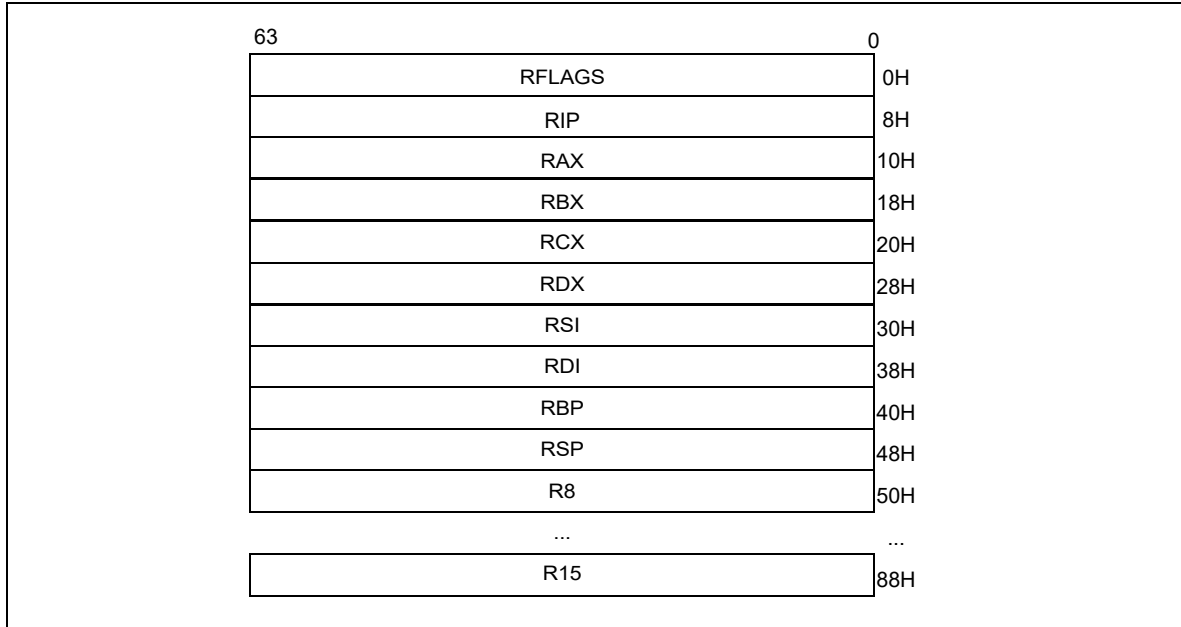


Figure 17-10. 64-bit PEBS Record Format

Fields in the buffer management area of a DS save area are described in Section 17.4.9.

The format of a branch trace record and a PEBS record are the same as the 64-bit record formats shown in Figures 17-9 and Figures 17-10, 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 17.4.9.3 and Section 17.4.9.4.

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 19-3).

17.4.9.2 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 19.6.2.4.1, “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 17.4.9, “BTS and DS Save Area,” and Section 17.4.9.1, “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 10.5.1, “Local Vector Table.”
4. Establish an interrupt handler in the IDT for the vector associated with the performance counter entry in the xAPIC LVT.

- Write an interrupt service routine to handle the interrupt. See Section 17.4.9.5, “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 19.3.9.1 and Section 19.9.5 for more information. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.
- The DS save area can be larger than a page, but the pages must be mapped to contiguous linear addresses. The buffer may share a page, so it need not be aligned on a 4-KByte boundary. For performance reasons, the base of the buffer must be aligned on a doubleword boundary and should be aligned on a cache line boundary.
- It is recommended that the buffer size for the BTS buffer and the PEBS buffer be an integer multiple of the corresponding record sizes.
- The precise event records buffer should be large enough to hold the number of precise event records that can occur while waiting for the interrupt to be serviced.
- The DS save area should be in kernel space. It must not be on the same page as code, to avoid triggering self-modifying code actions.
- There are no memory type restrictions on the buffers, although it is recommended that the buffers be designated as WB memory type for performance considerations.
- Either the system must be prevented from entering A20M mode while DS save area is active, or bit 20 of all addresses within buffer bounds must be 0.
- Pages that contain buffers must be mapped to the same physical addresses for all processes, such that any change to control register CR3 will not change the DS addresses.
- The DS save area is expected to be used only on systems with an enabled APIC. The LVT Performance Counter entry in the APIC must be initialized to use an interrupt gate instead of the trap gate.

17.4.9.3 Setting Up the BTS Buffer

Three flags in the MSR_DEBUGCTLA MSR (see Table 17-5), IA32_DEBUGCTL (see Figure 17-3), or MSR_DEBUGCTLB (see Figure 17-16) 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 17-5. 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:

- 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.
- 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.

17.4.9.4 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 17-6.

Table 17-6. 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

17.4.9.5 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.

17.5 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 17.4.1 for a description of the flags. See Figure 17-3 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 17.5.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts**
 - See Section 17.4.2 and Section 17.4.3. 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 17.4.4.
- **Last exception records** — See Section 17.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 17.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 17.4.7 for legacy Freeze_LBRS_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 17.4.7 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 17.4.1.

17.5.1 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 17.4.8.1. 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.

17.5.2 LBR Stack in Intel Atom® Processors based on the Silvermont Microarchitecture

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in Intel Atom processors based on the Silvermont and Airmont microarchitectures. Eight pairs of MSRs are supported in the LBR stack.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 17-11.

17.6 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 17.5.2 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 17-13.
- Elapsed cycle information is added to MSR_LASTBRANCH_x_TO_IP. Format is shown in Table 17-7.
- Misprediction info is reported in the upper bits of MSR_LASTBRANCH_x_FROM_IP. MISPREDE bit format is shown in Table 17-8.
- Streamlined Freeze_LBRs_On_PMI operation; see Section 17.12.2.
- LBR MSRs may be cleared when MWAIT is used to request a C-state that is numerically higher than C1; see Section 17.12.3.

Table 17-7. 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 17.4.8.1 for address format.
Cycle Count (Saturating)	63:48	R/W	Elapsed core clocks since last update to the LBR stack.

17.7 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 17.6 with the following changes:

- Enumeration of new LBR format: encoding 00111b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 17.4.8.1.
- Each LBR stack entry consists of three MSRs:
 - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 17-9.
 - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 17-9.
 - MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data. Layout is the same as Table 17-16.

17.8 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 17-11.

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 17.4.8.1. 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.

17.9 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 17.4.1 for a description of the flags. See Figure 17-11 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 17.9.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts** — See Section 17.4.2 and Section 17.4.3. 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 17.4.4. However, not all BTM messages are observable using the Intel® QPI link.
- **Last exception records** — See Section 17.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 17.4.6 and Section 17.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 17.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 17.4.7 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 17.4.1.

Processors based on Nehalem microarchitecture provide additional capabilities:

- **Independent control of uncore PMI** — The IA32_DEBUGCTL MSR provides a bit field (see Figure 17-11) 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.

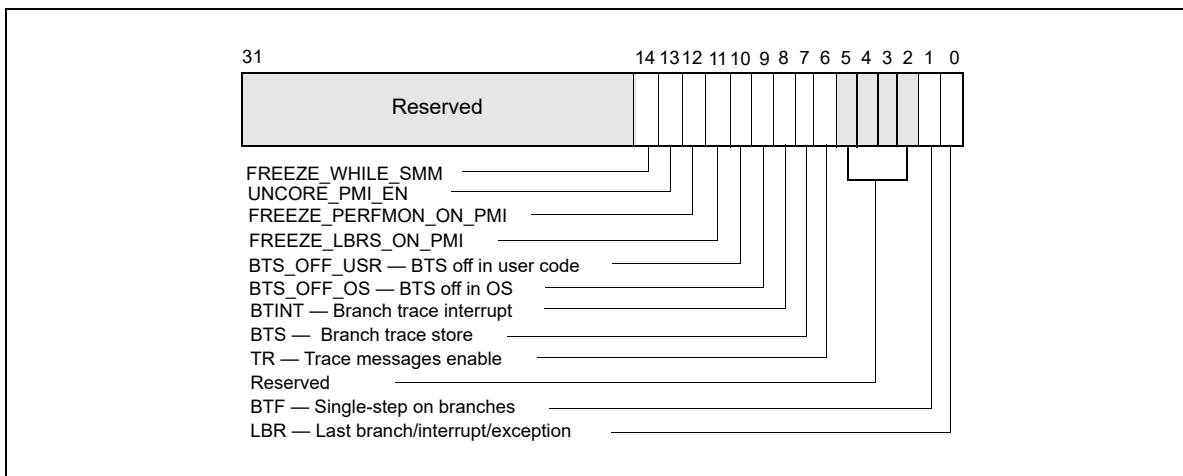


Figure 17-11. IA32_DEBUGCTL MSR for Processors Based on Nehalem Microarchitecture

17.9.1 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 17-8 and Table 17-9.

Table 17-8. MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch from” address. See Section 17.4.8.1 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 17-9. MSR_LASTBRANCH_x_TO_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 17.4.8.1 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 17-10.

Table 17-10. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_1AH	16	0 to 15

17.9.2 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 17-11.

Table 17-11. 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

17.10 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 17.9, “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 17-12.

Table 17-12. 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

17.11 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 17.10, “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 17-13. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 17.10.

Table 17-13. 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.)

17.11.1 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 17-14 and Table 17-9.

Table 17-14. 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 17.4.8.1 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.

17.12 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 17.4.8.1.
- Each LBR stack entry consists of a triplets of MSRs:

- MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 17-9.
- MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 17-9.
- MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data.
- Size of LBR stack increased to 32.

Processors based on the Skylake microarchitecture supports the same LBR filtering capabilities as described in Table 17-13.

Table 17-15. 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

17.12.1 MSR_LBR_INFO_x MSR

The layout of each MSR_LBR_INFO_x MSR is shown in Table 17-16.

Table 17-16. 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.

17.12.2 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 17.4.7).

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.

17.12.3 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.

17.13 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_LASTBRANCH_0 through MSR_LASTBRANCH_3) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. The LBR stack consists of 16 MSR pairs (MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_15_FROM_IP and MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_15_TO_IP) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H].
- **Last branch record top-of-stack (TOS) pointer** — The TOS Pointer MSR contains a 2-bit pointer (0-3) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. This pointer becomes a 4-bit pointer (0-15) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H]. See also: Table 17-17, Figure 17-12, and Section 17.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **Last exception record** — See Section 17.13.3, “Last Exception Records.”

17.13.1 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 17-12 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 17.13.2, “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 17.4.3, “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 17.4.4, “Branch Trace Messages.”

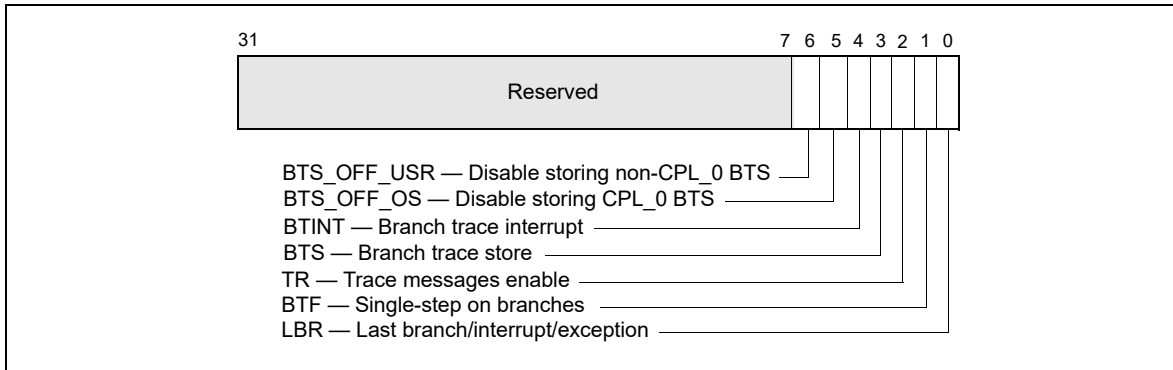


Figure 17-12. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **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 17.4.9, “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 17.4.5, “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 17.13.2, “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 17.13.2, “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 17-12. 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.

17.13.2 LBR Stack for Processors Based on Intel NetBurst® Microarchitecture

The LBR stack is made up of LBR MSRs that are treated by the processor as a circular stack. The TOS pointer (MSR_LASTBRANCH_TOS MSR) points to the LBR MSR (or LBR MSR pair) that contains the most recent (last) branch record placed on the stack. Prior to placing a new branch record on the stack, the TOS is incremented by 1. When the TOS pointer reaches its maximum value, it wraps around to 0. See Table 17-17 and Figure 17-12.

Table 17-17. 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 17-13 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.

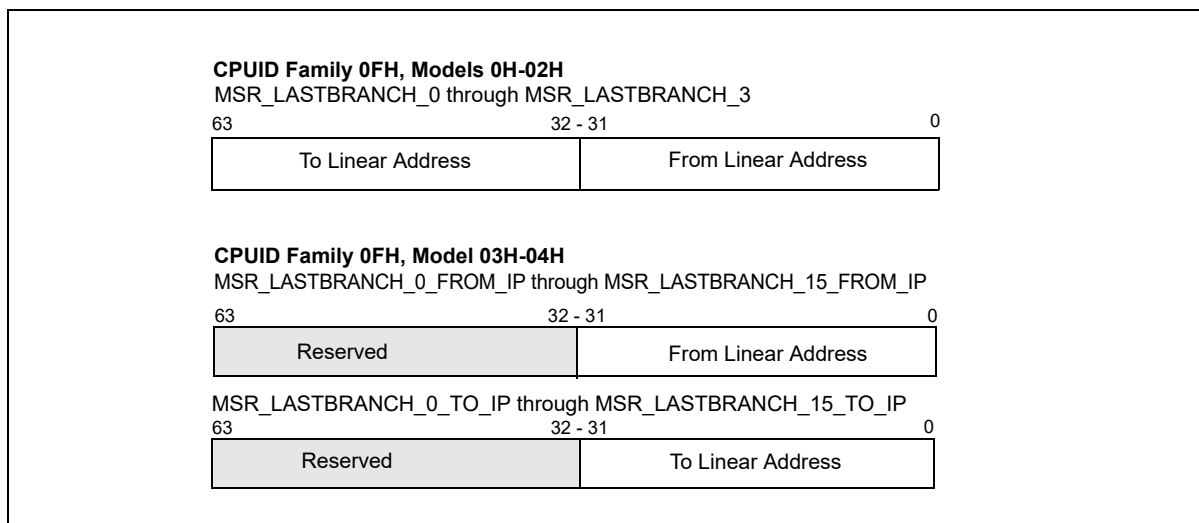


Figure 17-13. LBR MSR Branch Record Layout for the Pentium 4 and Intel® Xeon® Processor Family

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.

17.13.3 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.

17.14 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 17-14 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 17.4.3, “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 17.4.4, “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 17.4.9, “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 17.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

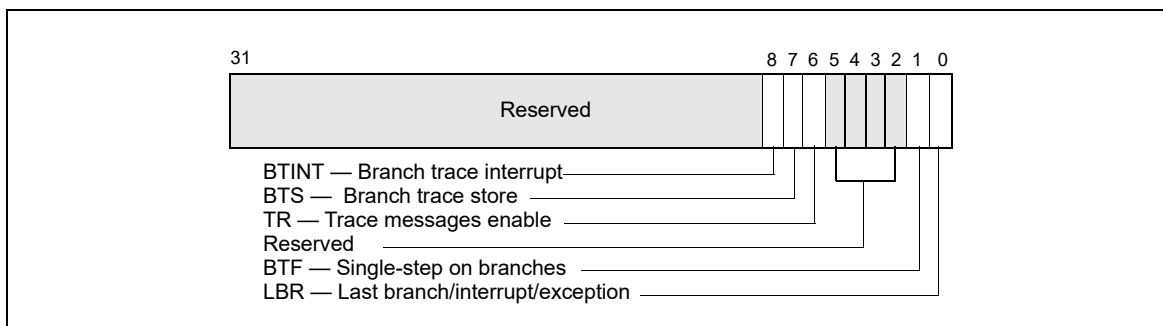


Figure 17-14. IA32_DEBUGCTL MSR for Intel® Core™ Solo and Intel® Core™ Duo Processors

- **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 17.4.5, “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 17-15.

- Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Intel Core Solo and Intel Core Duo processors, this MSR is located at register address 01C9H.

For compatibility, the Intel Core Solo and Intel Core Duo processors provide two 32-bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

For details, see Section 17.12, “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*.

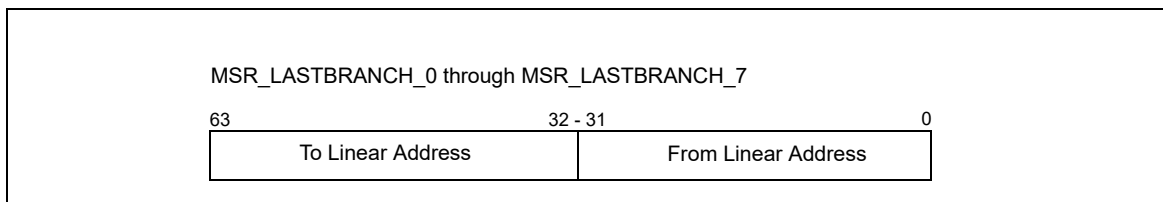


Figure 17-15. LBR Branch Record Layout for the Intel® Core™ Solo and Intel® Core™ Duo Processor

17.15 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 17-16 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 17.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
 - PBi (performance monitoring/breakpoint pins) flags (bits 5-2)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BPi# pin when a breakpoint match occurs. When a PBi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
 - TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 17.4.4, “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 17.4.9, “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 17.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

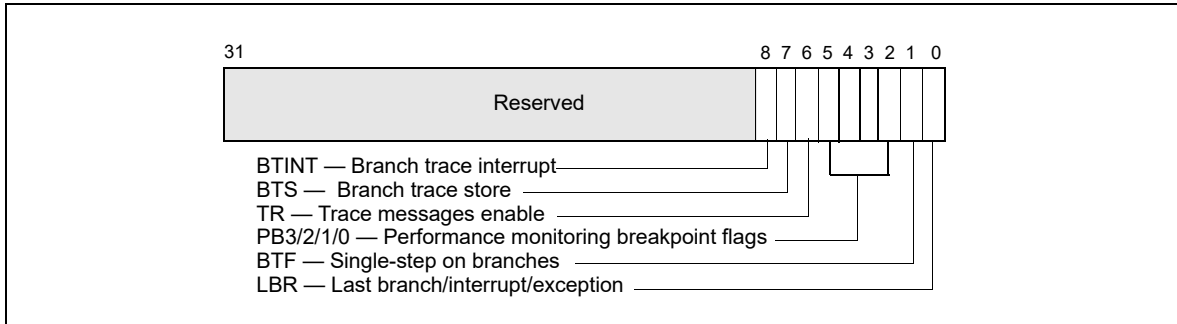


Figure 17-16. MSR_DEBUGCTLB MSR for Pentium M Processors

- **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 17.4.5, “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 17-17.
- **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. For Pentium M Processors, this MSR is located at register address 01C9H.

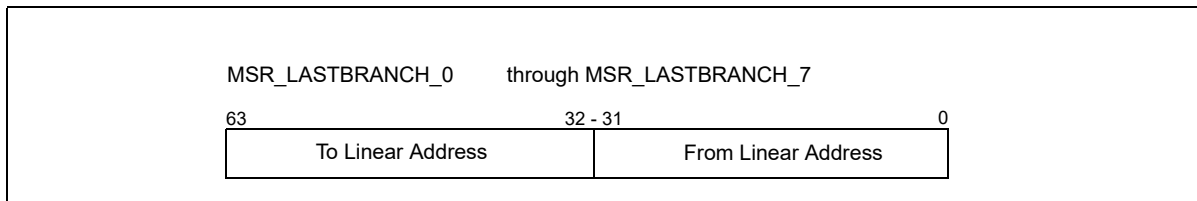


Figure 17-17. LBR Branch Record Layout for the Pentium M Processor

For more detail on these capabilities, see Section 17.13.3, “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*.

17.16 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (P6 FAMILY PROCESSORS)

The P6 family processors provide five MSRs for recording the last branch, interrupt, or exception taken by the processor: DEBUGCTLMR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP. These registers can be used to collect last branch records, to set breakpoints on branches, interrupts, and exceptions, and to single-step from one branch to the next.

See Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4* for a detailed description of each of the last branch recording MSRs.

17.16.1 DEBUGCTLMR Register

The version of the DEBUGCTLMR register found in the P6 family processors enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.

A protected-mode operating system procedure is required to provide user access to this register. Figure 17-18 shows the flags in the DEBUGCTLMR register for the P6 family processors. The functions of these flags are as follows:

- LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records the source and target addresses (in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs) for the last branch and the last exception or interrupt taken by the processor prior to a debug exception being generated. The processor clears this flag whenever a debug exception, such as an instruction or data breakpoint or single-step trap occurs.

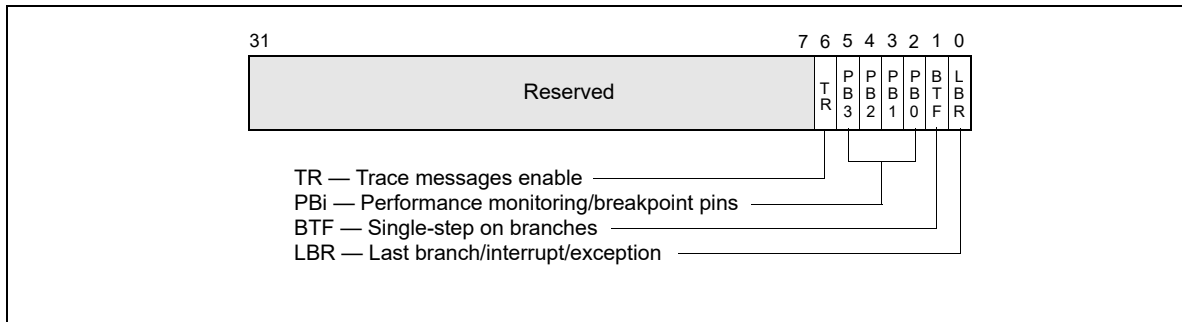


Figure 17-18. DEBUGCTLMR Register (P6 Family Processors)

- BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag. See Section 17.4.3, “Single-Stepping on Branches.”
- PB_i (performance monitoring/breakpoint pins) flags (bits 2 through 5)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding PB_i# pin when a breakpoint match occurs. When a PB_i flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
- TR (trace message enable) flag (bit 6)** — When set, trace messages are enabled as described in Section 17.4.4, “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.

17.16.2 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.

17.16.3 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.

17.17 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 19.7.2 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 19.6.4.5, “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.

17.17.1 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.

17.17.2 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.

17.17.3 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).

17.17.4 Invariant Time-Keeping

The invariant TSC is based on the invariant timekeeping hardware (called Always Running Timer or ART), that runs at the core crystal clock frequency. The ratio defined by CPUID leaf 15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX[InvariantTSC] = 1, the following linearity relationship holds between TSC and the ART hardware:

$$\text{TSC_Value} = (\text{ART_Value} * \text{CPUID.15H:EBX[31:0]}) / \text{CPUID.15H:EAX[31:0]} + K$$

Where 'K' is an offset that can be adjusted by a privileged agent¹.

When ART hardware is reset, both invariant TSC and K are also reset.

17.18 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.

17.18.1 Overview of Cache Monitoring Technology and Memory Bandwidth Monitoring

The shared resource monitoring features described in this chapter provide a layer of abstraction between applications and logical processors through the use of **Resource Monitoring IDs** (RMIDs). Each logical processor in the system can be assigned an RMID independently, or multiple logical processors can be assigned to the same RMID value (e.g., to track an application with multiple threads). For each logical processor, only one RMID value is active at a time. This is enforced by the IA32_PQR_ASSOC MSR, which specifies the active RMID of a logical processor. Writing to this MSR by software changes the active RMID of the logical processor from an old value to a new value.

The underlying platform shared resource monitoring hardware tracks cache metrics such as cache utilization and misses as a result of memory accesses according to the RMIDs and reports monitored data via a counter register (IA32_QM_CTR). The specific event types supported vary by generation and can be enumerated via CPUID. Before reading back monitored data software must configure an event selection MSR (IA32_QM_EVTSEL) to specify which metric is to be reported, and the specific RMID for which the data should be returned.

Processor support of the monitoring framework and sub-features such as CMT is reported via the CPUID instruction. The resource type available to the monitoring framework is enumerated via a new leaf function in CPUID. Reading and writing to the monitoring MSRs requires the RDMSR and WRMSR instructions.

The Cache Monitoring Technology feature set provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the CMT feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- CMT-specific event codes to read occupancy for a given level of the cache hierarchy.

The Memory Bandwidth Monitoring feature provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the MBM feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- MBM-specific event codes to read bandwidth out to the next level of the hierarchy and various sub-event codes to read more specific metrics as discussed later (e.g., total bandwidth vs. bandwidth only from local memory controllers on the same package).

17.18.2 Enabling Monitoring: Usage Flow

Figure 17-19 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.

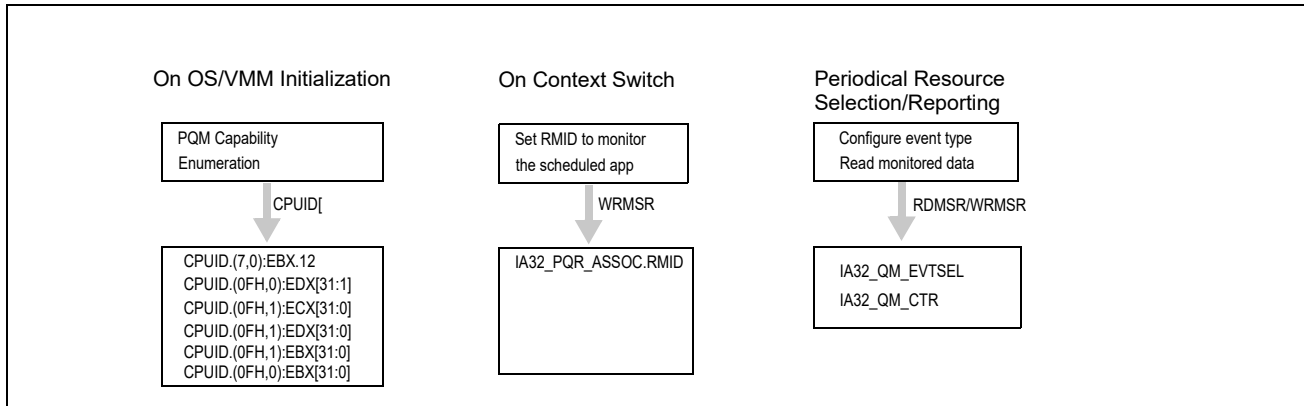


Figure 17-19. Platform Shared Resource Monitoring Usage Flow

17.18.3 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 17.18.4), and monitoring capabilities for each resource type (see Section 17.18.5). 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 17.18.6.
- 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 17.18.7.
- 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.

17.18.4 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 17-21 and Figure 17-22). 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.

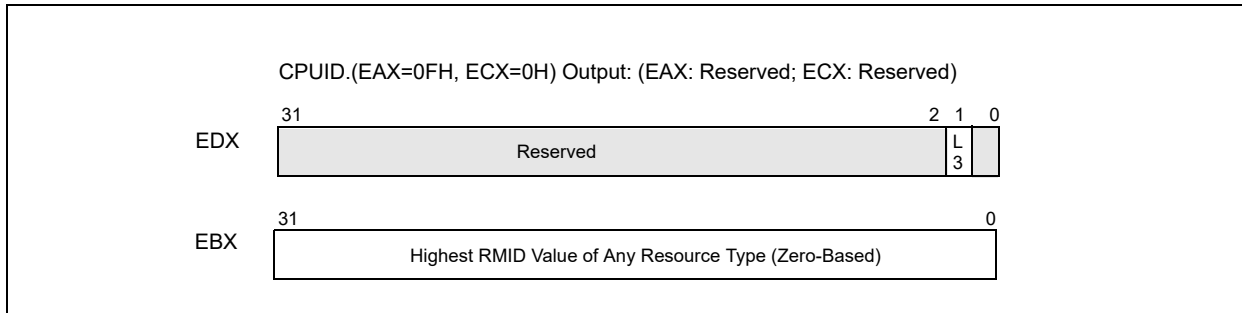


Figure 17-20. CPUID.(EAX=0FH, ECX=0H) Monitoring Resource Type Enumeration

17.18.5 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.

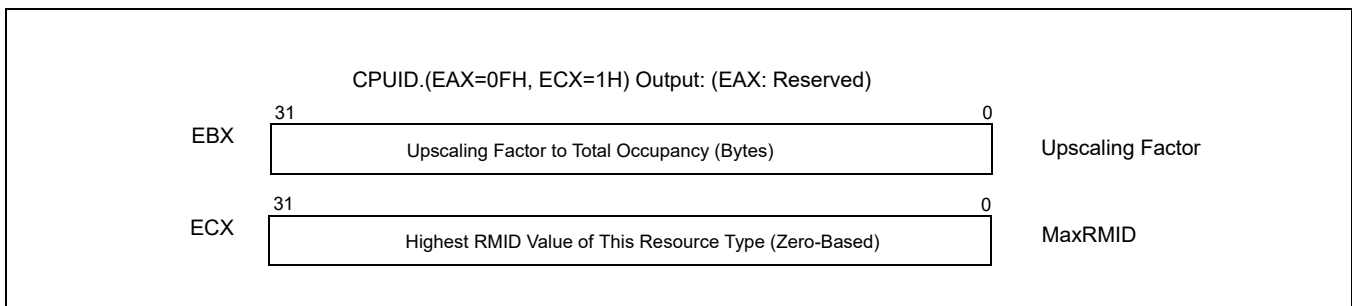


Figure 17-21. L3 Cache Monitoring Capability Enumeration Data (CPUID.(EAX=0FH, ECX=1H))

For each supported Cache Monitoring resource type, hardware supports only a finite number of RMIDs. CPUID.(EAX=0FH, ECX=1H).ECX enumerates the highest RMID value that can be monitored with this resource type, see Figure 17-21.

CPUID.(EAX=0FH, ECX=1H).EDX specifies a bit vector that is used to look up the EventID (See Figure 17-22 and Table 17-18) 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 17-21.

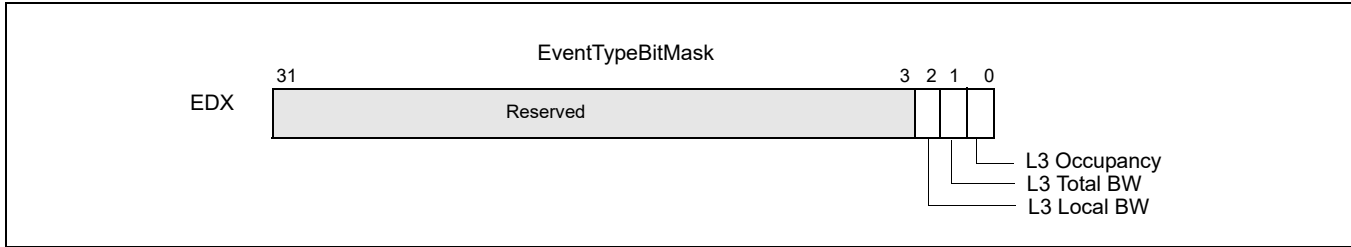


Figure 17-22. L3 Cache Monitoring Capability Enumeration Event Type Bit Vector (CPUID.(EAX=0FH, ECX=1H))

17.18.5.1 Cache Monitoring Technology

On processors for which Cache Monitoring Technology supports the L3 cache occupancy event, CPUID.(EAX=0FH, ECX=1H).EDX would return with only bit 0 set. The corresponding event ID can be looked up from Table 17-18. The L3 occupancy data accumulated in IA32_QM_CTR can be converted to total occupancy (in bytes) by multiplying with CPUID.(EAX=0FH, ECX=1H).EBX.

Event codes for Cache Monitoring Technology are discussed in the next section.

17.18.5.2 Memory Bandwidth Monitoring

On processors that monitoring supports Memory Bandwidth Monitoring using ResID=1 (L3), two additional bits will be set in the vector at CPUID.(EAX=0FH, ECX=1H).EDX:

- CPUID.(EAX=0FH, ECX=1H).EDX[bit 1]: indicates the L3 total external bandwidth monitoring event is supported if set. This event monitors the L3 total external bandwidth to the next level of the cache hierarchy, including all demand and prefetch misses from the L3 to the next hierarchy of the memory system. In most platforms, this represents memory bandwidth.
- CPUID.(EAX=0FH, ECX=1H).EDX[bit 2]: indicates L3 local memory bandwidth monitoring event is supported if set. This event monitors the L3 external bandwidth satisfied by the local memory. In most platforms that support this event, L3 requests are likely serviced by a memory system with non-uniform memory architecture. This allows bandwidth to off-package memory resources to be tracked by subtracting local from total bandwidth (for instance, bandwidth over QPI to a memory controller on another physical processor could be tracked by subtraction).

The corresponding Event ID can be looked up from Table 17-18. 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 17-18. 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

17.18.6 Monitoring Resource RMID Association

After Monitoring and sub-features has been enumerated, software can begin using the monitoring features. The first step is to associate a given software thread (or multiple threads as part of an application, VM, group of applications or other abstraction) with an RMID.

Note that the process of associating an RMID with a given software thread is the same for all shared resource monitoring features (CMT, MBM), and a given RMID number has the same meaning from the viewpoint of any logical processors in a package. Stated another way, a thread may be associated in a 1:1 mapping with an RMID, and that

RMID may allow cache occupancy, memory bandwidth information or other monitoring data to be read back later with monitoring event codes (retrieving data is discussed in a previous section).

The association of an application thread with an RMID requires an OS to program the per-logical-processor MSR IA32_PQR_ASSOC at context swap time (updates may also be made at any other arbitrary points during program execution such as application phase changes). The IA32_PQR_ASSOC MSR specifies the active RMID that monitoring hardware will use to tag internal operations, such as L3 cache requests. The layout of the MSR is shown in Figure 17-23. Software specifies the active RMID to monitor in the IA32_PQR_ASSOC.RMID field. The width of the RMID field can vary from one implementation to another, and is derived from Ceil ($\log_2(1 + \text{CPUID}(\text{EAX}=0\text{FH}, \text{ECX}=0):\text{EBX}[31:0])$). The value of IA32_PQR_ASSOC after power-on is 0.

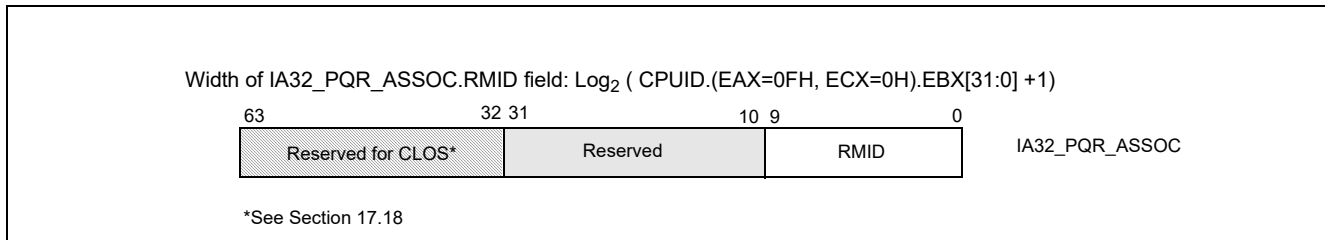


Figure 17-23. IA32_PQR_ASSOC MSR

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.

17.18.7 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 17-24. Bits IA32_QM_EVTSEL.EvtID (bits 7:0) specify an event code of a supported resource type for hardware to report monitored data associated with IA32_QM_EVTSEL.RMID (bits 41:32). Software can configure IA32_QM_EVTSEL.RMID with any RMID that is active within the physical processor. The width of IA32_QM_EVTSEL.RMID matches that of IA32_PQR_ASSOC.RMID. Supported event codes for the IA32_QM_EVTSEL register are shown in Table 17-18. 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 17.18.5, which covers feature-specific details.

Thread access to the IA32_QM_EVTSEL and IA32_QM_CTR MSR pair should be serialized (that is, treated as a critical section under lock) to avoid situations where one thread changes the RMID/EvtID just before another thread reads monitoring data from IA32_QM_CTR.

- IA32_QM_CTR: This MSR reports monitored data when available. It contains three bit fields. If software configures an unsupported RMID or event type in IA32_QM_EVTSEL, then IA32_QM_CTR.Error (bit 63) will be set, indicating there is no valid data to report. If IA32_QM_CTR.Unavailable (bit 62) is set, it indicates monitored data for the RMID is not available, and IA32_QM_CTR.data (bits 61:0) should be ignored. Therefore, IA32_QM_CTR.data (bits 61:0) is valid only if bit 63 and 62 are both clear. For Cache Monitoring Technology, software can convert IA32_QM_CTR.data into cache occupancy or bandwidth metrics expressed in bytes by multiplying with the conversion factor from CPUID.(EAX=0FH, ECX=1H).EBX.

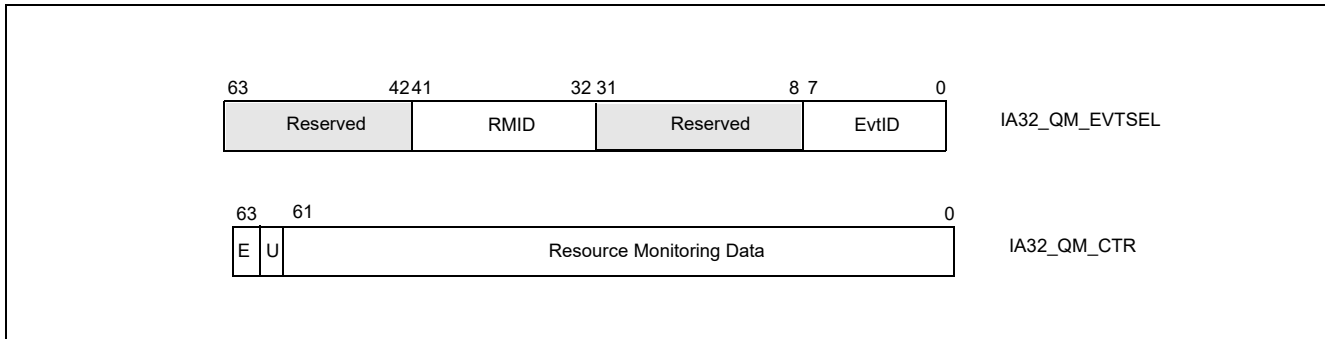


Figure 17-24. IA32_QM_EVTSEL and IA32_QM_CTR MSRs

17.18.8 Monitoring Programming Considerations

Figure 17-23 illustrates how system software can program IA32_QOSEVTSEL and IA32_QM_CTR to perform resource monitoring.

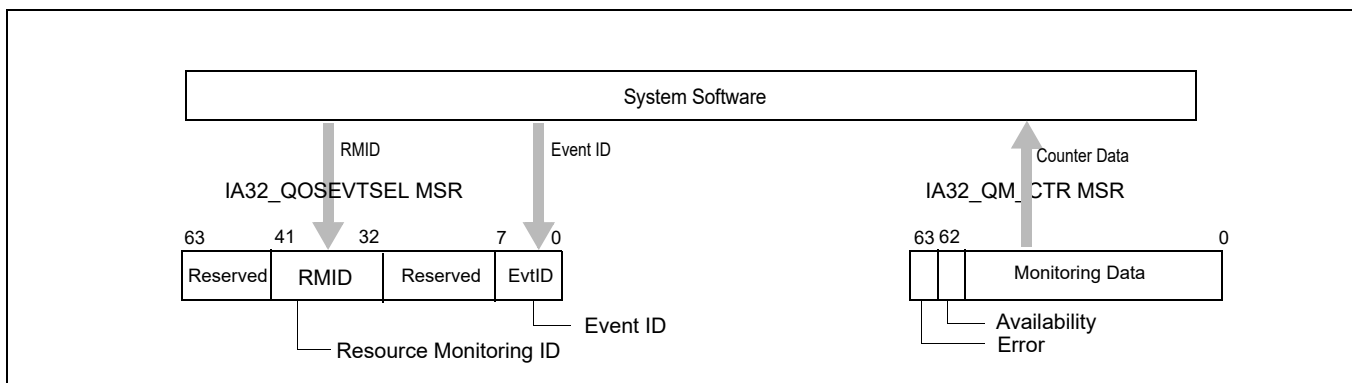


Figure 17-25. Software Usage of Cache Monitoring Resources

Though the field provided in IA32_QM_CTR allows for up to 62 bits of data to be returned, often a subset of bits are used. With Cache Monitoring Technology for instance, the number of bits used will be proportional to the base-two logarithm of the total cache size divided by the Upscaling Factor from CPUID.

In Memory Bandwidth Monitoring the initial counter size is 24 bits, and retrieving the value at 1Hz or faster is sufficient to ensure at most one rollover per sampling period. Any future changes to counter width will be enumerated to software.

17.18.8.1 Monitoring Dynamic Configuration

Both the IA32_QM_EVTSEL and IA32_PQR_ASSOC registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,
- An RMID exceeding the maxRMID is used.

17.18.8.2 Monitoring Operation With Power Saving Features

Note that some advanced power management features such as deep package C-states may shrink the L3 cache and cause CMT occupancy count to be reduced. MBM bandwidth counts may increase due to flushing cached data out of L3.

17.18.8.3 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.

17.18.8.4 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.

17.19 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 Processor Scalable 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.

17.19.1 Introduction to Cache Allocation Technology (CAT)

Cache Allocation Technology enables an Operating System (OS), Hypervisor /Virtual Machine Manager (VMM) or similar system service management agent to specify the amount of cache space into which an application can fill (as a hint to hardware - certain features such as power management may override CAT settings). Specialized user-level implementations with minimal OS support are also possible, though not necessarily recommended (see notes below for OS/Hypervisor with respect to ring 3 software and virtual guests). Depending on the processor family, L2 or L3 cache allocation capability may be provided, and the technology is designed to scale across multiple cache levels and technology generations.

Software can determine which levels are supported in a given platform programmatically using CPUID as described in the following sections.

The CAT mechanisms defined in this document provide the following key features:

- A mechanism to enumerate platform Cache Allocation Technology capabilities and available resource types that provides CAT control capabilities. For implementations that support Cache Allocation Technology, CPUID provides enumeration support to query which levels of the cache hierarchy are supported and specific CAT capabilities, such as the max allocation bitmask size,
- A mechanism for the OS or Hypervisor to configure the amount of a resource available to a particular Class of Service via a list of allocation bitmasks,
- Mechanisms for the OS or Hypervisor to signal the Class of Service to which an application belongs, and
- Hardware mechanisms to guide the LLC fill policy when an application has been designated to belong to a specific Class of Service.

Note that for many usages, an OS or Hypervisor may not want to expose Cache Allocation Technology mechanisms to Ring3 software or virtualized guests.

The Cache Allocation Technology feature enables more cache resources (i.e., cache space) to be made available for high priority applications based on guidance from the execution environment as shown in Figure 17-26. 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 17-26 the L3 Cache is shown as an example resource.

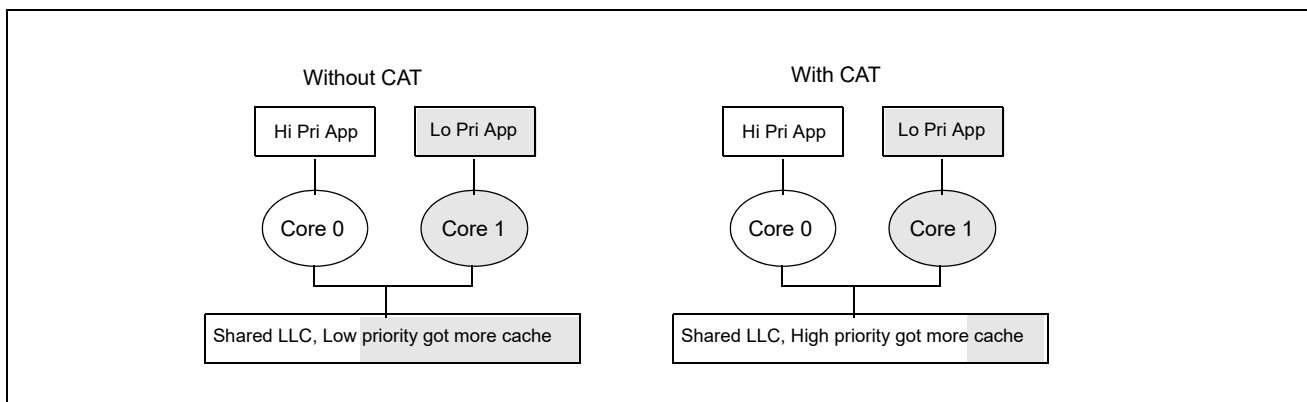


Figure 17-26. Cache Allocation Technology Enables Allocation of More Resources to High Priority Applications

17.19.2 Cache Allocation Technology Architecture

The fundamental goal of Cache Allocation Technology is to enable resource allocation based on application priority or Class of Service (COS or CLOS). The processor exposes a set of Classes of Service into which applications (or individual threads) can be assigned. Cache allocation for the respective applications or threads is then restricted based on the class with which they are associated. Each Class of Service can be configured using capacity bitmasks (CBMs) which represent capacity and indicate the degree of overlap and isolation between classes. For each logical processor there is a register exposed (referred to here as the IA32_PQR_ASSOC MSR or PQR) to allow the OS/VMM to specify a COS when an application, thread or VM is scheduled.

The usage of Classes of Service (COS) are consistent across resources and a COS may have multiple resource control attributes attached, which reduces software overhead at context swap time. Rather than adding new types of COS tags per resource for instance, the COS management overhead is constant. Cache allocation for the indicated application/thread/container/VM is then controlled automatically by the hardware based on the class and the bitmask associated with that class. Bitmasks are configured via the IA32_resourceType_MASK_n MSRs, where resourceType indicates a resource type (e.g., "L3" for the L3 cache) and "n" indicates a COS number.

The basic ingredients of Cache Allocation Technology are as follows:

- An architecturally exposed mechanism using CPUID to indicate whether CAT is supported, and what resource types are available which can be controlled,
- For each available resourceType, CPUID also enumerates the total number of Classes of Services and the length of the capacity bitmasks that can be used to enforce cache allocation to applications on the platform,
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to configure the behavior of different classes of service using the bitmasks available,
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to assign a COS to an executing software thread (i.e., associating the active CR3 of a logical processor with the COS in IA32_PQR_ASSOC),
- Implementation-dependent mechanisms to indicate which COS is associated with a memory access and to enforce the cache allocation on a per COS basis.

A capacity bitmask (CBM) provides a hint to the hardware indicating the cache space an application should be limited to as well as providing an indication of overlap and isolation in the CAT-capable cache from other applications contending for the cache. The bit length of the capacity mask available generally depends on the configuration of the cache and is specified in the enumeration process for CAT in CPUID (this may vary between models in a processor family as well). Similarly, other parameters such as the number of supported COS may vary for each resource type, and these details can be enumerated via CPUID.

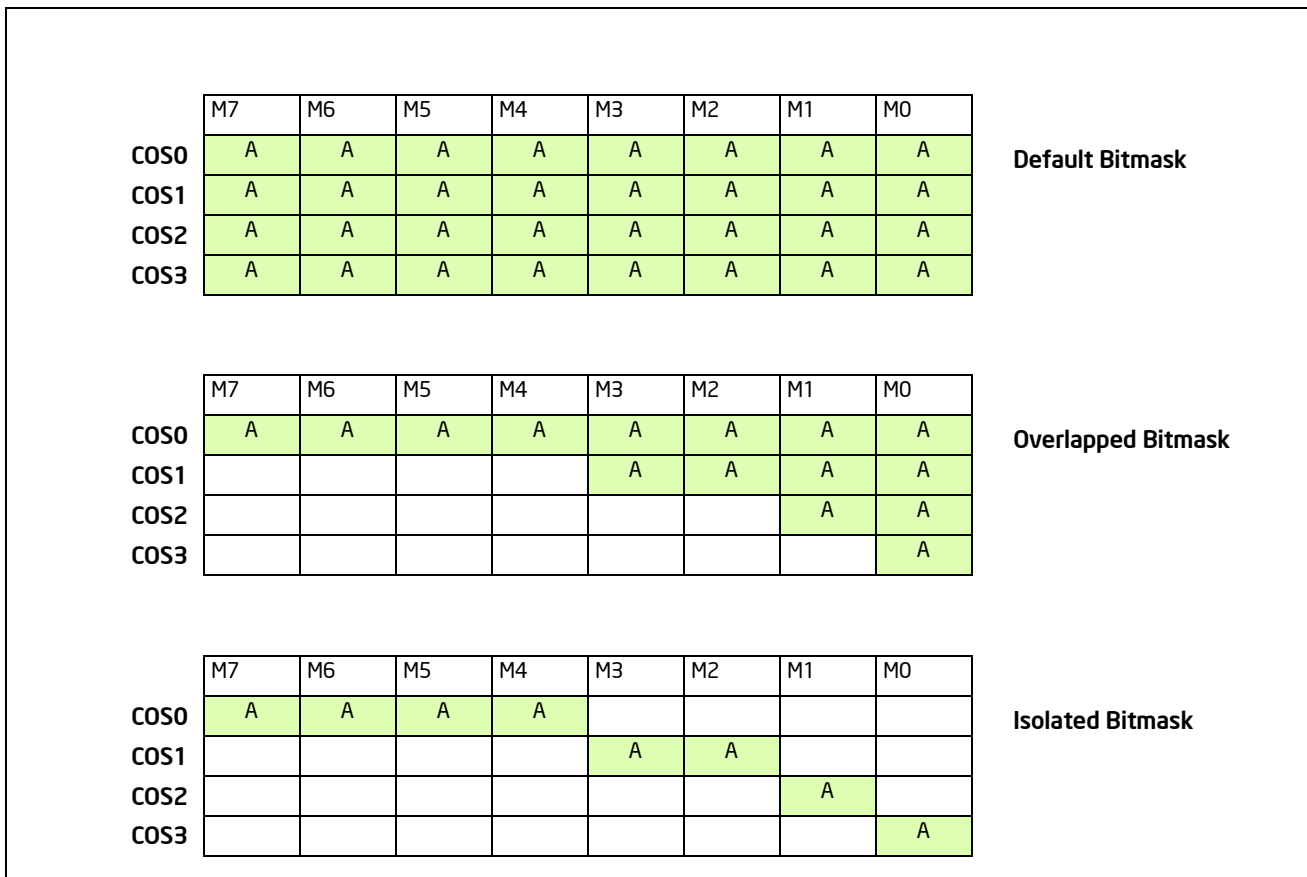


Figure 17-27. Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bit length of 8 are shown in Figure 17-27. Please note that all (and only) contiguous '1' combinations are allowed (e.g., FFFFH, 0FF0H, 003CH, etc.). Attempts to program a value without contiguous '1's (including zero) will result in a general protection fault (#GP(0)). It is generally expected that in way-based implementations, one capacity mask bit corresponds to some number of ways in cache, but the specific mapping is implementation-dependent. In all cases, a mask bit set to '1' specifies that a particular Class of Service can allocate into the cache subset represented by that bit. A value of '0' in a mask bit specifies that a Class of

Service cannot allocate into the given cache subset. In general, allocating more cache to a given application is usually beneficial to its performance.

Figure 17-27 also shows three examples of sets of Cache Capacity Bitmasks. For simplicity these are represented as 8-bit vectors, though this may vary depending on the implementation and how the mask is mapped to the available cache capacity. The first example shows the default case where all 4 Classes of Service (the total number of COS are implementation-dependent) have full access to the cache. The second case shows an overlapped case, which would allow some lower-priority threads share cache space with the highest priority threads. The third case shows various non-overlapped partitioning schemes. As a matter of software policy for extensibility COS0 should typically be considered and configured as the highest priority COS, followed by COS1, and so on, though there is no hardware restriction enforcing this mapping. When the system boots all threads are initialized to COS0, which has full access to the cache by default.

Though the representation of the CBMs looks similar to a way-based mapping they are independent of any specific enforcement implementation (e.g., way partitioning.) Rather, this is a convenient manner to represent capacity, overlap and isolation of cache space. For example, executing a *POPCNT* instruction (population count of set bits) on the capacity bitmask can provide the fraction of cache space that a class of service can allocate into. In addition to the fraction, the exact location of the bits also shows whether the class of service overlaps with other classes of service or is entirely isolated in terms of cache space used.

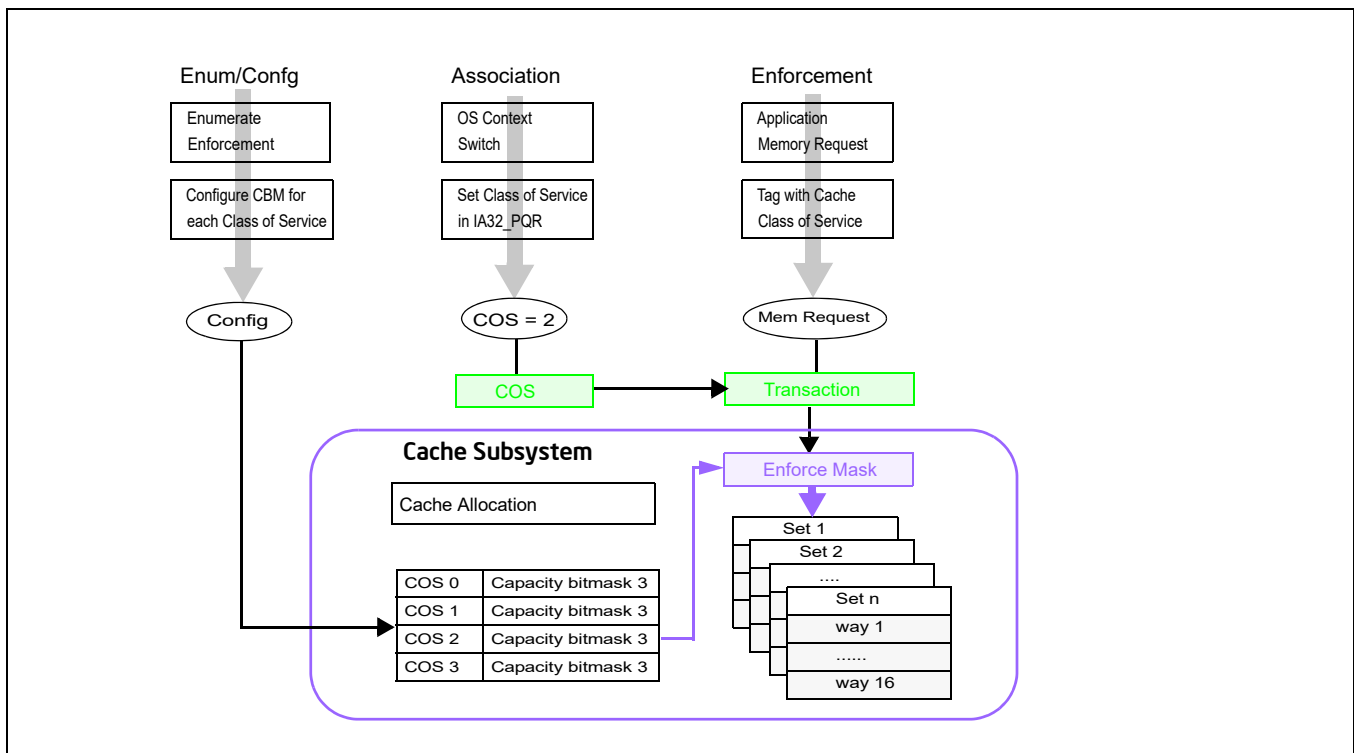


Figure 17-28. Class of Service and Cache Capacity Bitmasks

Figure 17-28 shows how the Cache Capacity Bitmasks and the per-logical-processor Class of Service are logically used to enable Cache Allocation Technology. All (and only) contiguous 1's in the CBM are permitted. The length of a CBM may vary from resource to resource or between processor generations and can be enumerated using CPUID. From the available mask set and based on the goals of the OS/VMM (shared or isolated cache, etc.) bitmasks are selected and associated with different classes of service. For the available Classes of Service the associated CBMs can be programmed via the global set of CAT configuration registers (in the case of L3 CAT, via the IA32_L3_MASK_n MSRs, where "n" is the Class of Service, starting from zero). In all architectural implementations supporting CPUID it is possible to change the CBMs dynamically, during program execution, unless stated otherwise by Intel.

The currently running application's Class of Service is communicated to the hardware through the per-logical-processor PQR MSR (IA32_PQR_ASSOC MSR). When the OS schedules an application thread on a logical processor,

the application thread is associated with a specific COS (i.e., the corresponding COS in the PQR) and all requests to the CAT-capable resource from that logical processor are tagged with that COS (in other words, the application thread is configured to belong to a specific COS). The cache subsystem uses this tagged request information to enforce QoS. The capacity bitmask may be mapped into a way bitmask (or a similar enforcement entity based on the implementation) at the cache before it is applied to the allocation policy. For example, the capacity bitmask can be an 8-bit mask and the enforcement may be accomplished using a 16-way bitmask for a cache enforcement implementation based on way partitioning.

The following sections describe extensions of CAT such as Code and Data Prioritization (CDP), followed by details on specific features such as L3 CAT, L3 CDP, L2 CAT, and L2 CDP. Depending on the specific processor a mix of features may be supported, and CPUID provides enumeration capabilities to enable software to dynamically detect the set of supported features.

17.19.3 Code and Data Prioritization (CDP) Technology

Code and Data Prioritization Technology is an extension of CAT. CDP enables isolation and separate prioritization of code and data fetches to the L2 or L3 cache in a software configurable manner, depending on hardware support, which can enable workload prioritization and tuning of cache capacity to the characteristics of the workload. CDP extends Cache Allocation Technology (CAT) by providing separate code and data masks per Class of Service (COS). Support for the L2 CDP feature and the L3 CDP features are separately enumerated (via CPUID) and separately controlled (via remapping the L2 CAT MSRs or L3 CAT MSRs respectively). Section 17.19.6.3 and Section 17.19.7 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 17-29),
- the range of COS for CAT is re-indexed, with the lower-half of the COS range available for CDP.

Using the CDP feature, virtual isolation between code and data can be configured on the L2 or L3 cache if desired, similar to how some processor cache levels provide separate L1 data and L1 instruction caches.

Like the CAT feature, CDP may be dynamically configured by privileged software at any point during normal system operation, including dynamically enabling or disabling the feature provided that certain software configuration requirements are met (see Section 17.19.5).

An example of the operating mode of CDP is shown in Figure 17-29. Shown at the top are traditional CAT usage models where capacity masks map 1:1 with a COS number to enable control over the cache space which a given COS (and thus applications, threads or VMs) may occupy. Shown at the bottom are example mask configurations where CDP is enabled, and each COS number maps 1:2 to two masks, one for code and one for data. This enables code and data to be either overlapped or isolated to varying degrees either globally or on a per-COS basis, depending on application and system needs.

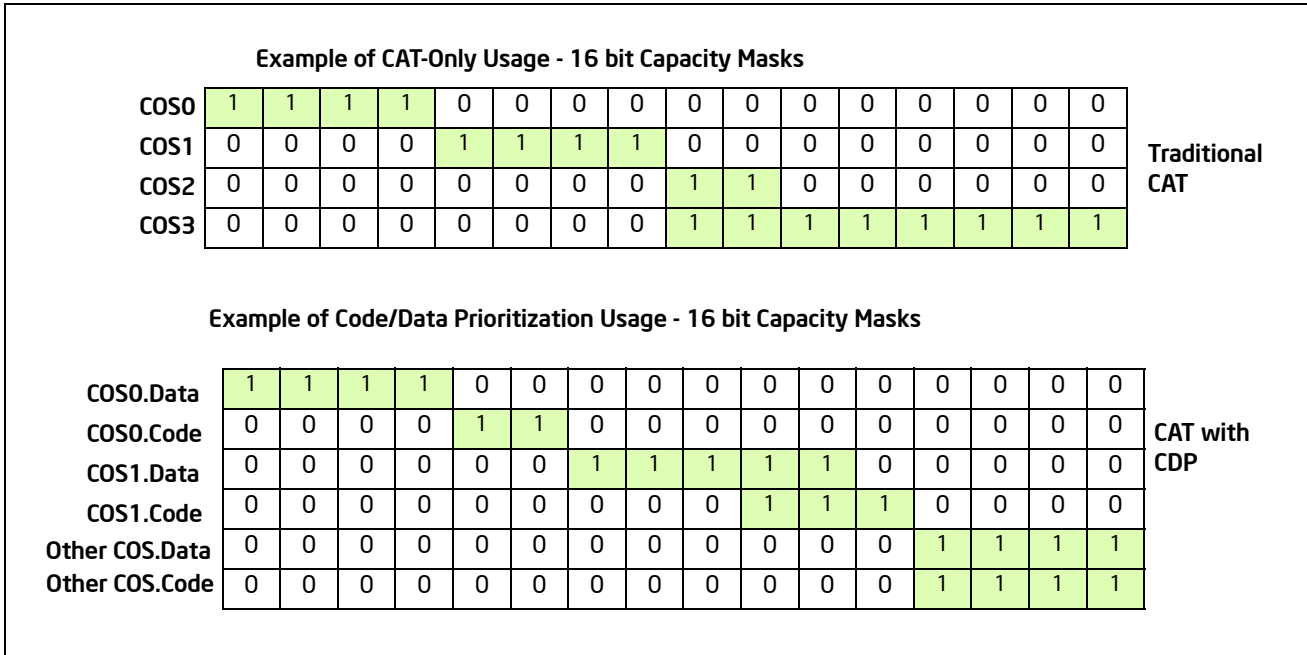


Figure 17-29. Code and Data Capacity Bitmasks of CDP

When CDP is enabled, the existing mask space for CAT-only operation is split. As an example if the system supports 16 CAT-only COS, when CDP is enabled the same MSR interfaces are used, however half of the masks correspond to code, half correspond to data, and the effective number of COS is reduced by half. Code/Data masks are defined per-COS and interleaved in the MSR space as described in subsequent sections.

In cases where CPUID exposes a non-even number of supported Classes of Service for the CAT or CDP features, software using CDP should use the lower matched pairs of code/data masks, and any upper unpaired masks should not be used. As an example, if CPUID exposes 5 CLOS, when CDP is enabled then two code/data pairs are available (masks 0/1 for CLOS[0] data/code and masks 2/3 for CLOS[1] data/code), however the upper un-paired mask should not be used (mask 4 in this case) or undefined behavior may result.

17.19.4 Enabling Cache Allocation Technology Usage Flow

Figure 17-30 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.

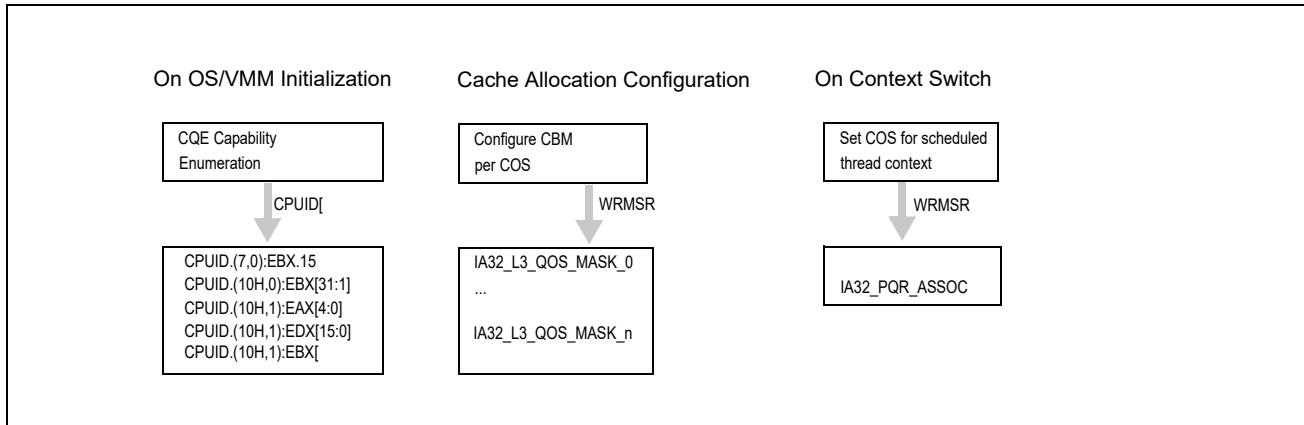


Figure 17-30. Cache Allocation Technology Usage Flow

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.

17.19.4.1 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 17.19.4.2).
- IA32_L3_MASK_n: A range of MSRs is provided for each resource type, each MSR within that range specifying a software-configured capacity bitmask for each class of service. For L3 with Cache Allocation support, the CBM is specified using one of the IA32_L3_QOS_MASK_n MSR, where 'n' corresponds to a number within the supported range of COS, i.e., the range between 0 and CPUID.(EAX=10H, ECX=ResID):EDX[15:0], inclusive. See Section 17.19.4.3 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 17.19.4.3 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 15.5.2, "System Software Recommendation for Managing CMC and Machine Check Resources". Software may also use CPUID leaf 4 to determine the maximum number of logical processor IDs that may share a given level of the cache.

- IA32_PQR_ASSOC.CLOS: The IA32_PQR_ASSOC MSR provides a COS field that OS/VMM can use to assign a logical processor to an available COS. The set of COS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software COS management overhead at context swap time. See Section 17.19.4.4 for details.

17.19.4.2 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 17-31).

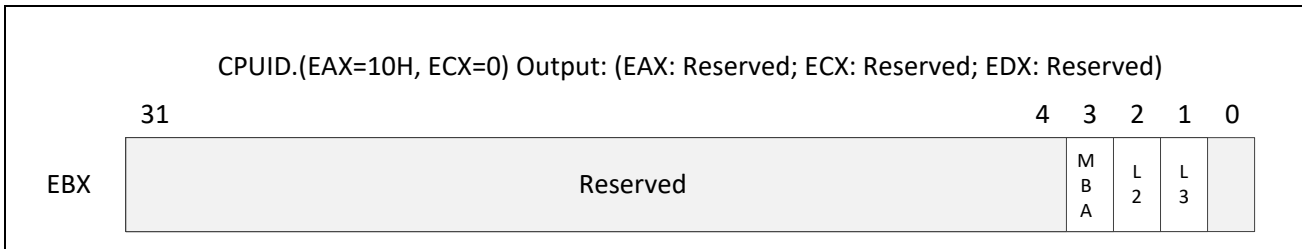


Figure 17-31. CPUID.(EAX=10H, ECX=0H) Available Resource Type Identification

- For ECX>0, EAX[4:0] reports the length of the capacity bitmask (ECX=1 or 2 for L3 CAT or L2 CAT respectively) using minus-one notation, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- Sub-functions of CPUID.EAX=10H with a non-zero ECX input matching a supported ResID enumerate the specific enforcement details of the corresponding ResID. The capabilities enumerated include the length of the capacity bitmasks and the number of Classes of Service for a given ResID. Software should query the capability of each available ResID that supports CAT from a sub-leaf of leaf 10H using the sub-leaf index reported by the corresponding non-zero bit in CPUID.(EAX=10H, ECX=0):EBX[31:1] in order to obtain additional feature details.
- CAT capability for L3 is enumerated by CPUID.(EAX=10H, ECX=1H), see Figure 17-32. The specific CAT capabilities reported by CPUID.(EAX=10H, ECX=1) are:

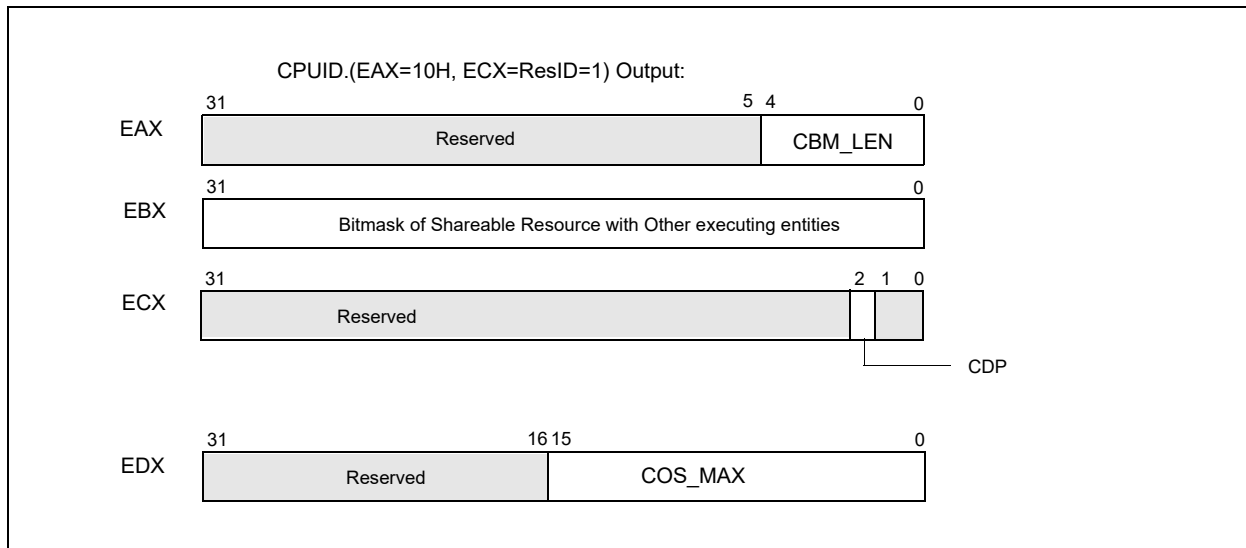


Figure 17-32. L3 Cache Allocation Technology and CDP Enumeration

- CPUID.(EAX=10H, ECX=ResID=1):EAX[4:0] reports the length of the capacity bitmask using minus-one notation, e.g., a value of 15 corresponds to the capability bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- CPUID.(EAX=10H, ECX=1):EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L3 allocation may be used by other entities in the platform (e.g., an

integrated graphics engine or hardware units outside the processor core and have direct access to L3). Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.

- CPUID.(EAX=10H, ECX=1):ECX.CDP[bit 2]: If 1, indicates L3 Code and Data Prioritization Technology is supported (see Section 17.19.5). Other bits of CPUID.(EAX=10H, ECX=1):ECX are reserved.
- CPUID.(EAX=10H, ECX=1):EDX[15:0] reports the maximum COS supported for the resource (COS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported COS). Bits 31:16 are reserved.
- CAT capability for L2 is enumerated by CPUID.(EAX=10H, ECX=2H), see Figure 17-33. The specific CAT capabilities reported by CPUID.(EAX=10H, ECX=2) are:

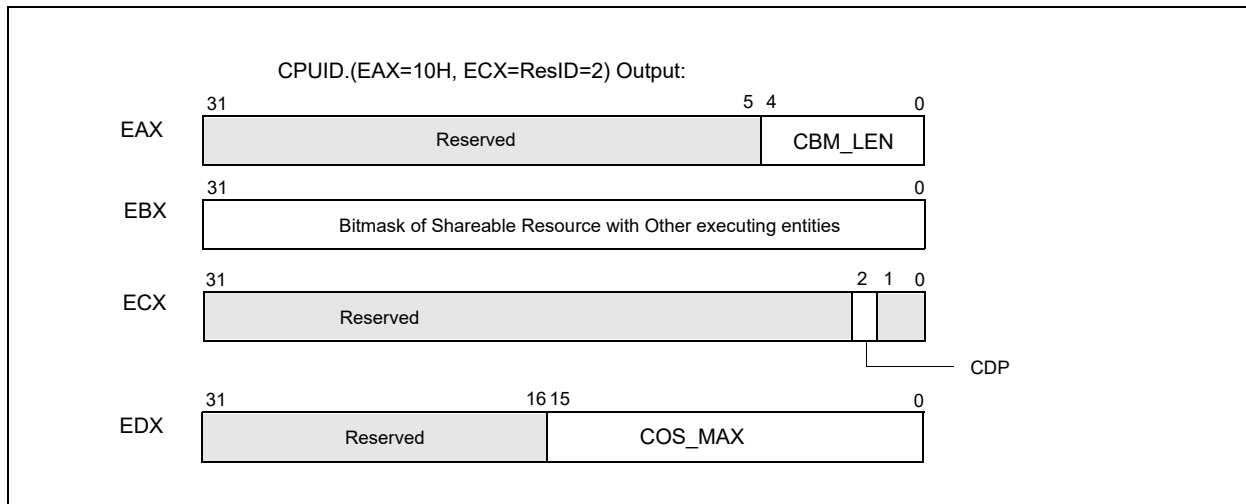


Figure 17-33. L2 Cache Allocation Technology

- CPUID.(EAX=10H, ECX=ResID=2):EAX[4:0] reports the length of the capacity bitmask using minus-one notation, e.g., a value of 15 corresponds to the capability bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- CPUID.(EAX=10H, ECX=2):EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L2 allocation may be used by other entities in the platform. Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
- CPUID.(EAX=10H, ECX=2):ECX.CDP[bit 2]: If 1, indicates L2 Code and Data Prioritization Technology is supported (see Section 17.19.6). Other bits of CPUID.(EAX=10H, ECX=2):ECX are reserved.
- CPUID.(EAX=10H, ECX=2):EDX[15:0] reports the maximum COS supported for the resource (COS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported COS). Bits 31:16 are reserved.

A note on migration of Classes of Service (COS): Software should minimize migrations of COS across logical processors (across threads or cores), as a reduction in the performance of the Cache Allocation Technology feature may result if COS are migrated frequently. This is aligned with the industry-standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and COS migration across processor logical threads and processor cores.

17.19.4.3 Cache Allocation Technology: Cache Mask Configuration

After determining the length of the capacity bitmasks (CBM) and number of COS supported using CPUID (see Section 17.19.4.2), each COS needs to be programmed with a CBM to dictate its available cache via a write to the corresponding IA32_resourceType_MASK_n register, where 'n' corresponds to a number within the supported range of COS, i.e., the range between 0 and CPUID.(EAX=10H, ECX=ResID):EDX[15:0], inclusive, and 'resourceType' corresponds to a specific resource as enumerated by the set bits of CPUID.(EAX=10H, ECX=0):EAX[31:1], for instance, 'L2' or 'L3' cache.

A hierarchy of MSR is reserved for Cache Allocation Technology registers of the form IA32_resourceType_MASK_n:

- From 0C90H through 0D8FH (inclusive), providing support for multiple sub-ranges to support varying resource types. The first supported resourceType is 'L3', corresponding to the L3 cache in a platform. The MSRs range from 0C90H through 0D0FH (inclusive), enables support for up to 128 L3 CAT Classes of Service.

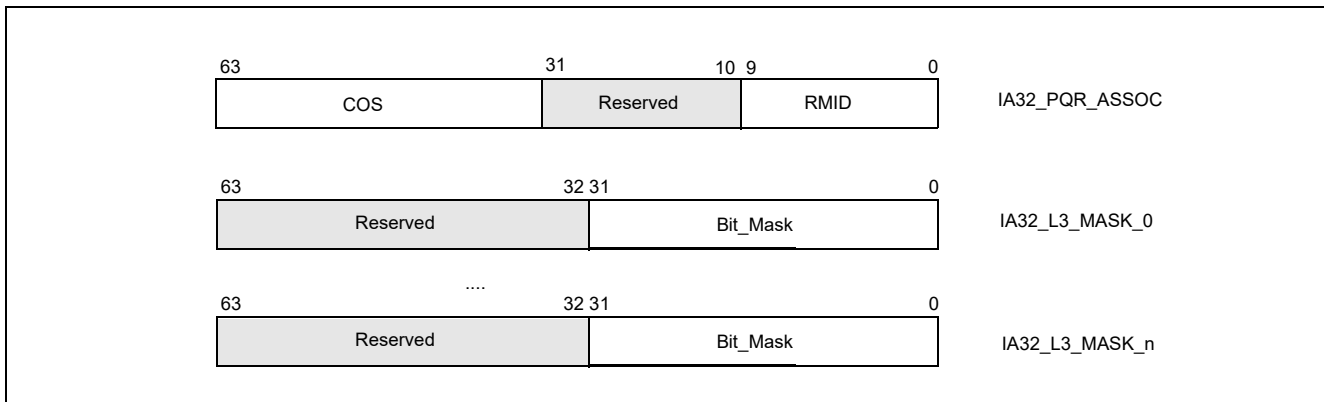


Figure 17-34. IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- 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 17-34 and Figure 17-35 provide an overview of the relevant registers.

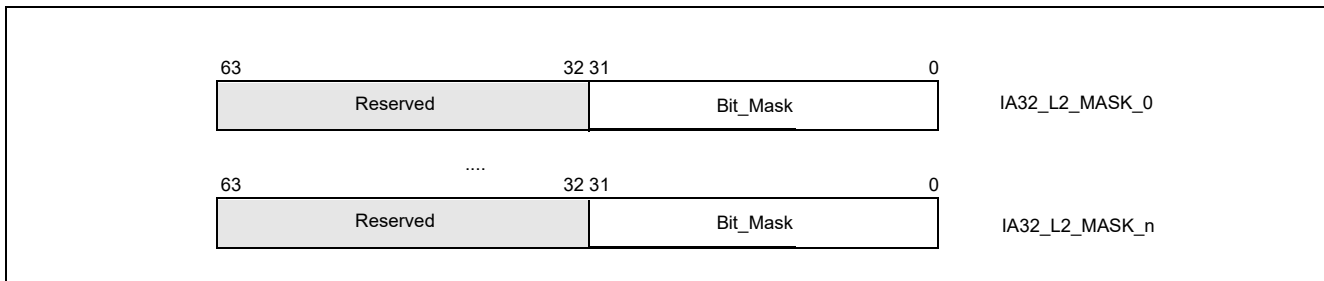


Figure 17-35. IA32_L2_MASK_n MSRs

All CAT configuration registers can be accessed using the standard RDMSR / WRMSR instructions.

Note that once L3 or L2 CAT masks are configured, threads can be grouped into Classes of Service (COS) using the IA32_PQR_ASSOC MSR as described in Chapter 17, "Class of Service to Cache Mask Association: Common Across Allocation Features".

17.19.4.4 Class of Service to Cache Mask Association: Common Across Allocation Features

After configuring the available classes of service with the preferred set of capacity bitmasks, the OS/VMM can set the IA32_PQR_ASSOC.COS of a logical processor to the class of service with the desired CBM when a thread

context switch occurs. This allows the OS/VMM to indicate which class of service an executing thread/VM belongs within. Each logical processor contains an instance of the IA32_PQR_ASSOC register at MSR location 0C8FH, and Figure 17-34 shows the bit field layout for this register. Bits[63:32] contain the COS field for each logical processor.

Note that placing the RMID field within the same PQR register enables both RMID and CLOS to be swapped at context swap time for simultaneous use of monitoring and allocation features with a single register write for efficiency.

When CDP is enabled, Specifying a COS value in IA32_PQR_ASSOC.COS greater than MAX_COS_CDP = (CPUID.(EAX=10H, ECX=1):EDX[15:0] >> 1) will cause undefined performance impact to code and data fetches. In all cases, code and data masks for L2 and L3 CDP should be programmed with at least one bit set.

Note that if the IA32_PQR_ASSOC.COS is never written then the CAT capability defaults to using COS 0, which in turn is set to the default mask in IA32_L3_MASK_0 - which is all "1"s (on reset). This essentially disables the enforcement feature by default or for legacy operating systems and software.

See Section 17.19.7, "Introduction to Memory Bandwidth Allocation" for important COS programming considerations including maximum values when using CAT and CDP.

17.19.5 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 17-32). Most of the CPUID.(EAX=10H, ECX=1) sub-leaf data that applies to CAT also apply to CDP. However, CPUID.(EAX=10H, ECX=1):EDX.COS_MAX_CAT specifies the maximum COS applicable to CAT-only operation. For CDP operations, COS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=1):EDX.COS_MAX_CAT >> 1).

If CPUID.(EAX=10H, ECX=1):ECX.CDP[bit 2] = 1, the processor supports CDP and provides a new MSR IA32_L3_QOS_CFG at address 0C81H. The layout of IA32_L3_QOS_CFG is shown in Figure 17-36. The bit field definition of IA32_L3_QOS_CFG are:

- Bit 0: L3 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.COS is COS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

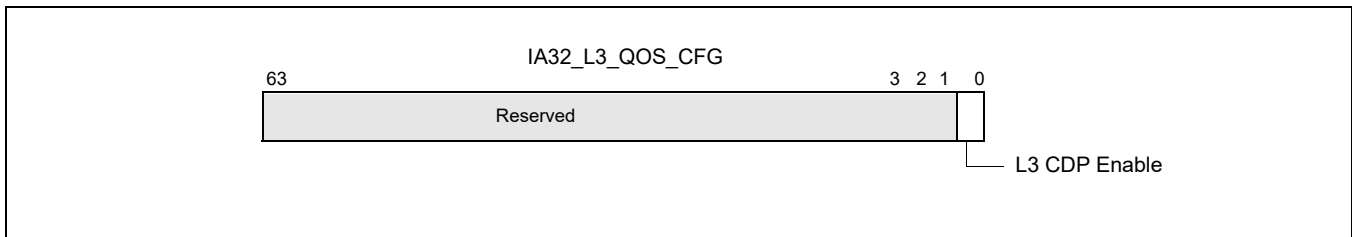


Figure 17-36. Layout of IA32_L3_QOS_CFG

IA32_L3_QOS_CFG default values are all 0s at RESET, the mask MSRs are all 1s. Hence, all logical processors are initialized in COS0 allocated with the entire L3 with CDP disabled, until software programs CAT and CDP. The scope of the IA32_L3_QOS_CFG MSR is defined to be the same scope as the L3 cache (e.g., typically per processor socket). Refer to Section 17.19.7 for software considerations while enabling or disabling L3 CDP.

17.19.5.1 Mapping Between L3 CDP Masks and CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per COS. The re-mapping is shown in Table 17-19.

Table 17-19. Re-indexing of COS Numbers and Mapping to CAT/CDP Mask MSRs

Mask MSR	CAT-only Operation	CDP Operation
IA32_L3_QOS_Mask_0	COS0	COS0.Data
IA32_L3_QOS_Mask_1	COS1	COS0.Code
IA32_L3_QOS_Mask_2	COS2	COS1.Data
IA32_L3_QOS_Mask_3	COS3	COS1.Code
IA32_L3_QOS_Mask_4	COS4	COS2.Data
IA32_L3_QOS_Mask_5	COS5	COS2.Code
....
IA32_L3_QOS_Mask_‘2n’	COS‘2n’	COS‘n’.Data
IA32_L3_QOS_Mask_‘2n+1’	COS‘2n+1’	COS‘n’.Code

One can derive the MSR address for the data mask or code mask for a given COS number ‘n’ by:

- data_mask_address (n) = base + (n <<1), where base is the address of IA32_L3_QOS_MASK_0.
- code_mask_address (n) = base + (n <<1) +1.

When CDP is enabled, each COS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 17-29).

Mask MSR field definitions remain the same. Capacity masks must be formed of contiguous set bits, with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003 and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

17.19.6 Code and Data Prioritization (CDP): Enumerating and Enabling L2 CDP Technology

L2 CDP is an extension of the L2 CAT feature. The presence of the L2 CDP feature is enumerated via CPUID.(EAX=10H, ECX=2):ECX.CDP[bit 2] (see Figure 17-33). Most of the CPUID.(EAX=10H, ECX=2) sub-leaf data that applies to CAT also apply to CDP. However, CPUID.(EAX=10H, ECX=2):EDX.COS_MAX_CAT specifies the maximum COS applicable to CAT-only operation. For CDP operations, COS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=2):EDX.COS_MAX_CAT >>1).

If CPUID.(EAX=10H, ECX=2):ECX.CDP[bit 2] =1, the processor supports L2 CDP and provides a new MSR IA32_L2_QOS_CFG at address 0C82H. The layout of IA32_L2_QOS_CFG is shown in Figure 17-37. The bit field definition of IA32_L2_QOS_CFG are:

- Bit 0: L2 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.COS is COS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

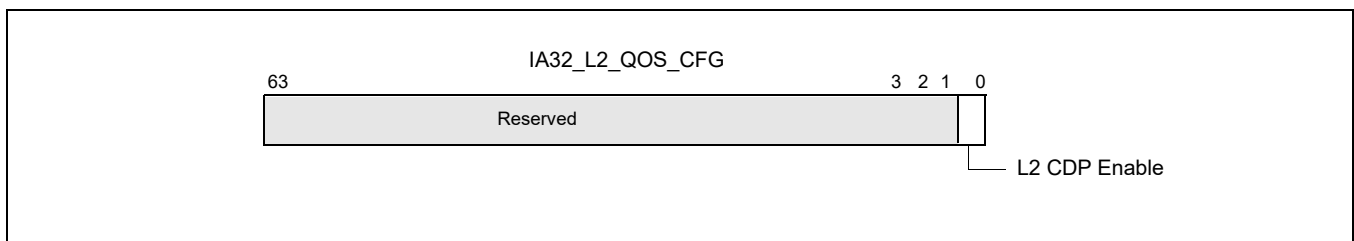


Figure 17-37. Layout of IA32_L2_QOS_CFG

IA32_L2_QOS_CFG default values are all 0s at RESET, and the mask MSRs are all 1s. Hence all logical processors are initialized in COS0 allocated with the entire L2 available and with CDP disabled, until software programs CAT and CDP. The IA32_L2_QOS_CFG MSR is defined at the same scope as the L2 cache, typically at the module level for Intel Atom processors for instance. In processors with multiple modules present it is recommended to program the IA32_L2_QOS_CFG MSR consistently across all modules for simplicity.

17.19.6.1 Mapping Between L2 CDP Masks and L2 CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per COS. This remapping is the same as the remapping shown in Table 17-19 for L3 CDP, but for the L2 MSR block (IA32_L2_QOS_MASK_n) instead of the L3 MSR block (IA32_L3_QOS_MASK_n). The same code / data mask mapping algorithm applies to remapping the MSR block between code and data masks.

As with L3 CDP, when L2 CDP is enabled, each COS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 17-29).

Mask MSR field definitions for L2 CDP remain the same as for L2 CAT. Capacity masks must be formed of contiguous set bits, with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003 and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

17.19.6.2 Common L2 and L3 CDP Programming Considerations

Before enabling or disabling L2 or L3 CDP, software should write all 1's to all of the corresponding CAT/CDP masks to ensure proper behavior (e.g., the IA32_L3_QOS_Mask_n set of MSRs for the L3 CAT feature). When enabling CDP, software should also ensure that only COS number which are valid in CDP operation is used, otherwise undefined behavior may result. For instance in a case with 16 CAT COS, since COS are reduced by half when CDP is enabled, software should ensure that only COS 0-7 are in use before enabling CDP (along with writing 1's to all mask bits before enabling or disabling CDP).

Software should also account for the fact that mask interpretations change when CDP is enabled or disabled, meaning for instance that a CAT mask for a given COS may become a code mask for a different Class of Service when CDP is enabled. In order to simplify this behavior and prevent unintended remapping software should consider resetting all threads to COS[0] before enabling or disabling CDP.

17.19.6.3 Cache Allocation Technology Dynamic Configuration

All Resource Director Technology (RDT) interfaces including the IA32_PQR_ASSOC MSR, CAT/CDP masks, MBA delay values and CQM/MBM registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,
- Accessing a QOS mask register outside the supported COS (the max COS number is specified in CPUID.(EAX=10H, ECX=ResID):EDX[15:0]), or
- Writing a COS greater than the supported maximum (specified as the maximum value of CPUID.(EAX=10H, ECX=ResID):EDX[15:0] for all valid ResID values) is written to the IA32_PQR_ASSOC.CLOS field.

When CDP is enabled, specifying a COS value in IA32_PQR_ASSOC.COS outside of the lower half of the COS space will cause undefined performance impact to code and data fetches due to MSR space re-indexing into code/data masks when CDP is enabled.

When reading the IA32_PQR_ASSOC register the currently programmed COS on the core will be returned.

When reading an IA32_resourceType_MASK_n register the current capacity bit mask for COS 'n' will be returned.

As noted previously, software should minimize migrations of COS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if COS are migrated frequently.

This is aligned with the industry standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and COS migration across processor logical threads and processor cores.

17.19.6.4 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.

17.19.6.5 Cache Allocation Technology Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and mask registers are unmodified across an SMI delivery. Thus, the execution of SMM handler code can interact with the Cache Allocation Technology resource and manifest some degree of non-determinism to the non-SMM software stack. An SMM handler may also perform certain system-level or power management practices that affect CAT operation.

It is possible for an SMM handler to minimize the impact on data determinism in the cache by reserving a COS with a dedicated partition in the cache. Such an SMM handler can switch to the dedicated COS immediately upon entering SMM, and switching back to the previously running COS upon exit.

17.19.6.6 Associating Threads with CAT/CDP Classes of Service

Threads are associated with Classes of Service (CLOS) via the per-logical-processor IA32_PQR_ASSOC MSR. The same COS concept applies to both CAT and CDP (for instance, COS[5] means the same thing whether CAT or CDP is in use, and the COS has associated resource usage constraint attributes including cache capacity masks). The mapping of COS to mask MSRs does change when CDP is enabled, according to the following guidelines:

- In CAT-only Mode - one set of bitmasks in one mask MSR control both code and data.
 - Each COS number map 1:1 with a capacity mask on the applicable resource (e.g., L3 cache).
- When CDP is enabled,
 - Two mask sets exist for each COS number, one for code, one for data.
 - Masks for code/data are interleaved in the MSR address space (see Table 17-19).

17.19.7 Introduction to Memory Bandwidth Allocation

The Memory Bandwidth Allocation (MBA) feature provides indirect and approximate control over memory bandwidth available per-core, and was introduced on the Intel Xeon Processor Scalable Family. This feature provides a method to control applications which may be over-utilizing bandwidth relative to their priority in environments such as the data-center.

The MBA feature uses existing constructs from the Resource Director Technology (RDT) feature set including Classes of Service (CLOS). A given CLOS used for L3 CAT for instance means the same thing as a CLOS used for MBA. Infrastructure such as the MSR used to associate a thread with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID leaf 10H) are shared.

The high-level implementation of Memory Bandwidth Allocation is shown in Figure 17-38.

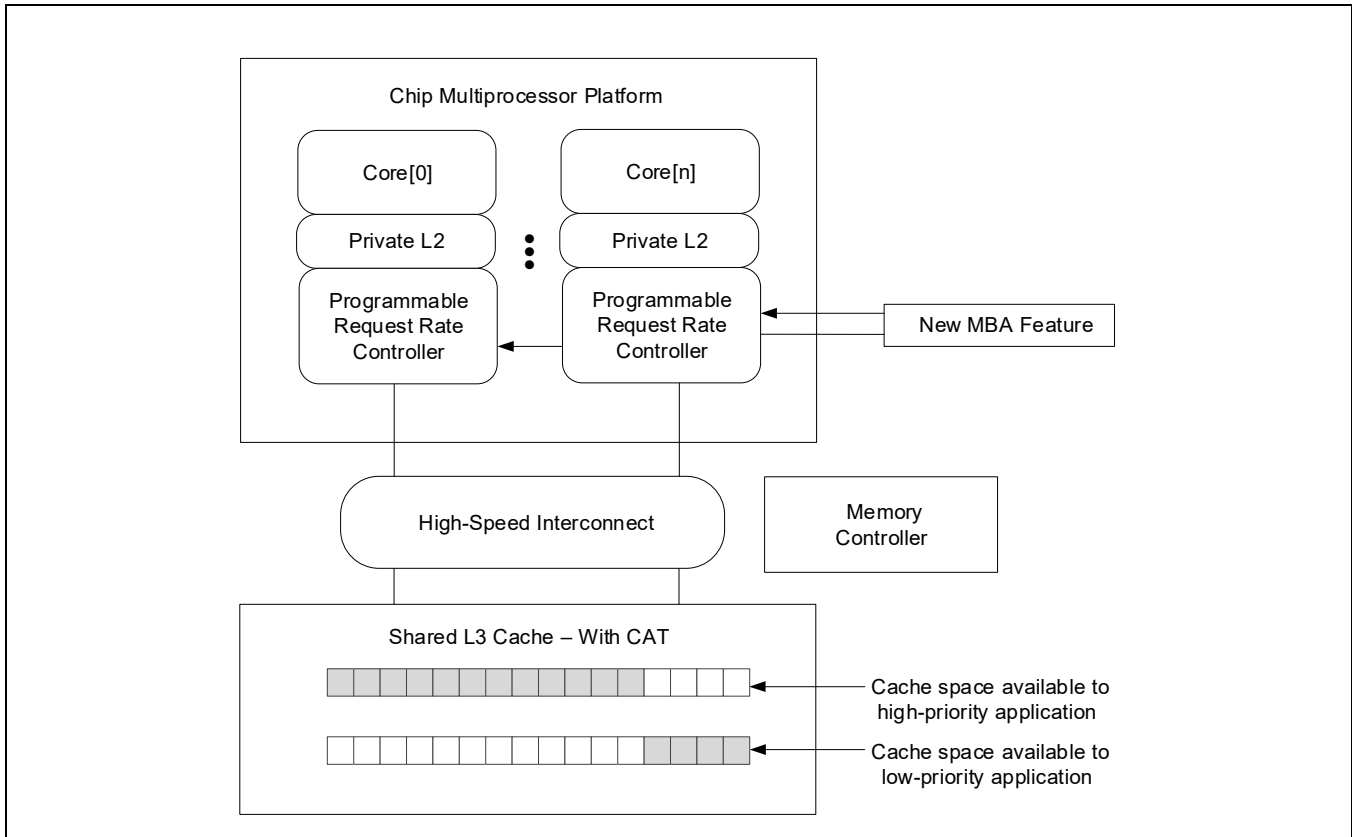


Figure 17-38. A High-Level Overview of the MBA Feature

As shown in Figure 17-38, the MBA feature introduces a programmable request rate controller between the cores and the high-speed interconnect, enabling indirect control over memory bandwidth for cores over-utilizing bandwidth relative to their priority. For instance, high-priority cores may be run un-throttled, but lower priority cores generating an excessive amount of traffic may be throttled to enable more bandwidth availability for the high-priority cores.

Since MBA uses a programmable rate controller between the cores and the interconnect, higher-level shared caches and memory controller, bandwidth to these caches may also be reduced, so care should be taken to throttle only bandwidth-intense applications which do not use the off-core caches effectively.

The throttling values exposed by MBA are approximate, and are calibrated to specific traffic patterns. As work-load characteristics vary, the throttling values provided may affect each workload differently. In cases where precise control is needed, the Memory Bandwidth Monitoring (MBM) feature can be used as input to a software controller which makes decisions about the MBA throttling level to apply.

Enumeration and configuration details are discussed below followed by usage model considerations.

17.19.7.1 Memory Bandwidth Allocation Enumeration

Similar to other RDT features, enumeration of the presence and details of the MBA feature is provided via a sub-leaf of the CPUID instruction.

Key components of the enumeration are as follows.

- Support for the MBA feature on the processor, and if MBA is supported, the following details:
 - Number of supported Classes of Service (CLOS) for the processor.
 - The maximum MBA delay value supported (which also implicitly provides a definition of the granularity).
 - An indication of whether the delay values which can be programmed are linearly spaced or not.

The presence of any of the RDT features which enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.(EAX=07H, ECX=0):EBX.PQE[bit 15] reports 1, the processor supports software control over shared processor resources. Software may then use CPUID leaf 10H to enumerate additional details on the specific controls provided.

Through CPUID leaf 10H software may determine whether MBA is supported on the platform. Specifically, as shown in Figure 17-31, bit 3 of the EBX register indicates whether MBA is supported on the processor, and the bit position (3) constitutes a Resource ID (ResID) which allows enumeration of MBA details. For instance, if bit 3 is supported this implies the presence of CPUID.10H.[ResID=3] as shown in Figure 17-38 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 RDT features support different numbers of CLOS the lowest numerical CLOS support the common set of features, while higher CLOS may support a subset. For instance, if L3 CAT supports 8 CLOS while MBA supports 4 CLOS, all 8 CLOS would have L3 CAT masks available for cache control, but the upper 4 CLOS would not offer MBA support. In this case the upper 4 CLOS would not be subject to any throttling control. Software can manage supported resources / CLOS in order to either have consistent capabilities across CLOS by using the common subset or enable more flexibility by selectively applying resource control where needed based on careful CLOS and thread mapping. In all cases, CLOS[0] supports all RDT resource control features present on the platform.

Discussion on the interpretation and usage of the MBA delay values is provided in Section 17.19.7.2 on MBA configuration.

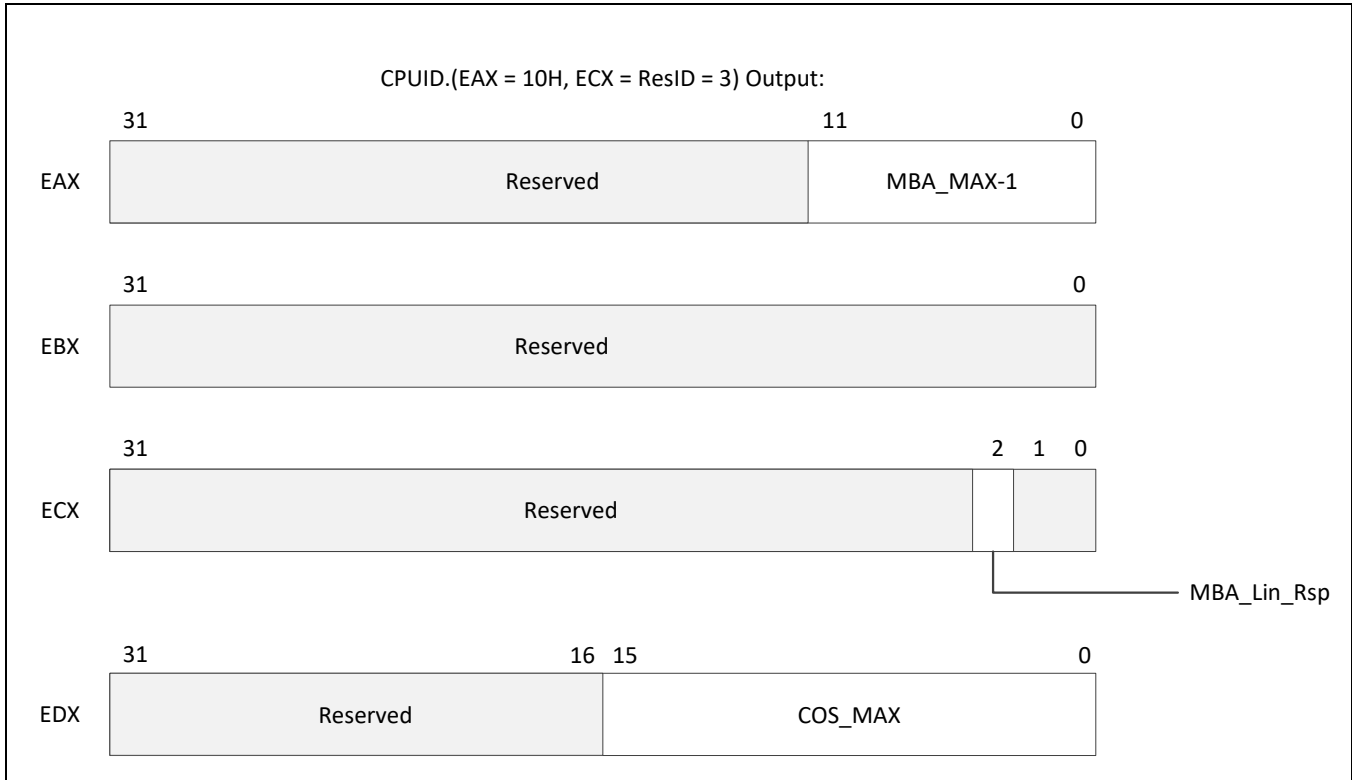


Figure 17-39. CPUID.(EAX=10H, ECX=3H) MBA Feature Details Identification

17.19.7.2 Memory Bandwidth Allocation Configuration

The configuration of MBA takes consists of two processes once enumeration is complete.

- Association of threads to Classes of Service (CLOS) - accomplished in a common fashion across RDT features as described in Section 17.19.7.1 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 17.19.7.3 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 17-20.

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 17.19.7.1. The throttling values are approximate and do not sum to 100% across CLOS, rather they should be viewed as a maximum bandwidth “cap” per-CLOS.

Software may select an MBA delay value then write the value into one or more of the IA32_L2_QoS_Ext_BW_Thrtl_n MSRs to update the delay values applied for a specific CLOS. As shown in Table 17-20 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 17.19.7.1. For instance, if 16 CLOS are supported then the valid MSR range will extend from D50H through D5F inclusive.

Table 17-20. MBA Delay Value MSRs

Delay Value MSR	Address
IA32_L2_QoS_Ext_BW_Thrtl_0	D50H
IA32_L2_QoS_Ext_BW_Thrtl_1	D51H
IA32_L2_QoS_Ext_BW_Thrtl_2	D52H
....
IA32_L2_QoS_Ext_BW_Thrtl_'COS_MAX'	D50H + COS_MAX from CPUID.10H.3

The definition for the MBA delay value MSRs is provided in Figure 17.39. The lower 16 bits are used for MBA delay values, and values from zero to the maximum from the CPUID.MBA_MAX-1 value are supported. Values outside this range will generate #GP(0).

If linear input throttling values are indicated by CPUID.(EAX=10H, ECX=ResID=3):ECX[bit 2] then values from zero through the MBA_MAX field from CPUID.(EAX=10H, ECX=ResID=3):EAX[11:0] are supported as inputs. In the linear mode the input precision is defined as 100-(MBA_MAX). For instance, if the MBA_MAX value is 90, the input precision is 10%. Values not an even multiple of the precision (e.g., 12%) will be rounded down (e.g., to 10% delay applied).

- If linear values are not supported (CPUID.(EAX=10H, ECX=ResID=3):ECX[bit 2] = 0) then input delay values are powers-of-two from zero to the MBA_MAX value from CPUID. In this case any values not a power of two will be rounded down the next nearest power of two.

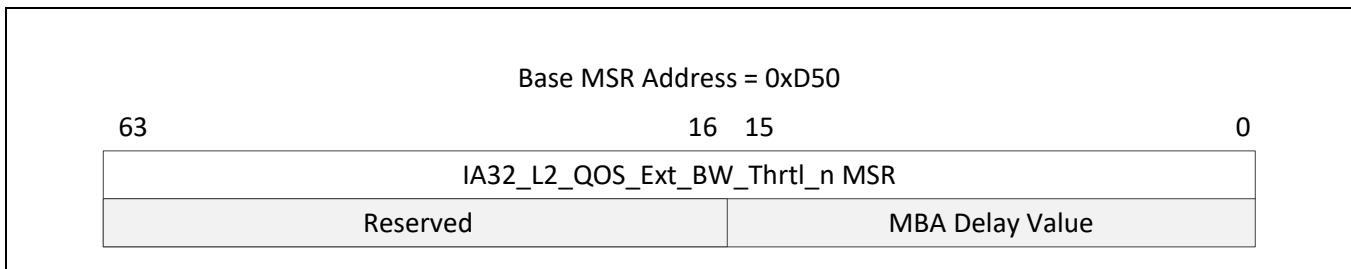


Figure 17-40. IA32_L2_QoS_Ext_BW_Thrtl_n MSR Definition

Note that the throttling values provided to software are calibrated through specific traffic patterns, however as workload characteristics may vary the response precision and linearity of the delay values will vary across products, and should be treated as approximate values only.

17.19.7.3 Memory Bandwidth Allocation Usage Considerations

As the memory bandwidth control that MBA provides is indirect and approximate, using the feature with a closed-loop controller to also monitor memory bandwidth and how effectively the applications use the cache (via the Cache Monitoring Technology feature) may provide additional value. This approach also allows administrators to provide a band-width target or set-point which a controller could use to guide MBA throttling values applied, and this allows bandwidth control independent of the execution characteristics of the application.

As control is provided per processor core (the max of the delay values of the per-thread CLOS applied to the core) care should be taking in scheduling threads so as to not inadvertently place a high-priority thread (with zero intended MBA throttling) next to a low-priority thread (with MBA throttling intended), which would lead to inadvertent throttling of the high-priority thread.

11. Updates to Chapter 19, Volume 3B

Change bars and green text show changes to Chapter 19 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter: Updates to Table 19-54, "E-core PEBS Data Source Encodings".

Intel 64 and IA-32 architectures provide facilities for monitoring performance via a PMU (Performance Monitoring Unit).

NOTE

Performance monitoring events can be found here: <https://perfmon-events.intel.com/>.

Additionally, performance monitoring event files for Intel processors are hosted by the Intel Open Source Technology Center. These files can be downloaded here:

<https://download.01.org/perfmon/>.

19.1 PERFORMANCE MONITORING OVERVIEW

Performance monitoring was introduced in the Pentium processor with a set of model-specific performance-monitoring counter MSRs. These counters permit selection of processor performance parameters to be monitored and measured. The information obtained from these counters can be used for tuning system and compiler performance.

In Intel P6 family of processors, the performance monitoring mechanism was enhanced to permit a wider selection of events to be monitored and to allow greater control events to be monitored. Next, Intel processors based on Intel NetBurst microarchitecture introduced a distributed style of performance monitoring mechanism and performance events.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, and Intel processors based on Intel NetBurst microarchitecture are not architectural. They are all model specific (not compatible among processor families). Intel Core Solo and Intel Core Duo processors support a set of architectural performance events and a set of non-architectural performance events. Newer Intel processor generations support enhanced architectural performance events and non-architectural performance events.

Starting with Intel Core Solo and Intel Core Duo processors, there are two classes of performance monitoring capabilities. The first class supports events for monitoring performance using counting or interrupt-based event sampling usage. These events are non-architectural and vary from one processor model to another. They are similar to those available in Pentium M processors. These non-architectural performance monitoring events are specific to the microarchitecture and may change with enhancements. They are discussed in Section 19.6.3, "Performance Monitoring (Processors Based on Intel NetBurst[®] Microarchitecture)." Non-architectural events for a given microarchitecture cannot be enumerated using CPUID; and they can be found at: <https://perfmon-events.intel.com/>.

The second class of performance monitoring capabilities is referred to as architectural performance monitoring. This class supports the same counting and interrupt-based event sampling usages, with a smaller set of available events. The visible behavior of architectural performance events is consistent across processor implementations. Availability of architectural performance monitoring capabilities is enumerated using the CPUID.0AH. These events are discussed in Section 19.2.

See also:

- Section 19.2, "Architectural Performance Monitoring".
- Section 19.3, "Performance Monitoring (Intel[®] Core[™] Processors and Intel[®] Xeon[®] Processors)".
 - Section 19.3.1, "Performance Monitoring for Processors Based on Nehalem Microarchitecture".
 - Section 19.3.2, "Performance Monitoring for Processors Based on Westmere Microarchitecture".
 - Section 19.3.3, "Intel[®] Xeon[®] Processor E7 Family Performance Monitoring Facility".
 - Section 19.3.4, "Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture".
 - Section 19.3.5, "3rd Generation Intel[®] Core[™] Processor Performance Monitoring Facility".

- Section 19.3.6, “4th Generation Intel® Core™ Processor Performance Monitoring Facility”.
- Section 19.3.7, “5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility”.
- Section 19.3.8, “6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility”.
- Section 19.3.9, “10th Generation Intel® Core™ Processor Performance Monitoring Facility”.
- Section 19.3.10, “12th Generation Intel® Core™ Processor Performance Monitoring Facility”.
- Section 19.4, “Performance monitoring (Intel® Xeon™ Phi Processors)”.
 - Section 19.4.1, “Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring”.
- Section 19.5, “Performance Monitoring (Intel Atom® Processors)”.
 - Section 19.5.1, “Performance Monitoring (45 nm and 32 nm Intel Atom® Processors)”.
 - Section 19.5.2, “Performance Monitoring for Silvermont Microarchitecture”.
 - Section 19.5.3, “Performance Monitoring for Goldmont Microarchitecture”.
 - Section 19.5.4, “Performance Monitoring for Goldmont Plus Microarchitecture”.
 - Section 19.5.5, “Performance Monitoring for Tremont Microarchitecture”.
- Section 19.6, “Performance Monitoring (Legacy Intel Processors)”.
 - Section 19.6.1, “Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)”.
 - Section 19.6.2, “Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)”.
 - Section 19.6.3, “Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)”.
 - Section 19.6.4, “Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture”.
 - Section 19.6.4.5, “Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture”.
 - Section 19.6.5, “Performance Monitoring and Dual-Core Technology”.
 - Section 19.6.6, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache”.
 - Section 19.6.7, “Performance Monitoring on L3 and Caching Bus Controller Sub-Systems”.
 - Section 19.6.8, “Performance Monitoring (P6 Family Processor)”.
 - Section 19.6.9, “Performance Monitoring (Pentium Processors)”.
- Section 19.7, “Counting Clocks”.
- Section 19.8, “IA32_PERF_CAPABILITIES MSR Enumeration”.
- Section 19.9, “PEBS Facility”.

19.2 ARCHITECTURAL PERFORMANCE MONITORING

Performance monitoring events are architectural when they behave consistently across microarchitectures. Intel Core Solo and Intel Core Duo processors introduced architectural performance monitoring. The feature provides a mechanism for software to enumerate performance events and provides configuration and counting facilities for events.

Architectural performance monitoring does allow for enhancement across processor implementations. The CPUID.0AH leaf provides version ID for each enhancement. Intel Core Solo and Intel Core Duo processors support base level functionality identified by version ID of 1. Processors based on Intel Core microarchitecture support, at a minimum, the base level functionality of architectural performance monitoring. Intel Core 2 Duo processor T 7700 and newer processors based on Intel Core microarchitecture support both the base level functionality and enhanced architectural performance monitoring identified by version ID of 2.

45 nm and 32 nm Intel Atom processors and Intel Atom processors based on the Silvermont microarchitecture support the functionality provided by versionID 1, 2, and 3; CPUID.0AH:EAX[7:0] reports versionID = 3 to indicate the aggregate of architectural performance monitoring capabilities. Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture. Intel Atom processors based on the Goldmont and Goldmont Plus microarchitectures support versionID 4. Intel Atom processors starting with processors based on the Tremont microarchitecture support versionID 5.

Intel Core processors and related Intel Xeon processor families based on the Nehalem through Broadwell microarchitectures support version ID 3. Intel processors based on the Skylake through Coffee Lake microarchitectures support versionID 4. Intel processors starting with processors based on the Ice Lake microarchitecture support versionID 5.

19.2.1 Architectural Performance Monitoring Version 1

Configuring an architectural performance monitoring event involves programming performance event select registers. There are a finite number of performance event select MSR (IA32_PERFEVTSELx MSRs). The result of a performance monitoring event is reported in a performance monitoring counter (IA32_PMCx MSR). Performance monitoring counters are paired with performance monitoring select registers.

Performance monitoring select registers and counters are architectural in the following respects:

- Bit field layout of IA32_PERFEVTSELx is consistent across microarchitectures.
- Addresses of IA32_PERFEVTSELx MSRs remain the same across microarchitectures.
- Addresses of IA32_PMC MSRs remain the same across microarchitectures.
- Each logical processor has its own set of IA32_PERFEVTSELx and IA32_PMCx MSRs. Configuration facilities and counters are not shared between logical processors sharing a processor core.

Architectural performance monitoring provides a CPUID mechanism for enumerating the following information:

- Number of performance monitoring counters available to software in a logical processor (each IA32_PERFEVTSELx MSR is paired to the corresponding IA32_PMCx MSR).
- Number of bits supported in each IA32_PMCx.
- Number of architectural performance monitoring events supported in a logical processor.

Software can use CPUID to discover architectural performance monitoring availability (CPUID.0AH). The architectural performance monitoring leaf provides an identifier corresponding to the version number of architectural performance monitoring available in the processor.

The version identifier is retrieved by querying CPUID.0AH:EAX[bits 7:0] (see Chapter 3, “Instruction Set Reference, A-L,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*). If the version identifier is greater than zero, architectural performance monitoring capability is supported. Software queries the CPUID.0AH for the version identifier first; it then analyzes the value returned in CPUID.0AH.EAX, CPUID.0AH.EBX to determine the facilities available.

In the initial implementation of architectural performance monitoring; software can determine how many IA32_PERFEVTSELx/ IA32_PMCx MSR pairs are supported per core, the bit-width of PMC, and the number of architectural performance monitoring events available.

19.2.1.1 Architectural Performance Monitoring Version 1 Facilities

Architectural performance monitoring facilities include a set of performance monitoring counters and performance event select registers. These MSRs have the following properties:

- IA32_PMCx MSRs start at address 0C1H and occupy a contiguous block of MSR address space; the number of MSRs per logical processor is reported using CPUID.0AH:EAX[15:8]. Note that this may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.
- IA32_PERFEVTSELx MSRs start at address 186H and occupy a contiguous block of MSR address space. Each performance event select register is paired with a corresponding performance counter in the 0C1H address

block. Note the number of IA32_PERFEVTSELx MSRs may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.

- The bit width of an IA32_PMCx MSR is reported using the CPUID.0AH:EAX[23:16]. This the number of valid bits for read operation. On write operations, the lower-order 32 bits of the MSR may be written with any value, and the high-order bits are sign-extended from the value of bit 31.
- Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally.

See Figure 19-1 for the bit field layout of IA32_PERFEVTSELx MSRs. The bit fields are:

- **Event select field (bits 0 through 7)** — Selects the event logic unit used to detect microarchitectural conditions (see Table 19-1, for a list of architectural events and their 8-bit codes). The set of values for this field is defined architecturally; each value corresponds to an event logic unit for use with an architectural performance event. The number of architectural events is queried using CPUID.0AH:EAX. A processor may support only a subset of pre-defined values.

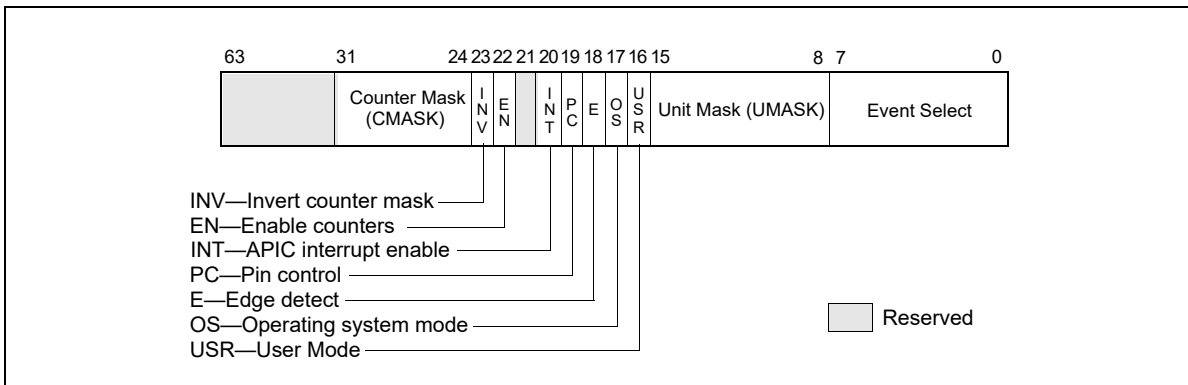


Figure 19-1. Layout of IA32_PERFEVTSELx MSRs

- **Unit mask (UMASK) field (bits 8 through 15)** — These bits qualify the condition that the selected event logic unit detects. Valid UMASK values for each event logic unit are specific to the unit. For each architectural performance event, its corresponding UMASK value defines a specific microarchitectural condition.

A pre-defined microarchitectural condition associated with an architectural event may not be applicable to a given processor. The processor then reports only a subset of pre-defined architectural events. Pre-defined architectural events are listed in Table 19-1; support for pre-defined architectural events is enumerated using CPUID.0AH:EBX.

- **USR (user mode) flag (bit 16)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege levels 1, 2 or 3. This flag can be used with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege level 0. This flag can be used with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields. The mechanism does not permit back-to-back assertions to be distinguished.

This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

- **PC (pin control) flag (bit 19)** — Beginning with Sandy Bridge microarchitecture, this bit is reserved (not writeable). On processors based on previous microarchitectures, the logical processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.

- **INT (APIC interrupt enable) flag (bit 20)** — When set, the logical processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — When set, performance counting is enabled in the corresponding performance-monitoring counter; when clear, the corresponding counter is disabled. The event logic unit for a UMASK must be disabled by setting IA32_PERFEVTSELx[bit 22] = 0, before writing to IA32_PMCx.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When this field is not zero, a logical processor compares this mask to the events count of the detected microarchitectural condition during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented.

This mask is intended for software to characterize microarchitectural conditions that can count multiple occurrences per cycle (for example, two or more instructions retired per clock; or bus queue occupations). If the counter-mask field is 0, then the counter is incremented each cycle by the event count associated with multiple occurrences.

19.2.1.2 Pre-defined Architectural Performance Events

Table 19-1 lists architecturally defined events.

Table 19-1. UMask and Event Select Encodings for Pre-Defined Architectural Performance Events

Bit Position CPUID.AH.EBX	Event Name	UMask	Event Select
0	UnHalted Core Cycles	00H	3CH
1	Instruction Retired	00H	C0H
2	UnHalted Reference Cycles ¹	01H	3CH
3	LLC Reference	4FH	2EH
4	LLC Misses	41H	2EH
5	Branch Instruction Retired	00H	C4H
6	Branch Misses Retired	00H	C5H
7	Topdown Slots	01H	A4H

NOTES:

1. Implementations prior to the 12th generation Intel® Core™ processor P-cores count at core crystal clock, TSC, or bus clock frequency.

A processor that supports architectural performance monitoring may not support all the predefined architectural performance events (Table 19-1). The number of architectural events is reported through CPUID.0AH:EAX[31:24], while non-zero bits in CPUID.0AH:EBX indicate any architectural events that are not available.

The behavior of each architectural performance event is expected to be consistent on all processors that support that event. Minor variations between microarchitectures are noted below:

- **UnHalted Core Cycles** — Event select 3CH, Umask 00H
This event counts core clock cycles when the clock signal on a specific core is running (not halted). The counter does not advance in the following conditions:
 - an ACPI C-state other than C0 for normal operation
 - HLT
 - STPCLK# pin asserted
 - being throttled by TM1
 - during the frequency switching phase of a performance state transition (see Chapter 14, “Power and Thermal Management”)

The performance counter for this event counts across performance state transitions using different core clock frequencies.

- **Instructions Retired** — Event select C0H, Umask 00H

This event counts the number of instructions at retirement. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. An instruction with a REP prefix counts as one instruction (not per iteration). Faults before the retirement of the last micro-op of a multi-ops instruction are not counted.

This event does not increment under VM-exit conditions. Counters continue counting during hardware interrupts, traps, and inside interrupt handlers.

- **UnHalted Reference Cycles** — Event select 3CH, Umask 01H

This event counts reference clock cycles at a fixed frequency while the clock signal on the core is running. The event counts at a fixed frequency, irrespective of core frequency changes due to performance state transitions. Processors may implement this behavior differently. Current implementations use the core crystal clock, TSC or the bus clock. Because the rate may differ between implementations, software should calibrate it to a time source with known frequency.

- **Last Level Cache References** — Event select 2EH, Umask 4FH

This event counts requests originating from the core that reference a cache line in the last level on-die cache. The event count includes speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Last Level Cache Misses** — Event select 2EH, Umask 41H

This event counts each cache miss condition for references to the last level on-die cache. The event count may include speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Branch Instructions Retired** — Event select C4H, Umask 00H

This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction.

- **All Branch Mispredict Retired** — Event select C5H, Umask 00H

This event counts mispredicted branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction in the architectural path of execution and experienced misprediction in the branch prediction hardware.

Branch prediction hardware is implementation-specific across microarchitectures; value comparison to estimate performance differences is not recommended.

- **Topdown Slots** — Event select A4H, Umask 01H

This event counts the total number of available slots for an unhalted logical processor.

The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core, in processors that support Intel Hyper-Threading Technology.

Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

NOTE

Programming decisions or software precisions on functionality should not be based on the event values or dependent on the existence of performance monitoring events.

19.2.2 Architectural Performance Monitoring Version 2

The enhanced features provided by architectural performance monitoring version 2 include the following:

- **Fixed-function performance counter register and associated control register** — Three of the architectural performance events are counted using three fixed-function MSRs (IA32_FIXED_CTR0 through IA32_FIXED_CTR2). Each of the fixed-function PMC can count only one architectural performance event.
Configuring the fixed-function PMCs is done by writing to bit fields in the MSR (IA32_FIXED_CTR_CTRL) located at address 38DH. Unlike configuring performance events for general-purpose PMCs (IA32_PMCx) via UMASK field in (IA32_PERFECTSELx), configuring, programming IA32_FIXED_CTR_CTRL for fixed-function PMCs do not require any UMASK.
- **Simplified event programming** — Most frequent operation in programming performance events are enabling/disabling event counting and checking the status of counter overflows. Architectural performance event version 2 provides three architectural MSRs:
 - IA32_PERF_GLOBAL_CTRL allows software to enable/disable event counting of all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or any general-purpose PMCs via a single WRMSR.
 - IA32_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single RDMSR.
 - IA32_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single WRMSR.
- **PMI Overhead Mitigation** — Architectural performance monitoring version 2 introduces two bit field interface in IA32_DEBUGCTL for PMI service routine to accumulate performance monitoring data and LBR records with reduced perturbation from servicing the PMI. The two bit fields are:
 - IA32_DEBUGCTL.Freeze_LBR_On_PMI(bit 11). In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 17.4.7 for details of the legacy Freeze LBRs on PMI control.
 - IA32_DEBUGCTL.Freeze_PerfMon_On_PMI(bit 12). In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 17.4.7 for details of the legacy Freeze LBRs on PMI control.

The facilities provided by architectural performance monitoring version 2 can be queried from CPUID leaf 0AH by examining the content of register EDX:

- Bits 0 through 4 of CPUID.0AH.EDX indicates the number of fixed-function performance counters available per core,
- Bits 5 through 12 of CPUID.0AH.EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.

NOTE

Early generation of processors based on Intel Core microarchitecture may report in CPUID.0AH:EDX of support for version 2 but indicating incorrect information of version 2 facilities.

The IA32_FIXED_CTR_CTRL MSR include multiple sets of 4-bit field, each 4 bit field controls the operation of a fixed-function performance counter. Figure 19-2 shows the layout of 4-bit controls for each fixed-function PMC. Two sub-fields are currently defined within each control. The definitions of the bit fields are:

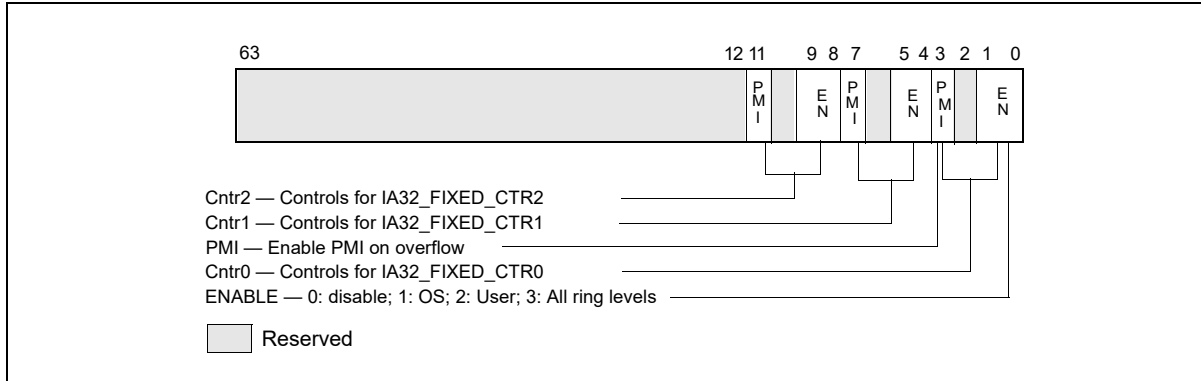


Figure 19-2. Layout of IA32_FIXED_CTR_CTRL MSR

- Enable field (lowest 2 bits within each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring 0. When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring greater than 0. Writing 0 to both bits stops the performance counter. Writing a value of 11B enables the counter to increment irrespective of privilege levels.
- PMI field (the fourth bit within each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

IA32_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting of each performance counter. Figure 19-3 shows the layout of IA32_PERF_GLOBAL_CTRL. Each enable bit in IA32_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

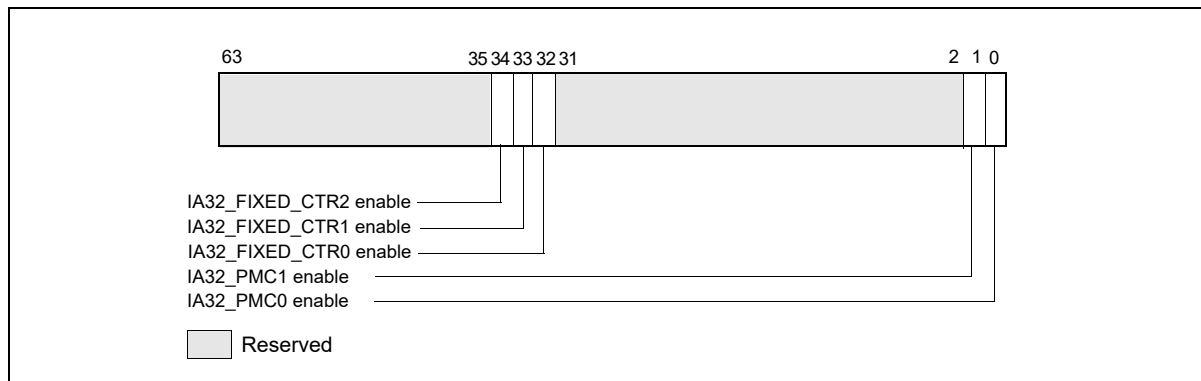


Figure 19-3. Layout of IA32_PERF_GLOBAL_CTRL MSR

The behavior of the fixed function performance counters supported by architectural performance version 2 is expected to be consistent on all processors that support those counters, and is defined as follows.

Table 19-2. Association of Fixed-Function Performance Counters with Architectural Performance Events

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_FIXED_CTR0	309H	INST_RETIRED.ANY	This event counts the number of instructions that retire execution. For instructions that consist of multiple uops, this event counts the retirement of the last uop of the instruction. The counter continues counting during hardware interrupts, traps, and in-side interrupt handlers.
IA32_FIXED_CTR1	30AH	CPU_CLK_UNHALTED.THREAD CPU_CLK_UNHALTED.CORE	The CPU_CLK_UNHALTED.THREAD event counts the number of core cycles while the logical processor is not in a halt state. If there is only one logical processor in a processor core, CPU_CLK_UNHALTED.CORE counts the unhalted cycles of the processor core. The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time.
IA32_FIXED_CTR2	30BH	CPU_CLK_UNHALTED.REF_TSC	This event counts the number of reference cycles at the TSC rate when the core is not in a halt state and not in a TM stop-clock state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction. This event is not affected by core frequency changes (e.g., P states) but counts at the same frequency as the time stamp counter. This event can approximate elapsed time while the core was not in a halt state and not in a TM stopclock state.
IA32_FIXED_CTR3	30CH	TOPDOWN.SLOTS	This event counts the number of available slots for an unhalted logical processor. The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core. Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

IA32_PERF_GLOBAL_STATUS MSR provides single-bit status for software to query the overflow condition of each performance counter. IA32_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. IA32_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware. Figure 19-4 shows the layout of IA32_PERF_GLOBAL_STATUS. A value of 1 in bits 0, 1, 32 through 34 indicates a counter overflow condition has occurred in the associated counter.

When a performance counter is configured for PEBS, overflow condition in the counter generates a performance-monitoring interrupt signaling a PEBS event. On a PEBS event, the processor stores data records into the buffer area (see Section 18.15.5), clears the counter overflow status, and sets the "OvfBuffer" bit in IA32_PERF_GLOBAL_STATUS.

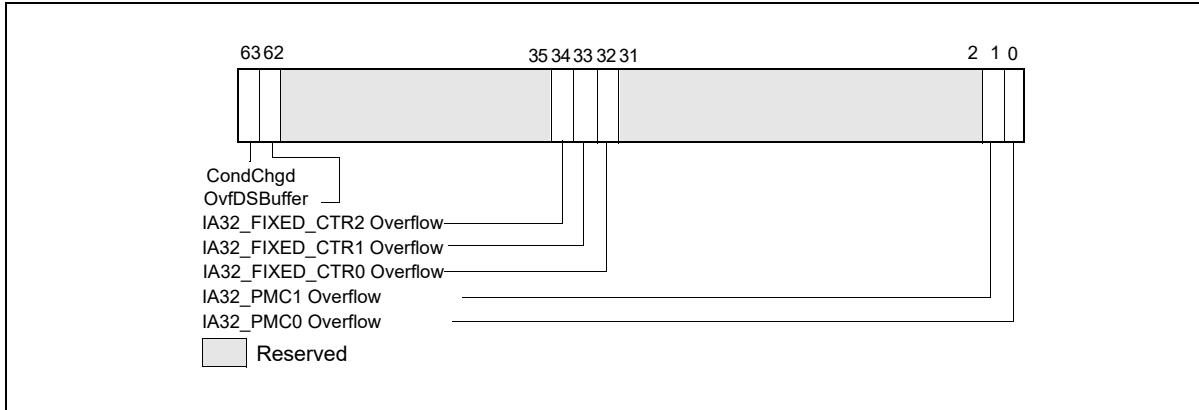


Figure 19-4. Layout of IA32_PERF_GLOBAL_STATUS MSR

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow indicator(s) of any general-purpose or fixed-function counters via a single WRMSR. Software should clear overflow indications when

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

The layout of IA32_PERF_GLOBAL_OVF_CTL is shown in Figure 19-5.

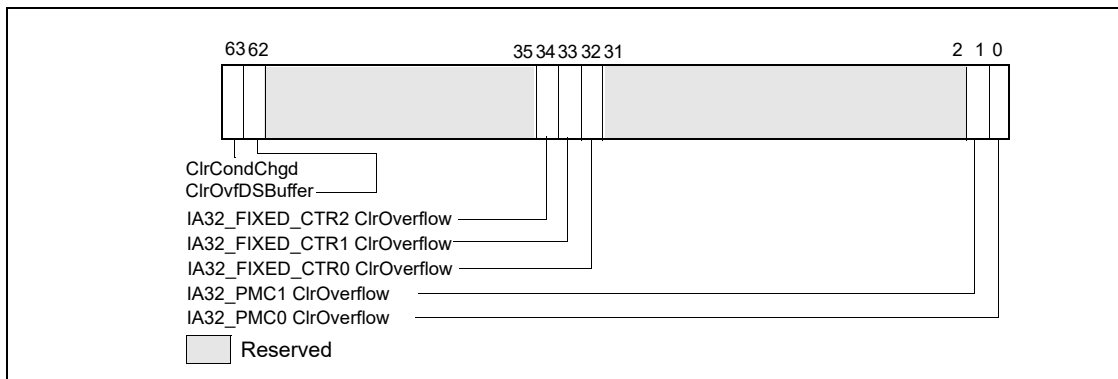


Figure 19-5. Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

19.2.3 Architectural Performance Monitoring Version 3

Processors supporting architectural performance monitoring version 3 also supports version 1 and 2, as well as capability enumerated by CPUID leaf 0AH. Specifically, version 3 provides the following enhancement in performance monitoring facilities if a processor core comprising of more than one logical processor, i.e., a processor core supporting Intel Hyper-Threading Technology or simultaneous multi-threading capability:

- AnyThread counting for processor core supporting two or more logical processors. The interface that supports AnyThread counting include:
 - Each IA32_PERFEVTSELx MSR (starting at MSR address 186H) support the bit field layout defined in Figure 19-6.

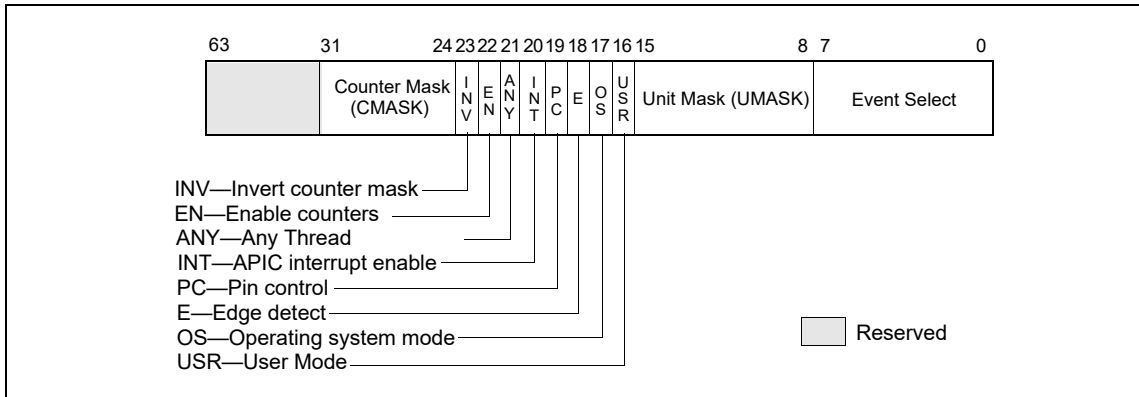


Figure 19-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3 for processor core comprising of two or more logical processors. When set to 1, it enables counting the associated event conditions (including matching the thread’s CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core. When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread’s CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.

- Each fixed-function performance counter IA32_FIXED_CTRx (starting at MSR address 309H) is configured by a 4-bit control block in the IA32_PERF_FIXED_CTR_CTRL MSR. The control block also allow thread-specificity configuration using an AnyThread bit. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown.

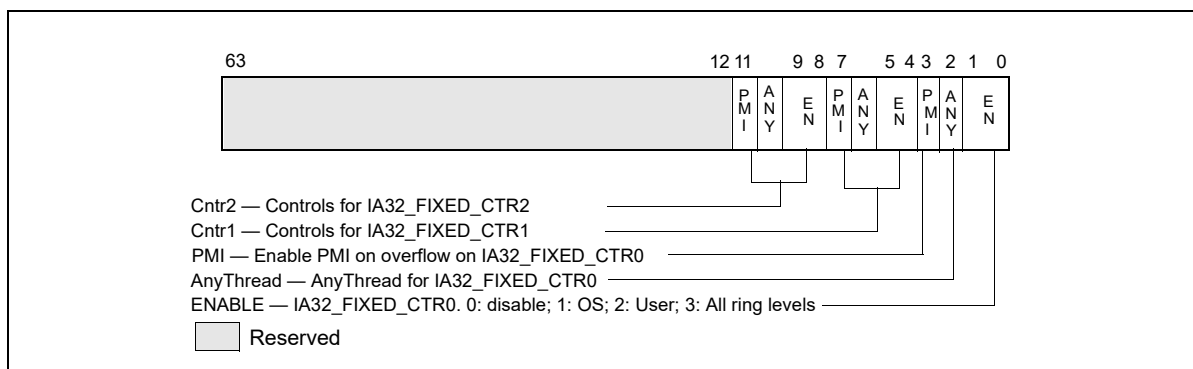


Figure 19-7. IA32_PERF_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

Each control block for a fixed-function performance counter provides an **AnyThread** (bit position $2 + 4*N$, $N=0, 1, \text{etc.}$) bit. When set to 1, it enables counting the associated event conditions (including matching the thread’s CPL with the ENABLE setting of the corresponding control block of IA32_PERF_FIXED_CTR_CTRL) occurring across all logical processors sharing a processor core. When an **AnyThread** bit is 0 in IA32_PERF_FIXED_CTR_CTRL, the corresponding fixed counter only increments the associated event conditions occurring in the logical processor which programmed the IA32_PERF_FIXED_CTR_CTRL MSR.

- The IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL MSRs provide single-bit controls/status for each general-purpose and fixed-function performance counter. Figure 19-8 and Figure 19-9 show the layout of these MSRs for N general-purpose performance counters (where N is reported by CPUID.0AH:EAX[15:8]) and three fixed-function counters.

19.2.3.1 AnyThread Counting and Software Evolution

The motivation for characterizing software workload over multiple software threads running on multiple logical processors of the same processor core originates from a time earlier than the introduction of the AnyThread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL. While AnyThread counting provides some benefits in simple software environments of an earlier era, the evolution contemporary software environments introduce certain concepts and pre-requisites that AnyThread counting does not comply with.

One example is the proliferation of software environments that support multiple virtual machines (VM) under VMX (see Chapter 23, “Introduction to Virtual-Machine Extensions”) where each VM represents a domain separated from one another.

A Virtual Machine Monitor (VMM) that manages the VMs may allow an individual VM to employ performance monitoring facilities to profiles the performance characteristics of a workload. The use of the Anythread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL is discouraged with software environments supporting virtualization or requiring domain separation.

Specifically, Intel recommends VMM:

- Configure the MSR bitmap to cause VM-exits for WRMSR to IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in VMX non-Root operation (see CHAPTER 24 for additional information),
- Clear the AnyThread bit of IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in the MSR-load lists for VM exits and VM entries (see CHAPTER 24, CHAPTER 26, and CHAPTER 27).

Even when operating in simpler legacy software environments which might not emphasize the pre-requisites of a virtualized software environment, the use of the AnyThread interface should be moderated and follow any event-specific guidance where explicitly noted.

19.2.4 Architectural Performance Monitoring Version 4

Processors supporting architectural performance monitoring version 4 also supports version 1, 2, and 3, as well as capability enumerated by CPUID leaf 0AH. Version 4 introduced a streamlined PMI overhead mitigation interface that replaces the legacy semantic behavior but retains the same control interface in IA32_DEBUGCTL.Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI. Specifically version 4 provides the following enhancements:

- New indicators (LBR_FRZ, CTR_FRZ) in IA32_PERF_GLOBAL_STATUS, see Section 19.2.4.1.
- Streamlined Freeze/PMI Overhead management interfaces to use IA32_DEBUGCTL.Freeze_LBRs_On_PMI and IA32_DEBUGCTL.Freeze_PerfMon_On_PMI: see Section 19.2.4.1. Legacy semantics of Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI (applicable to version 2 and 3) are not supported with version 4 or higher.
- Fine-grain separation of control interface to manage overflow/status of IA32_PERF_GLOBAL_STATUS and read-only performance counter enabling interface in IA32_PERF_GLOBAL_STATUS: see Section 19.2.4.2.
- Performance monitoring resource in-use MSR to facilitate cooperative sharing protocol between perfmon-managing privilege agents.

19.2.4.1 Enhancement in IA32_PERF_GLOBAL_STATUS

The IA32_PERF_GLOBAL_STATUS MSR provides the following indicators with architectural performance monitoring version 4:

- IA32_PERF_GLOBAL_STATUS.LBR_FRZ[bit 58]: This bit is set due to the following conditions:
 - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI has been set by the profiling agent, and
 - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently the LBR stack is frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.LBR_FRZ bit also serves as a control to enable capturing data in the LBR stack. To enable capturing LBR records, the following expression must hold with architectural perfmon version 4 or higher:

$$\text{— } (\text{IA32_DEBUGCTL.LBR} \ \& \ (\text{!IA32_PERF_GLOBAL_STATUS.LBR_FRZ})) = 1$$

- IA32_PERF_GLOBAL_STATUS.CTR_FRZ[bit 59]: This bit is set due to the following conditions:
 - IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI has been set by the profiling agent, and
 - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently, all the performance counters are frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.CTR_FRZ bit also serve as an read-only control to enable programmable performance counters and fixed counters in the core PMU. To enable counting with the performance counters, the following expression must hold with architectural perfmon version 4 or higher:

- $(IA32_PERFEVTSELn.EN \& IA32_PERF_GLOBAL_CTRL.PMCn \& (!IA32_PERF_GLOBAL_STATUS.CTR_FRZ)) = 1$ for programmable counter 'n', or
- $(IA32_PERF_FIXED_CTRL.ENi \& IA32_PERF_GLOBAL_CTRL.FCi \& (!IA32_PERF_GLOBAL_STATUS.CTR_FRZ)) = 1$ for fixed counter 'i'

The read-only enable interface IA32_PERF_GLOBAL_STATUS.CTR_FRZ provides a more efficient flow for a PMI handler to use IA32_DEBUGCTL.Freeze_Perfmon_On_PMI to filter out data that may distort target workload analysis, see Table 17-3. It should be noted the IA32_PERF_GLOBAL_CTRL register continue to serve as the primary interface to control all performance counters of the logical processor.

For example, when the Freeze-On-PMI mode is not being used, a PMI handler would be setting IA32_PERF_GLOBAL_CTRL as the very last step to commence the overall operation after configuring the individual counter registers, controls and PEBS facility. This does not only assure atomic monitoring but also avoids unnecessary complications (e.g., race conditions) when software attempts to change the core PMU configuration while some counters are kept enabled.

Additionally, IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55]: On processors that support Intel Processor Trace and configured to store trace output packets to physical memory using the ToPA scheme, bit 55 is set when a PMI occurred due to a ToPA entry memory buffer was completely filled.

IA32_PERF_GLOBAL_STATUS also provides an indicator to distinguish interaction of performance monitoring operations with other side-band activities, which apply Intel SGX on processors that support SGX (For additional information about Intel SGX, see the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D*):

- IA32_PERF_GLOBAL_STATUS.ASCI[bit 60]: This bit is set when data accumulated in any of the configured performance counters (i.e., IA32_PMCx or IA32_FIXED_CTRx) may include contributions from direct or indirect operation of Intel SGX to protect an enclave (since the last time IA32_PERF_GLOBAL_STATUS.ASCI was cleared).

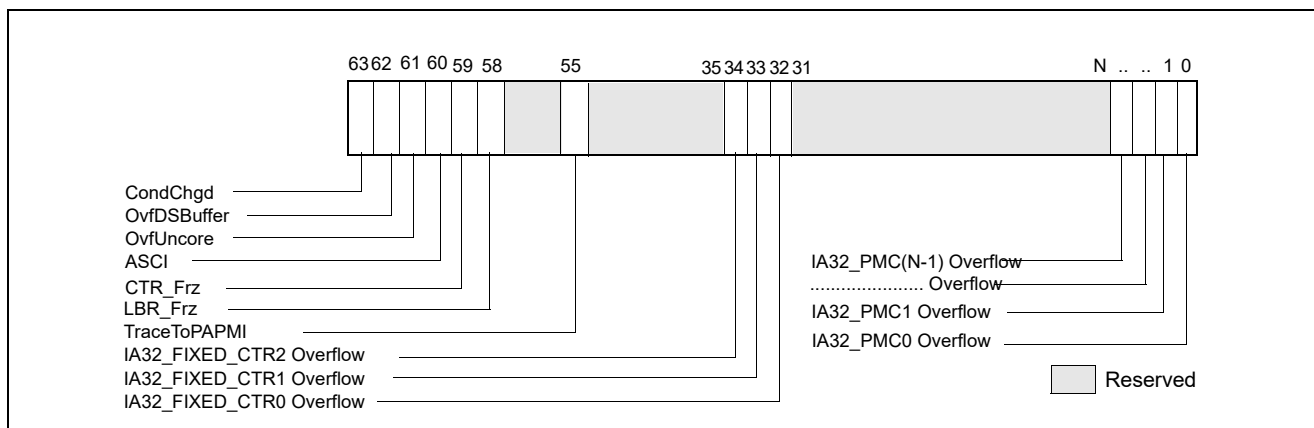


Figure 19-10. IA32_PERF_GLOBAL_STATUS MSR and Architectural Perfmon Version 4

Note, a processor’s support for IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55] is enumerated as a result of CPUID enumerated capability of Intel Processor Trace and the use of the ToPA buffer scheme. Support of IA32_PERF_GLOBAL_STATUS.ASCI[bit 60] is enumerated by the CPUID enumeration of Intel SGX.

19.2.4.2 IA32_PERF_GLOBAL_STATUS_RESET and IA32_PERF_GLOBAL_STATUS_SET MSRS

With architectural performance monitoring version 3 and lower, clearing of the set bits in IA32_PERF_GLOBAL_STATUS MSR by software is done via IA32_PERF_GLOBAL_OVF_CTRL MSR. Starting with architectural performance monitoring version 4, software can manage the overflow and other indicators in IA32_PERF_GLOBAL_STATUS using separate interfaces to set or clear individual bits.

The address and the architecturally-defined bits of IA32_PERF_GLOBAL_OVF_CTRL is inherited by IA32_PERF_GLOBAL_STATUS_RESET (see Figure 19-11). Further, IA32_PERF_GLOBAL_STATUS_RESET provides additional bit fields to clear the new indicators in IA32_PERF_GLOBAL_STATUS described in Section 19.2.4.1.

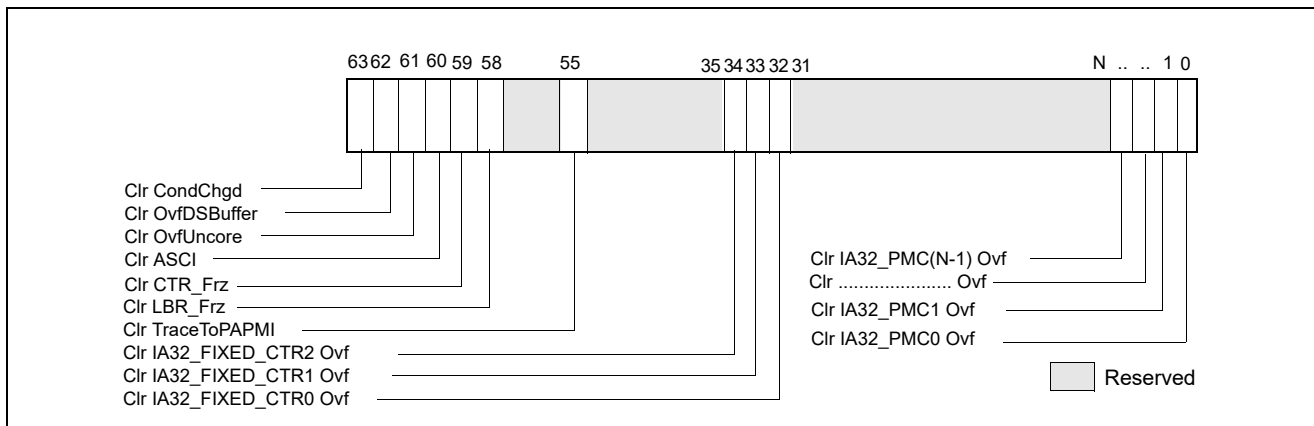


Figure 19-11. IA32_PERF_GLOBAL_STATUS_RESET MSR and Architectural Perfmon Version 4

The IA32_PERF_GLOBAL_STATUS_SET MSR is introduced with architectural performance monitoring version 4. It allows software to set individual bits in IA32_PERF_GLOBAL_STATUS. The IA32_PERF_GLOBAL_STATUS_SET interface can be used by a VMM to virtualize the state of IA32_PERF_GLOBAL_STATUS across VMs.

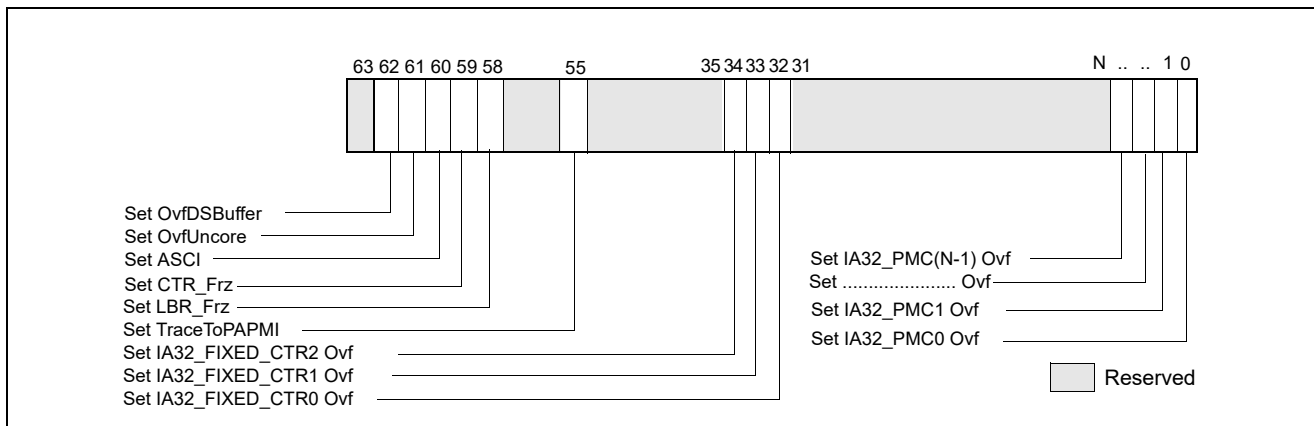


Figure 19-12. IA32_PERF_GLOBAL_STATUS_SET MSR and Architectural Perfmon Version 4

19.2.4.3 IA32_PERF_GLOBAL_INUSE MSR

In a contemporary software environment, multiple privileged service agents may wish to employ the processor’s performance monitoring facilities. The IA32_MISC_ENABLE.PERFMON_AVAILABLE[bit 7] interface could not serve

the need of multiple agent adequately. A white paper, “Performance Monitoring Unit Sharing Guideline”¹, proposed a cooperative sharing protocol that is voluntary for participating software agents.

Architectural performance monitoring version 4 introduces a new MSR, IA32_PERF_GLOBAL_INUSE, that simplifies the task of multiple cooperating agents to implement the sharing protocol.

The layout of IA32_PERF_GLOBAL_INUSE is shown in Figure 19-13.

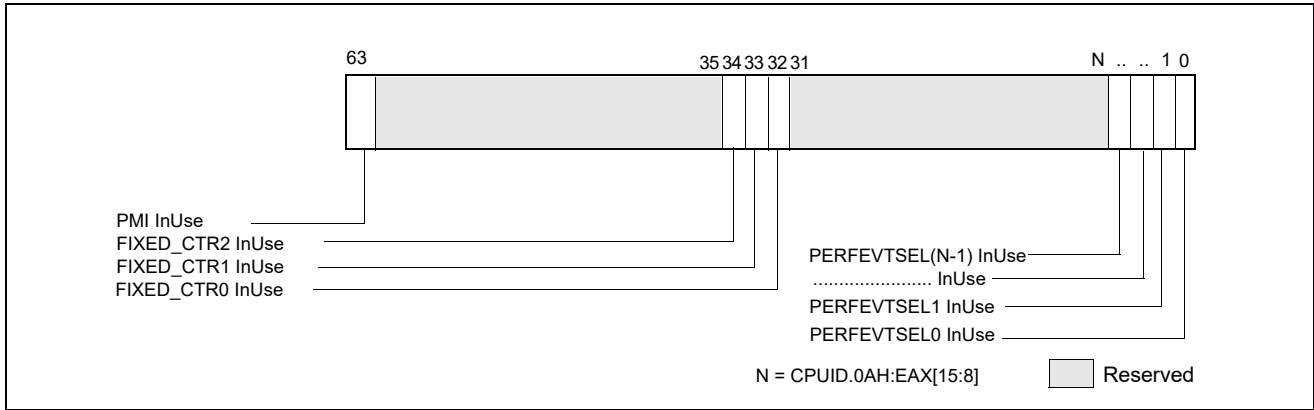


Figure 19-13. IA32_PERF_GLOBAL_INUSE MSR and Architectural Perfmon Version 4

The IA32_PERF_GLOBAL_INUSE MSR provides an “InUse” bit for each programmable performance counter and fixed counter in the processor. Additionally, it includes an indicator if the PMI mechanism has been configured by a profiling agent.

- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL0_InUse[bit 0]: This bit reflects the logical state of (IA32_PERFEVTSEL0[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL1_InUse[bit 1]: This bit reflects the logical state of (IA32_PERFEVTSEL1[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL2_InUse[bit 2]: This bit reflects the logical state of (IA32_PERFEVTSEL2[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSELn_InUse[bit n]: This bit reflects the logical state of (IA32_PERFEVTSELn[7:0] != 0), n < CPUID.0AH:EAX[15:8].
- IA32_PERF_GLOBAL_INUSE.FC0_InUse[bit 32]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[1:0] != 0).
- IA32_PERF_GLOBAL_INUSE.FC1_InUse[bit 33]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[5:4] != 0).
- IA32_PERF_GLOBAL_INUSE.FC2_InUse[bit 34]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[9:8] != 0).
- IA32_PERF_GLOBAL_INUSE.PMI_InUse[bit 63]: This bit is set if any one of the following bit is set:
 - IA32_PERFEVTSELn.INT[bit 20], n < CPUID.0AH:EAX[15:8].
 - IA32_FIXED_CTR_CTRL.ENi_PMI, i = 0, 1, 2.
 - Any IA32_PEBS_ENABLES bit which enables PEBS for a general-purpose or fixed-function performance counter.

1. Available at <http://www.intel.com/sdm>

19.2.5 Architectural Performance Monitoring Version 5

Processors supporting architectural performance monitoring version 5 also support versions 1, 2, 3 and 4, as well as capability enumerated by CPUID leaf 0AH. Specifically, version 5 provides the following enhancements:

- Deprecation of AnyThread mode, see Section 19.2.5.1.
- Individual enumeration of Fixed counters in CPUID.0AH, see Section 19.2.5.2.
- Domain separation, see Section 19.2.5.3.

19.2.5.1 AnyThread Mode Deprecation

With Architectural Performance Monitoring Version 5, a processor that supports AnyThread mode deprecation is enumerated by CPUID.0AH.EDX[15]. If set, software will not have to follow guidelines in Section 19.2.3.1.

19.2.5.2 Fixed Counter Enumeration

With Architectural Performance Monitoring Version 5, register CPUID.0AH.ECX indicates Fixed Counter enumeration. It is a bit mask which enumerates the supported Fixed Counters in a processor. If bit 'i' is set, it implies that Fixed Counter 'i' is supported. Software is recommended to use the following logic to check if a Fixed Counter is supported on a given processor:

```
FxCtr[i]_is_supported := ECX[i] || (EDX[4:0] > i);
```

19.2.5.3 Domain Separation

When the INV flag in IA32_PERFEVTSELx is used, a counter stops counting when the logical processor exits the C0 ACPI C-state.

19.2.6 Full-Width Writes to Performance Counter Registers

The general-purpose performance counter registers IA32_PMCx are writable via WRMSR instruction. However, the value written into IA32_PMCx by WRMSR is the signed extended 64-bit value of the EAX[31:0] input of WRMSR.

A processor that supports full-width writes to the general-purpose performance counters enumerated by CPUID.0AH:EAX[15:8] will set IA32_PERF_CAPABILITIES[13] to enumerate its full-width-write capability. See Figure 19-65.

If IA32_PERF_CAPABILITIES.FW_WRITE[bit 13] = 1, each IA32_PMCi is accompanied by a corresponding alias address starting at 4C1H for IA32_A_PMC0.

The bit width of the performance monitoring counters is specified in CPUID.0AH:EAX[23:16].

If IA32_A_PMCi is present, the 64-bit input value (EDX:EAX) of WRMSR to IA32_A_PMCi will cause IA32_PMCi to be updated by:

```
COUNTERWIDTH = CPUID.0AH:EAX[23:16] bit width of the performance monitoring counter
IA32_PMCi[COUNTERWIDTH-1:32] := EDX[COUNTERWIDTH-33:0];
IA32_PMCi[31:0] := EAX[31:0];
EDX[63:COUNTERWIDTH] are reserved
```

19.3 PERFORMANCE MONITORING (INTEL® CORE™ PROCESSORS AND INTEL® XEON® PROCESSORS)

19.3.1 Performance Monitoring for Processors Based on Nehalem Microarchitecture

Intel Core i7 processor family¹ supports architectural performance monitoring capability with version ID 3 (see Section 19.2.3) and a host of non-architectural monitoring capabilities. The Intel Core i7 processor family is based

on Nehalem microarchitecture, and provides four general-purpose performance counters (IA32_PMC0, IA32_PMC1, IA32_PMC2, IA32_PMC3) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2) in the processor core.

Non-architectural performance monitoring in Intel Core i7 processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>. Non-architectural performance monitoring events fall into two broad categories:

- Performance monitoring events in the processor core: These include many events that are similar to performance monitoring events available to processor based on Intel Core microarchitecture. Additionally, there are several enhancements in the performance monitoring capability for detecting microarchitectural conditions in the processor core or in the interaction of the processor core to the off-core sub-systems in the physical processor package. The off-core sub-systems in the physical processor package is loosely referred to as “uncore”.
- Performance monitoring events in the uncore: The uncore sub-system is shared by more than one processor cores in the physical processor package. It provides additional performance monitoring facility outside of IA32_PMCx and performance monitoring events that are specific to the uncore sub-system.

Architectural and non-architectural performance monitoring events in Intel Core i7 processor family support thread qualification using bit 21 of IA32_PERFEVTSELx MSR.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 19-6 and described in Section 19.2.1.1 and Section 19.2.3.

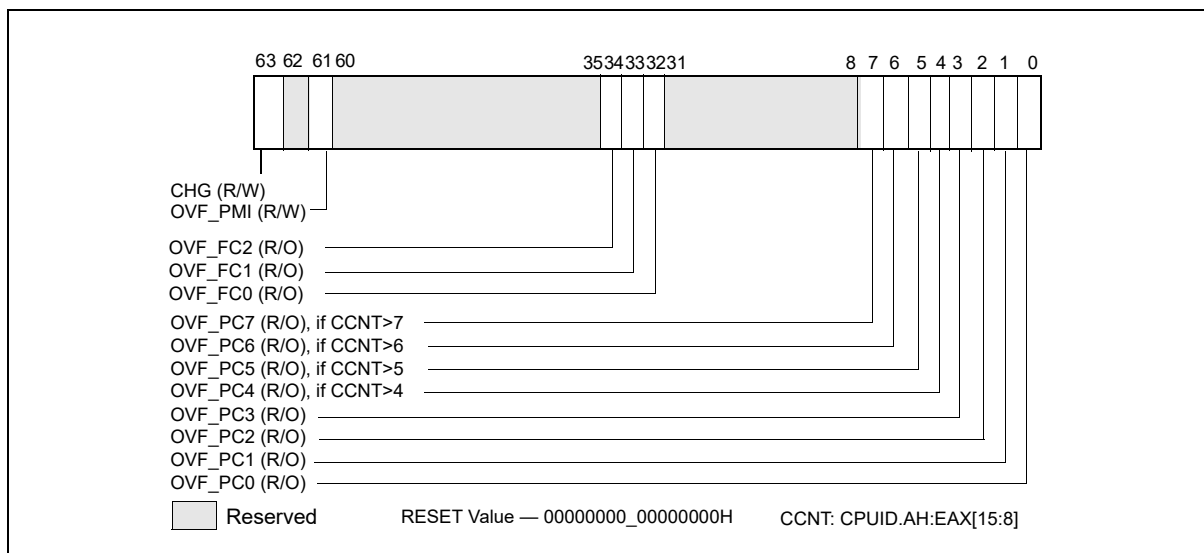


Figure 19-14. IA32_PERF_GLOBAL_STATUS MSR

19.3.1.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- Four general purpose performance counters, IA32_PMCx, associated counter configuration MSRs, IA32_PERFEVTSELx, and global counter control MSR supporting simplified control of four counters. Each of the four performance counter can support processor event based sampling (PEBS) and thread-qualification of architectural and non-architectural performance events. Width of IA32_PMCx supported by hardware has been increased. The width of counter reported by CPUID.0AH:EAX[23:16] is 48 bits. The PEBS facility in Nehalem

1. Intel Xeon processor 5500 series and 3400 series are also based on Nehalem microarchitecture; the performance monitoring facilities described in this section generally also apply.

microarchitecture has been enhanced to include new data format to capture additional information, such as load latency.

- Load latency sampling facility. Average latency of memory load operation can be sampled using load-latency facility in processors based on Nehalem microarchitecture. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches). This facility is used in conjunction with the PEBS facility.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSR's are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters. CPUID.0AH:EAX[15:8] reports the MSR's available to software; see Section 19.2.1.

19.3.1.1.1 Processor Event Based Sampling (PEBS)

All general-purpose performance counters, IA32_PMCx, can be used for PEBS if the performance event supports PEBS. Software uses IA32_MISC_ENABLE[7] and IA32_MISC_ENABLE[12] to detect whether the performance monitoring facility and PEBS functionality are supported in the processor. The MSR IA32_PEBS_ENABLE provides 4 bits that software must use to enable which IA32_PMCx overflow condition will cause the PEBS record to be captured.

Additionally, the PEBS record is expanded to allow latency information to be captured. The MSR IA32_PEBS_ENABLE provides 4 additional bits that software must use to enable latency data recording in the PEBS record upon the respective IA32_PMCx overflow condition. The layout of IA32_PEBS_ENABLE for processors based on Nehalem microarchitecture is shown in Figure 19-15.

When a counter is enabled to capture machine state (PEBS_EN_PMCx = 1), the processor will write machine state information to a memory buffer specified by software as detailed below. When the counter IA32_PMCx overflows from maximum count to zero, the PEBS hardware is armed.

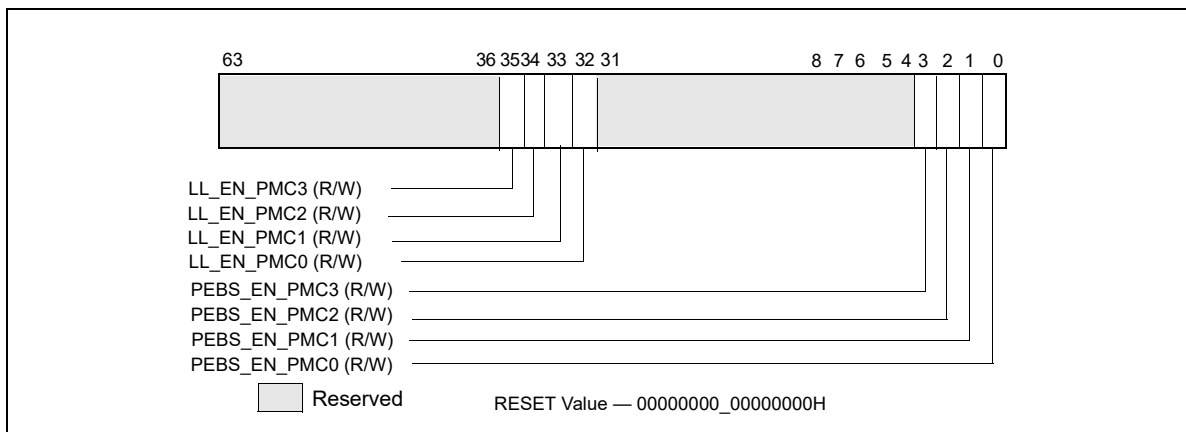


Figure 19-15. Layout of IA32_PEBS_ENABLE MSR

Upon occurrence of the next PEBS event, the PEBS hardware triggers an assist and causes a PEBS record to be written. The format of the PEBS record is indicated by the bit field IA32_PERF_CAPABILITIES[11:8] (see Figure 19-65).

The behavior of PEBS assists is reported by IA32_PERF_CAPABILITIES[6] (see Figure 19-65). The return instruction pointer (RIP) reported in the PEBS record will point to the instruction after (+1) the instruction that causes the PEBS assist. The machine state reported in the PEBS record is the machine state after the instruction that causes the PEBS assist is retired. For instance, if the instructions:

```
mov eax, [eax] ; causes PEBS assist
nop
```

are executed, the PEBS record will report the address of the nop, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation.

The PEBS record format is shown in Table 19-3, and each field in the PEBS record is 64 bits long. The PEBS record format, along with debug/store area storage format, does not change regardless of IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 19-3. PEBS Record Format for Intel Core i7 Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	58H	R9
08H	R/EIP	60H	R10
10H	R/EAX	68H	R11
18H	R/EBX	70H	R12
20H	R/ECX	78H	R13
28H	R/EDX	80H	R14
30H	R/ESI	88H	R15
38H	R/EDI	90H	IA32_PERF_GLOBAL_STATUS
40H	R/EBP	98H	Data Linear Address
48H	R/ESP	A0H	Data Source Encoding
50H	R8	A8H	Latency value (core cycles)

In IA-32e mode, the full 64-bit value is written to the register. If the processor is not operating in IA-32e mode, 32-bit value is written to registers with bits 63:32 zeroed. Registers not defined when the processor is not in IA-32e mode are written to zero.

Bytes AFH:90H are enhancement to the PEBS record format. Support for this enhanced PEBS record format is indicated by IA32_PERF_CAPABILITIES[11:8] encoding of 0001B.

The value written to bytes 97H:90H is the state of the IA32_PERF_GLOBAL_STATUS register before the PEBS assist occurred. This value is written so software can determine which counters overflowed when this PEBS record was written. Note that this field indicates the overflow status for all counters, regardless of whether they were programmed for PEBS or not.

Programming PEBS Facility

Only a subset of non-architectural performance events in the processor support PEBS. The subset of precise events are listed in Table 19-84. In addition to using IA32_PERFEVTSELx to specify event unit/mask settings and setting the EN_PMCx bit in the IA32_PEBS_ENABLE register for the respective counter, the software must also initialize the DS_BUFFER_MANAGEMENT_AREA data structure in memory to support capturing PEBS records for precise events.

The recording of PEBS records may not operate properly if accesses to the linear addresses in the DS buffer management area or in the PEBS buffer (see below) cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do "lazy" page-table entry propagation for these pages. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

The beginning linear address of the DS_BUFFER_MANAGEMENT_AREA data structure must be programmed into the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 19-16.

- **PEBS Buffer Base:** This field is programmed with the linear address of the first byte of the PEBS buffer allocated by software. The processor reads this field to determine the base address of the PEBS buffer.
- **PEBS Index:** This field is initially programmed with the same value as the PEBS Buffer Base field, or the beginning linear address of the PEBS buffer. The processor reads this field to determine the location of the next PEBS record to write to. After a PEBS record has been written, the processor also updates this field with the address of the next PEBS record to be written. The figure above illustrates the state of PEBS Index after the first PEBS record is written.
- **PEBS Absolute Maximum:** This field represents the absolute address of the maximum length of the allocated PEBS buffer plus the starting address of the PEBS buffer. The processor will not write any PEBS record beyond the end of PEBS buffer, when **PEBS Index** equals **PEBS Absolute Maximum**. No signaling is generated when PEBS buffer is full. Software must reset the **PEBS Index** field to the beginning of the PEBS buffer address to continue capturing PEBS records.

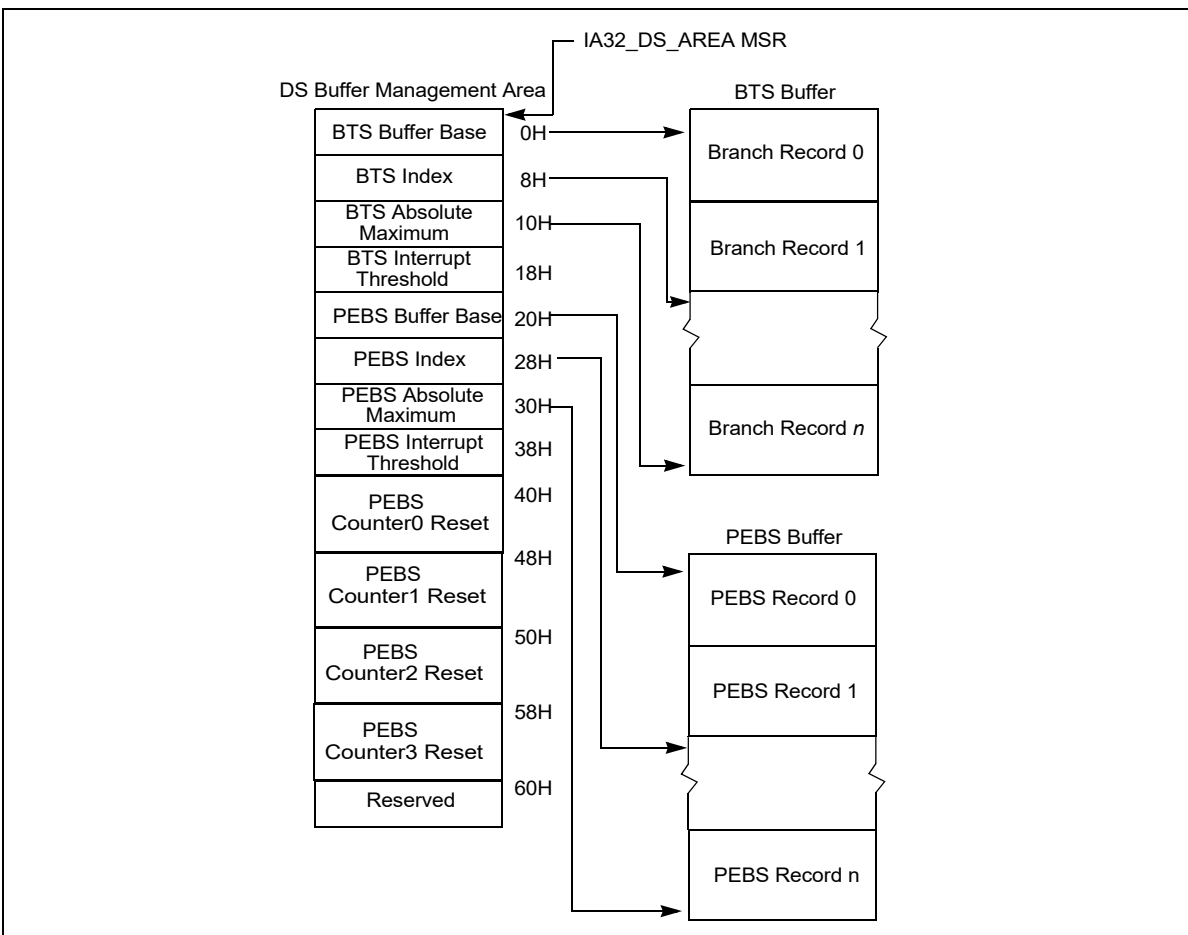


Figure 19-16. PEBS Programming Environment

- **PEBS Interrupt Threshold:** This field specifies the threshold value to trigger a performance interrupt and notify software that the PEBS buffer is nearly full. This field is programmed with the linear address of the first byte of the PEBS record within the PEBS buffer that represents the threshold record. After the processor writes a PEBS record and updates **PEBS Index**, if the **PEBS Index** reaches the threshold value of this field, the processor will generate a performance interrupt. This is the same interrupt that is generated by a performance counter overflow, as programmed in the Performance Monitoring Counters vector in the Local Vector Table of the Local APIC. When a performance interrupt due to PEBS buffer full is generated, the IA32_PERF_GLOBAL_STATUS.PEBS_Ovf bit will be set.
- **PEBS CounterX Reset:** This field allows software to set up PEBS counter overflow condition to occur at a rate useful for profiling workload, thereby generating multiple PEBS records to facilitate characterizing the profile the execution of test code. After each PEBS record is written, the processor checks each counter to see if it overflowed and was enabled for PEBS (the corresponding bit in IA32_PEBS_ENABLED was set). If these conditions are met, then the reset value for each overflowed counter is loaded from the DS Buffer Management Area. For example, if counter IA32_PMC0 caused a PEBS record to be written, then the value of "PEBS Counter 0 Reset" would be written to counter IA32_PMC0. If a counter is not enabled for PEBS, its value will not be modified by the PEBS assist.

Performance Counter Prioritization

Performance monitoring interrupts are triggered by a counter transitioning from maximum count to zero (assuming IA32_PerfEvtSelX.INT is set). This same transition will cause PEBS hardware to arm, but not trigger. PEBS hardware triggers upon detection of the first PEBS event after the PEBS hardware has been armed (a 0 to 1 transition of the counter). At this point, a PEBS assist will be undertaken by the processor.

Performance counters (fixed and general-purpose) are prioritized in index order. That is, counter IA32_PMC0 takes precedence over all other counters. Counter IA32_PMC1 takes precedence over counters IA32_PMC2 and IA32_PMC3, and so on. This means that if simultaneous overflows or PEBS assists occur, the appropriate action will be taken for the highest priority performance counter. For example, if IA32_PMC1 cause an overflow interrupt and IA32_PMC2 causes a PEBS assist simultaneously, then the overflow interrupt will be serviced first.

The PEBS threshold interrupt is triggered by the PEBS assist, and is by definition prioritized lower than the PEBS assist. Hardware will not generate separate interrupts for each counter that simultaneously overflows. General-purpose performance counters are prioritized over fixed counters.

If a counter is programmed with a precise (PEBS-enabled) event and programmed to generate a counter overflow interrupt, the PEBS assist is serviced before the counter overflow interrupt is serviced. If in addition the PEBS interrupt threshold is met, the

threshold interrupt is generated after the PEBS assist completes, followed by the counter overflow interrupt (two separate interrupts are generated).

Uncore counters may be programmed to interrupt one or more processor cores (see Section 19.3.1.2). It is possible for interrupts posted from the uncore facility to occur coincident with counter overflow interrupts from the processor core. Software must check core and uncore status registers to determine the exact origin of counter overflow interrupts.

19.3.1.1.2 Load Latency Performance Monitoring Facility

The load latency facility provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table 19-3. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_INST_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 100H). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.

- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001_00000001H.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The load-latency information written into a PEBS record (see Table 19-3, bytes AFH:98H) consists of:

- **Data Linear Address:** This is the linear address of the target of the load operation.
- **Latency Value:** This is the elapsed cycles of the tagged load operation between dispatch to GO, measured in processor core clock domain.
- **Data Source:** The encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 19-4. In the descriptions, local memory refers to system memory physically attached to a processor package, and remote memory refers to system memory physically attached to another processor package.

Table 19-4. Data Source Encoding for Load Latency Record

Encoding	Description
00H	Unknown L3 cache miss.
01H	Minimal latency core cache hit. This request was satisfied by the L1 data cache.
02H	Pending core cache HIT. Outstanding core cache miss to same cache-line address was already underway.
03H	This data request was satisfied by the L2.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found. (clean).
06H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found.
07H ¹	Reserved/LLC Snoop HitM. Local or Remote home requests that hit the last level cache and were serviced by another core with a cross core snoop where modified copies were found.
08H	Reserved/L3 MISS. Local homed requests that missed the L3 cache and were serviced by forwarded data following a cross package snoop where no modified copies were found. (Remote home requests are not counted).
09H	Reserved
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to shared state).
0CH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to exclusive state).
0DH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to exclusive state).
0EH	I/O, Request of input/output operation.
0FH	The request was to un-cacheable memory.

NOTES:

1. Bit 7 is supported only for processors with a CPUID DisplayFamily_DisplayModel signature of 06_2A, and 06_2E; otherwise it is reserved.

The layout of MSR_PEBS_LD_LAT_THRESHOLD is shown in Figure 19-17.

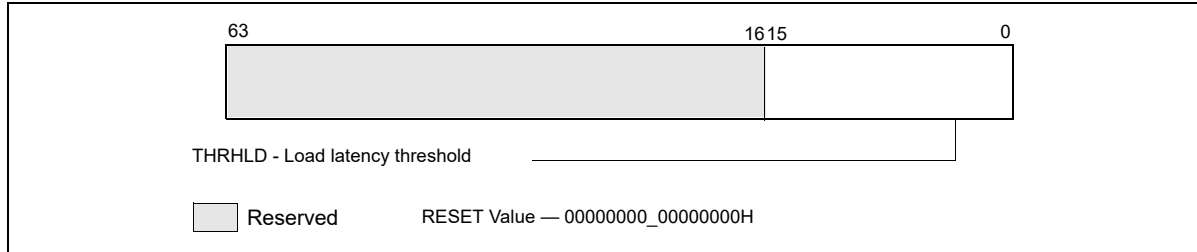


Figure 19-17. Layout of MSR_PEBS_LD_LAT MSR

Bits 15:0 specifies the threshold load latency in core clock cycles. Performance events with latencies greater than this value are counted in IA32_PMCx and their latency information is reported in the PEBS record. Otherwise, they are ignored. The minimum value that may be programmed in this field is 3.

19.3.1.1.3 Off-core Response Performance Monitoring in the Processor Core

Programming a performance event using the off-core response facility can choose any of the four IA32_PERFEVTSELx MSR with specific event codes and predefine mask bit value. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_0. There is only one off-core response configuration MSR. Table 19-5 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 19-5. Off-Core Response Event Encoding

Event code in IA32_PERFEVTSELx	Mask Value in IA32_PERFEVTSELx	Required Off-core Response MSR
B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)

The layout of MSR_OFFCORE_RSP_0 is shown in Figure 19-18. Bits 7:0 specifies the request type of a transaction request to the uncore. Bits 15:8 specifies the response of the uncore subsystem.

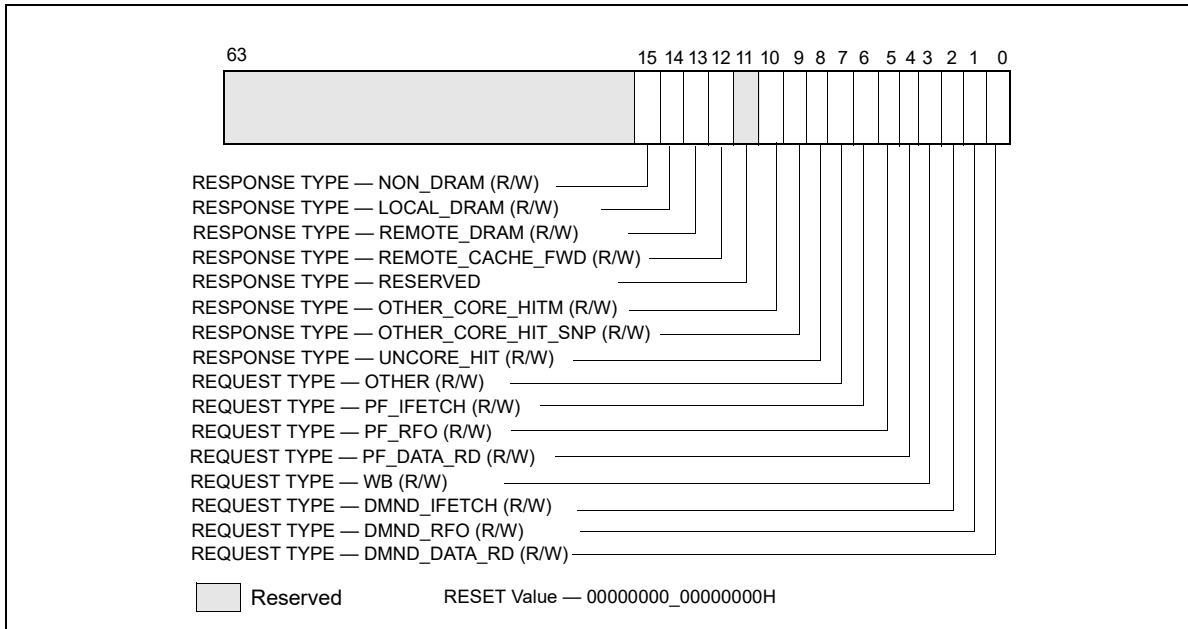


Figure 19-18. Layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 to Configure Off-core Response Events

Table 19-6. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
OTHER	7	Counts one of the following transaction types, including L3 invalidate, I/O, full or partial writes, WC or non-temporal stores, CLFLUSH, Fences, lock, unlock, split lock.
UNCORE_HIT	8	L3 Hit: local or remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
OTHER_CORE_HIT_SNP	9	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where no modified copies were found (clean).
OTHER_CORE_HITM	10	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where modified copies were found (HITM).
Reserved	11	Reserved
REMOTE_CACHE_FWD	12	L3 Miss: local homed requests that missed the L3 cache and was serviced by forwarded data following a cross package snoop where no modified copies found. (Remote home requests are not counted)
REMOTE_DRAM	13	L3 Miss: remote home requests that missed the L3 cache and were serviced by remote DRAM.
LOCAL_DRAM	14	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.

Table 19-6. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition (Contd.)

Bit Name	Offset	Description
NON_DRAM	15	Non-DRAM requests that were serviced by IOH.

19.3.1.2 Performance Monitoring Facility in the Uncore

The “uncore” in Nehalem microarchitecture refers to subsystems in the physical processor package that are shared by multiple processor cores. Some of the sub-systems in the uncore include the L3 cache, Intel QuickPath Interconnect link logic, and integrated memory controller. The performance monitoring facilities inside the uncore operates in the same clock domain as the uncore (U-clock domain), which is usually different from the processor core clock domain. The uncore performance monitoring facilities described in this section apply to Intel Xeon processor 5500 series and processors with the following CPUID signatures: 06_1AH, 06_1EH, 06_1FH (see Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*). An overview of the uncore performance monitoring facilities is described separately.

The performance monitoring facilities available in the U-clock domain consist of:

- Eight General-purpose counters (MSR_UNCORE_PerfCntr0 through MSR_UNCORE_PerfCntr7). The counters are 48 bits wide. Each counter is associated with a configuration MSR, MSR_UNCORE_PerfEvtSelx, to specify event code, event mask and other event qualification fields. A set of global uncore performance counter enabling/overflow/status control MSRs are also provided for software.
- Performance monitoring in the uncore provides an address/opcode match MSR that provides event qualification control based on address value or QPI command opcode.
- One fixed-function counter, MSR_UNCORE_FixedCntr0. The fixed-function uncore counter increments at the rate of the U-clock when enabled.

The frequency of the uncore clock domain can be determined from the uncore clock ratio which is available in the PCI configuration space register at offset C0H under device number 0 and Function 0.

19.3.1.2.1 Uncore Performance Monitoring Management Facility

MSR_UNCORE_PERF_GLOBAL_CTRL provides bit fields to enable/disable general-purpose and fixed-function counters in the uncore. Figure 19-19 shows the layout of MSR_UNCORE_PERF_GLOBAL_CTRL for an uncore that is shared by four processor cores in a physical package.

- EN_PCn (bit n, n = 0, 7): When set, enables counting for the general-purpose uncore counter MSR_UNCORE_PerfCntr n.
- EN_FC0 (bit 32): When set, enables counting for the fixed-function uncore counter MSR_UNCORE_FixedCntr0.
- EN_PMI_COREn (bit n, n = 0, 3 if four cores are present): When set, processor core n is programmed to receive an interrupt signal from any interrupt enabled uncore counter. PMI delivery due to an uncore counter overflow is enabled by setting IA32_DEBUGCTL.Offcore_PMI_EN to 1.
- PMI_FRZ (bit 63): When set, all U-clock uncore counters are disabled when any one of them signals a performance interrupt. Software must explicitly re-enable the counter by setting the enable bits in MSR_UNCORE_PERF_GLOBAL_CTRL upon exit from the ISR.

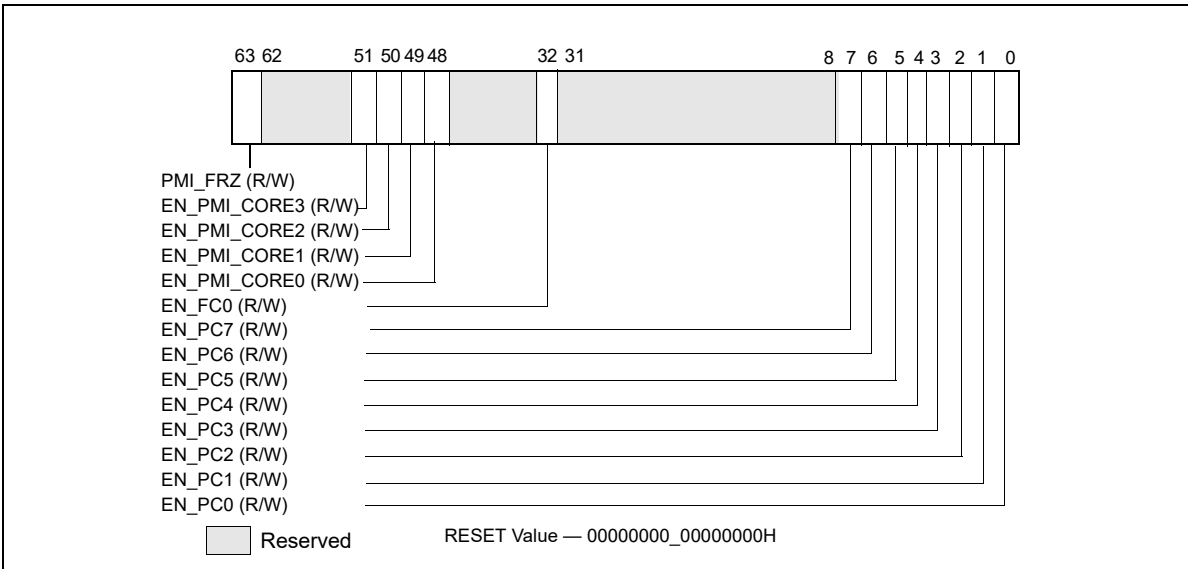


Figure 19-19. Layout of MSR_UNCORE_PERF_GLOBAL_CTRL MSR

MSR_UNCORE_PERF_GLOBAL_STATUS provides overflow status of the U-clock performance counters in the uncore. This is a read-only register. If an overflow status bit is set the corresponding counter has overflowed. The register provides a condition change bit (bit 63) which can be quickly checked by software to determine if a significant change has occurred since the last time the condition change status was cleared. Figure 19-20 shows the layout of MSR_UNCORE_PERF_GLOBAL_STATUS.

- OVF_PCn (bit n, n = 0, 7): When set, indicates general-purpose uncore counter MSR_UNCORE_PerfCntr n has overflowed.
- OVF_FC0 (bit 32): When set, indicates the fixed-function uncore counter MSR_UNCORE_FixedCntr0 has overflowed.
- OVF_PMI (bit 61): When set indicates that an uncore counter overflowed and generated an interrupt request.
- CHG (bit 63): When set indicates that at least one status bit in MSR_UNCORE_PERF_GLOBAL_STATUS register has changed state.

MSR_UNCORE_PERF_GLOBAL_OVF_CTRL allows software to clear the status bits in the UNCORE_PERF_GLOBAL_STATUS register. This is a write-only register, and individual status bits in the global status register are cleared by writing a binary one to the corresponding bit in this register. Writing zero to any bit position in this register has no effect on the uncore PMU hardware.

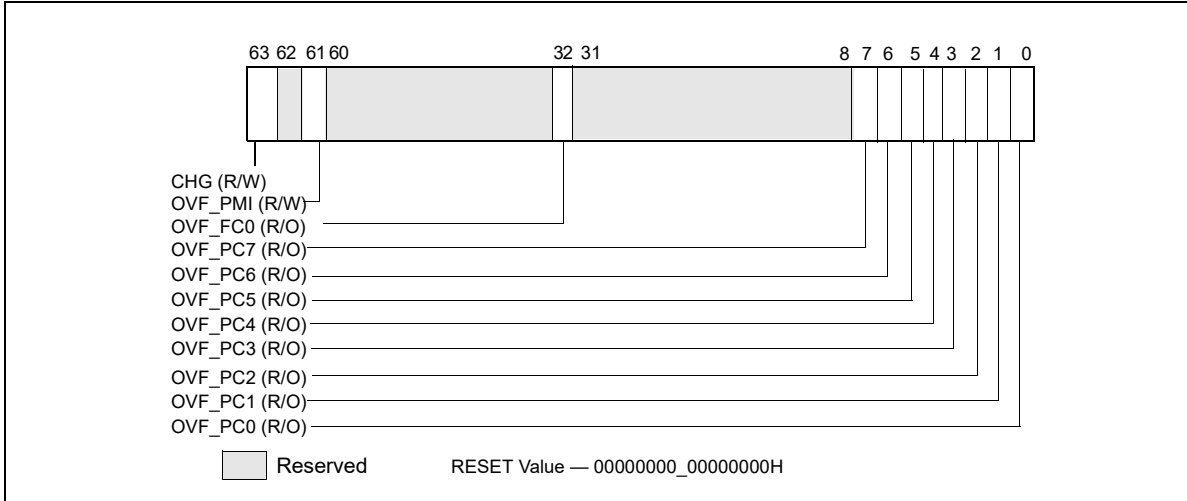


Figure 19-20. Layout of MSR_UNCORE_PERF_GLOBAL_STATUS MSR

Figure 19-21 shows the layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL.

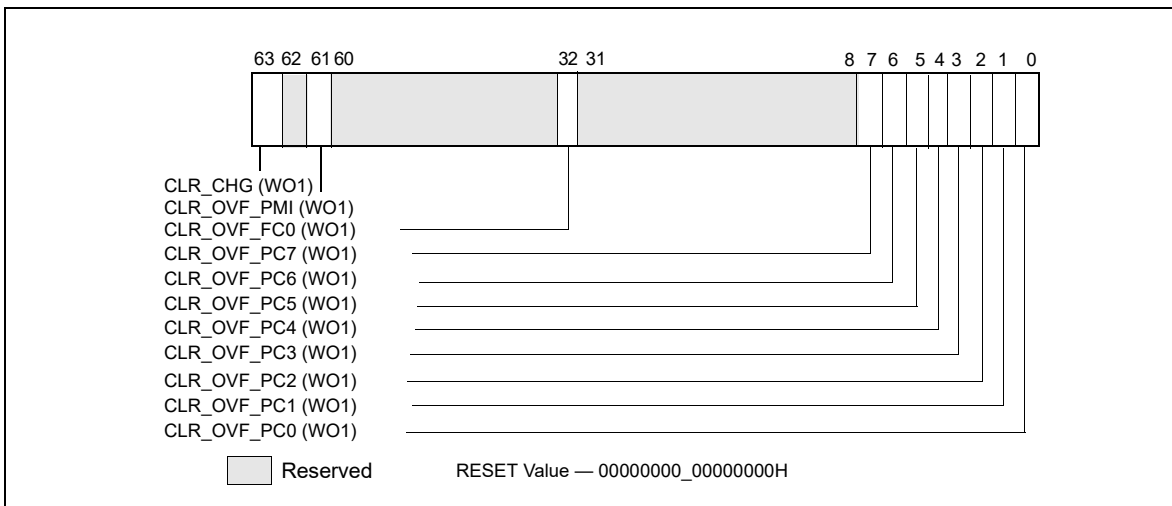


Figure 19-21. Layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL MSR

- CLR_OVF_PCn (bit n, n = 0, 7): Set this bit to clear the overflow status for general-purpose uncore counter MSR_UNCORE_PerfCntr n. Writing a value other than 1 is ignored.
- CLR_OVF_FC0 (bit 32): Set this bit to clear the overflow status for the fixed-function uncore counter MSR_UNCORE_FixedCntr0. Writing a value other than 1 is ignored.
- CLR_OVF_PMI (bit 61): Set this bit to clear the OVF_PMI flag in MSR_UNCORE_PERF_GLOBAL_STATUS. Writing a value other than 1 is ignored.
- CLR_CHG (bit 63): Set this bit to clear the CHG flag in MSR_UNCORE_PERF_GLOBAL_STATUS register. Writing a value other than 1 is ignored.

19.3.1.2.2 Uncore Performance Event Configuration Facility

MSR_UNCORE_PerfEvtSel0 through MSR_UNCORE_PerfEvtSel7 are used to select performance event and configure the counting behavior of the respective uncore performance counter. Each uncore PerfEvtSel MSR is paired with an uncore performance counter. Each uncore counter must be locally configured using the corre-

sponding MSR_UNCORE_PerEvtSelx and counting must be enabled using the respective EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL. Figure 19-22 shows the layout of MSR_UNCORE_PERFEVTSELx.

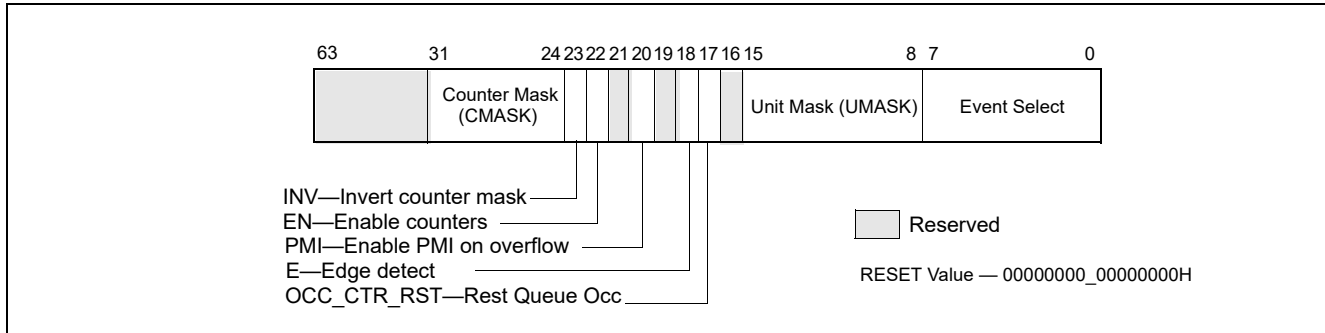


Figure 19-22. Layout of MSR_UNCORE_PERFEVTSELx MSRs

- Event Select (bits 7:0): Selects the event logic unit used to detect uncore events.
- Unit Mask (bits 15:8) : Condition qualifiers for the event selection logic specified in the Event Select field.
- OCC_CTR_RST (bit17): When set causes the queue occupancy counter associated with this event to be cleared (zeroed). Writing a zero to this bit will be ignored. It will always read as a zero.
- Edge Detect (bit 18): When set causes the counter to increment when a deasserted to asserted transition occurs for the conditions that can be expressed by any of the fields in this register.
- PMI (bit 20): When set, the uncore will generate an interrupt request when this counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.
- EN (bit 22): When clear, this counter is locally disabled. When set, this counter is locally enabled and counting starts when the corresponding EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- INV (bit 23): When clear, the Counter Mask field is interpreted as greater than or equal to. When set, the Counter Mask field is interpreted as less than.
- Counter Mask (bits 31:24): When this field is clear, it has no effect on counting. When set to a value other than zero, the logical processor compares this field to the event counts on each core clock cycle. If INV is clear and the event counts are greater than or equal to this field, the counter is incremented by one. If INV is set and the event counts are less than this field, the counter is incremented by one. Otherwise the counter is not incremented.

Figure 19-23 shows the layout of MSR_UNCORE_FIXED_CTR_CTRL.

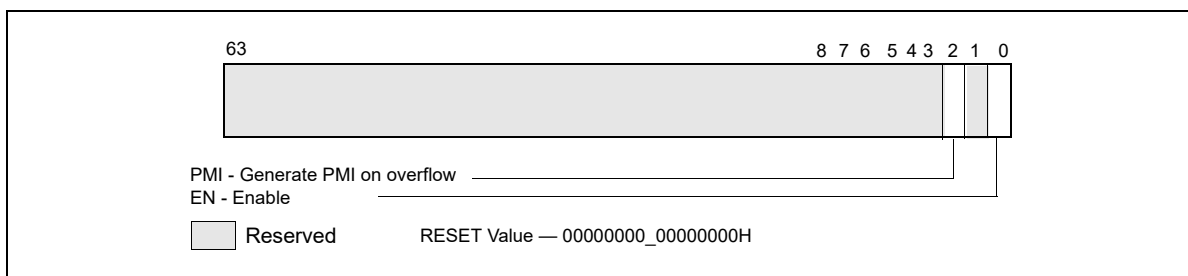


Figure 19-23. Layout of MSR_UNCORE_FIXED_CTR_CTRL MSR

- EN (bit 0): When clear, the uncore fixed-function counter is locally disabled. When set, it is locally enabled and counting starts when the EN_FC0 bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- PMI (bit 2): When set, the uncore will generate an interrupt request when the uncore fixed-function counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.

Both the general-purpose counters (MSR_UNCORE_PerfCnt) and the fixed-function counter (MSR_UNCORE_FixedCnt0) are 48 bits wide. They support both counting and interrupt based sampling usages. The event logic unit can filter event counts to specific regions of code or transaction types incoming to the home node logic.

19.3.1.2.3 Uncore Address/Opcode Match MSR

The Event Select field [7:0] of MSR_UNCORE_PERFEVTSELx is used to select different uncore event logic unit. When the event "ADDR_OPCODE_MATCH" is selected in the Event Select field, software can filter uncore performance events according to transaction address and certain transaction responses. The address filter and transaction response filtering requires the use of MSR_UNCORE_ADDR_OPCODE_MATCH register. The layout is shown in Figure 19-24.

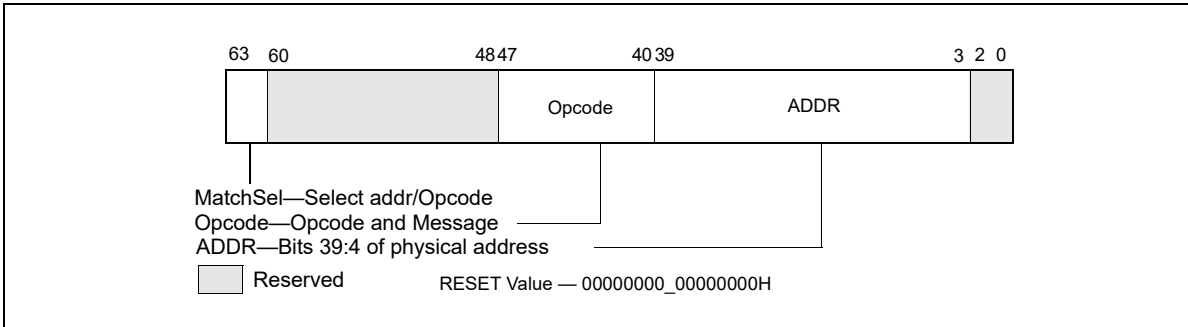


Figure 19-24. Layout of MSR_UNCORE_ADDR_OPCODE_MATCH MSR

- Addr (bits 39:3): The physical address to match if "MatchSel" field is set to select address match. The uncore performance counter will increment if the lowest 40-bit incoming physical address (excluding bits 2:0) for a transaction request matches bits 39:3.
- Opcode (bits 47:40) : Bits 47:40 allow software to filter uncore transactions based on QPI link message class/packed header opcode. These bits are consists two sub-fields:
 - Bits 43:40 specify the QPI packet header opcode.
 - Bits 47:44 specify the QPI message classes.

Table 19-7 lists the encodings supported in the opcode field.

Table 19-7. Opcode Field Encoding for MSR_UNCORE_ADDR_OPCODE_MATCH

Opcode [43:40]	QPI Message Class		
	Home Request [47:44] = 0000B	Snoop Response [47:44] = 0001B	Data Response [47:44] = 1110B
		1	
DMND_IFETCH	2	2	
WB	3	3	
PF_DATA_RD	4	4	
PF_RFO	5	5	
PF_IFETCH	6	6	
OTHER	7	7	
NON_DRAM	15	15	

- MatchSel (bits 63:61): Software specifies the match criteria according to the following encoding:
 - 000B: Disable addr_opcode match hardware.
 - 100B: Count if only the address field matches.
 - 010B: Count if only the opcode field matches.
 - 110B: Count if either opcode field matches or the address field matches.
 - 001B: Count only if both opcode and address field match.
 - Other encoding are reserved.

19.3.1.3 Intel® Xeon® Processor 7500 Series Performance Monitoring Facility

The performance monitoring facility in the processor core of Intel® Xeon® processor 7500 series are the same as those supported in Intel Xeon processor 5500 series. The uncore subsystem in Intel Xeon processor 7500 series are significantly different. The uncore performance monitoring facility consist of many distributed units associated with individual logic control units (referred to as boxes) within the uncore subsystem. A high level block diagram of the various box units of the uncore is shown in Figure 19-25.

Uncore PMUs are programmed via MSR interfaces. Each of the distributed uncore PMU units have several general-purpose counters. Each counter requires an associated event select MSR, and may require additional MSRs to configure sub-event conditions. The uncore PMU MSRs associated with each box can be categorized based on its functional scope: per-counter, per-box, or global across the uncore. The number counters available in each box type are different. Each box generally provides a set of MSRs to enable/disable, check status/overflow of multiple counters within each box.

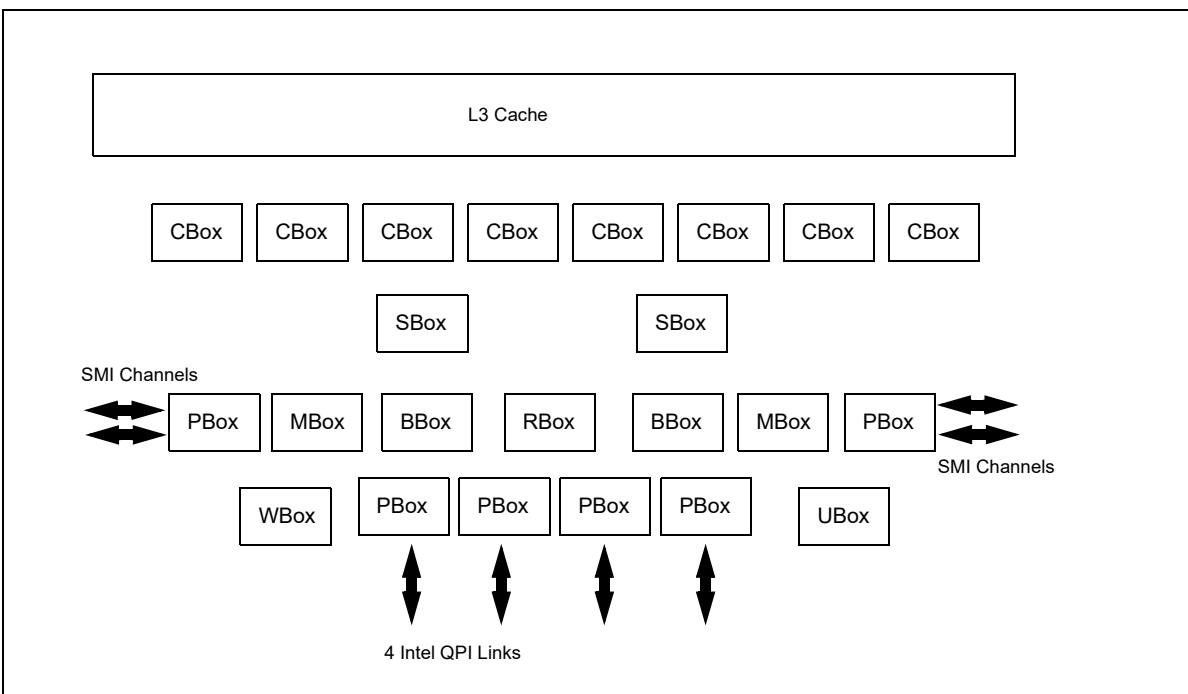


Figure 19-25. Distributed Units of the Uncore of Intel® Xeon® Processor 7500 Series

Table 19-8 summarizes the number MSRs for uncore PMU for each box.

Table 19-8. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

The W-Box provides 4 general-purpose counters, each requiring an event select configuration MSR, similar to the general-purpose counters in other boxes. There is also a fixed-function counter that increments clockticks in the uncore clock domain.

For C,S,B,M,R, and W boxes, each box provides an MSR to enable/disable counting, configuring PMI of multiple counters within the same box, this is somewhat similar the "global control" programming interface, IA32_PERF_GLOBAL_CTRL, offered in the core PMU. Similarly status information and counter overflow control for multiple counters within the same box are also provided in C,S,B,M,R, and W boxes.

In the U-Box, MSR_U_PMON_GLOBAL_CTL provides overall uncore PMU enable/disable and PMI configuration control. The scope of status information in the U-box is at per-box granularity, in contrast to the per-box status information MSR (in the C,S,B,M,R, and W boxes) providing status information of individual counter overflow. The difference in scope also apply to the overflow control MSR in the U-Box versus those in the other Boxes.

The individual MSRs that provide uncore PMU interfaces are listed in Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*, Table 2-17 under the general naming style of MSR_%box#%_PMON_%scope_function%, where %box#% designates the type of box and zero-based index if there are more the one box of the same type, %scope_function% follows the examples below:

- Multi-counter enabling MSRs: MSR_U_PMON_GLOBAL_CTL, MSR_S0_PMON_BOX_CTL, MSR_C7_PMON_BOX_CTL, etc.
- Multi-counter status MSRs: MSR_U_PMON_GLOBAL_STATUS, MSR_S0_PMON_BOX_STATUS, MSR_C7_PMON_BOX_STATUS, etc.
- Multi-counter overflow control MSRs: MSR_U_PMON_GLOBAL_OVF_CTL, MSR_S0_PMON_BOX_OVF_CTL, MSR_C7_PMON_BOX_OVF_CTL, etc.
- Performance counters MSRs: the scope is implicitly per counter, e.g., MSR_U_PMON_CTR, MSR_S0_PMON_CTR0, MSR_C7_PMON_CTR5, etc.
- Event select MSRs: the scope is implicitly per counter, e.g., MSR_U_PMON_EVNT_SEL, MSR_S0_PMON_EVNT_SEL0, MSR_C7_PMON_EVNT_SEL5, etc
- Sub-control MSRs: the scope is implicitly per-box granularity, e.g., MSR_M0_PMON_TIMESTAMP, MSR_R0_PMON_IPERF0_P1, MSR_S1_PMON_MATCH.

Details of uncore PMU MSR bit field definitions can be found in a separate document "Intel Xeon Processor 7500 Series Uncore Performance Monitoring Guide".

19.3.2 Performance Monitoring for Processors Based on Westmere Microarchitecture

All of the performance monitoring programming interfaces (architectural and non-architectural core PMU facilities, and uncore PMU) described in Section 19.6.3 also apply to processors based on Westmere microarchitecture.

Table 19-5 describes a non-architectural performance monitoring event (event code 0B7H) and associated MSR_OFFCORE_RSP_0 (address 1A6H) in the core PMU. This event and a second functionally equivalent offcore

response event using event code 0BBH and MSR_OFFCORE_RSP_1 (address 1A7H) are supported in processors based on Westmere microarchitecture. The event code and event mask definitions of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The load latency facility is the same as described in Section 19.3.1.1.2, but added enhancement to provide more information in the data source encoding field of each load latency record. The additional information relates to STLB_MISS and LOCK, see Table 19-13.

19.3.3 Intel® Xeon® Processor E7 Family Performance Monitoring Facility

The performance monitoring facility in the processor core of the Intel® Xeon® processor E7 family is the same as those supported in the Intel Xeon processor 5600 series¹. The uncore subsystem in the Intel Xeon processor E7 family is similar to those of the Intel Xeon processor 7500 series. The high level construction of the uncore subsystem is similar to that shown in Figure 19-25, with the additional capability that up to 10 C-Box units are supported.

Table 19-9 summarizes the number MSRs for uncore PMU for each box.

Table 19-9. Uncore PMU MSR Summary for Intel® Xeon® Processor E7 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	10	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E7 family is available in the “Intel® Xeon® Processor E7 Uncore Performance Monitoring Programming Reference Manual”.

19.3.4 Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture

Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel® Xeon® processor E3-1200 family are based on Sandy Bridge microarchitecture; this section describes the performance monitoring facilities provided in the processor core. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 19.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 19.2.3.

The core PMU’s capability is similar to those described in Section 19.3.1.1 and Section 19.6.3, with some differences and enhancements relative to Westmere microarchitecture summarized in Table 19-10.

1. Exceptions are indicated for event code 0FH in the event list for this processor (<https://perfmon-events.intel.com/>); and valid bits of data source encoding field of each load latency record is limited to bits 5:4 of Table 19-13.

Table 19-10. Core PMU Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 19.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W:32	See Section 19.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to determine # of counters. See Section 19.2.1.
PMI Overhead Mitigation	<ul style="list-style-type: none"> Freeze_Perfmon_on_PMI with legacy semantics. Freeze_LBR_on_PMI with legacy semantics for branch profiling. Freeze_while_SMM. 	<ul style="list-style-type: none"> Freeze_Perfmon_on_PMI with legacy semantics. Freeze_LBR_on_PMI with legacy semantics for branch profiling. Freeze_while_SMM. 	See Section 17.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 19-12.	See Table 19-84.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 19.3.4.4.2; <ul style="list-style-type: none"> Data source encoding STLB miss encoding Lock transaction encoding 	Data source encoding	
PEBS-Precise Store	Section 19.3.4.4.3	No	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL).	No	
Off-core Response Event	MSR 1A6H and 1A7H, extended request and response types.	MSR 1A6H and 1A7H, limited response types.	Nehalem supports 1A6H only.

19.3.4.1 Global Counter Control Facilities in Sandy Bridge Microarchitecture

The number of general-purpose performance counters visible to a logical processor can vary across Processors based on Sandy Bridge microarchitecture. Software must use CPUID to determine the number performance counters/event select registers (See Section 19.2.1.1.1).

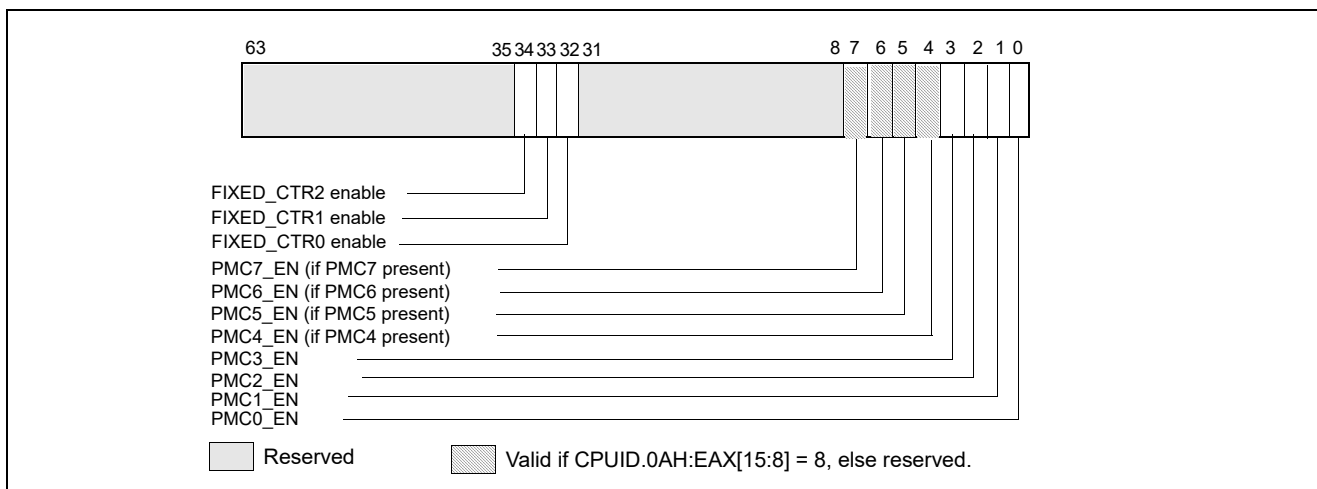


Figure 19-26. IA32_PERF_GLOBAL_CTRL MSR in Sandy Bridge Microarchitecture

Figure 19-44 depicts the layout of IA32_PERF_GLOBAL_CTRL MSR. The enable bits (PMC4_EN, PMC5_EN, PMC6_EN, PMC7_EN) corresponding to IA32_PMC4-IA32_PMC7 are valid only if CPUID.0AH:EAX[15:8] reports a value of '8'. If CPUID.0AH:EAX[15:8] = 4, attempts to set the invalid bits will cause #GP.

Each enable bit in IA32_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERF_FIXED_CTR_CTRL or IA32_PERF_FIXED_CTR_CTRL MSR to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

IA32_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. IA32_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer (see Figure 19-27). A value of 1 in each bit of the PMCx_OVF field indicates an overflow condition has occurred in the associated counter.

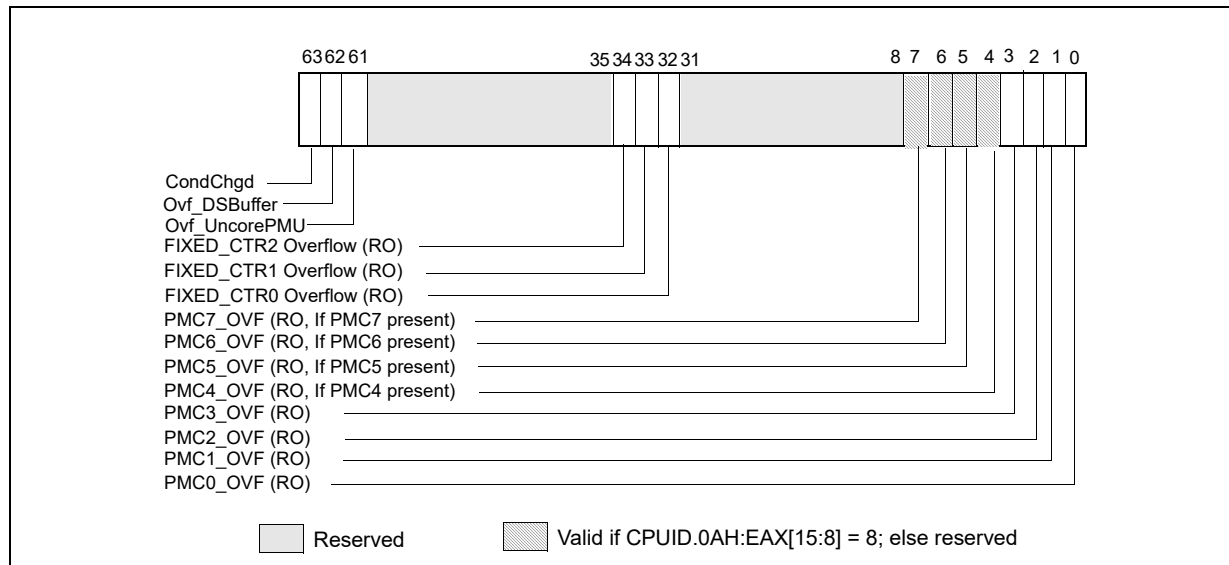


Figure 19-27. IA32_PERF_GLOBAL_STATUS MSR in Sandy Bridge Microarchitecture

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 17.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 19-28). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or interrupt based sampling.
- Reloading counter values to continue sampling.
- Disabling event counting or interrupt based sampling.

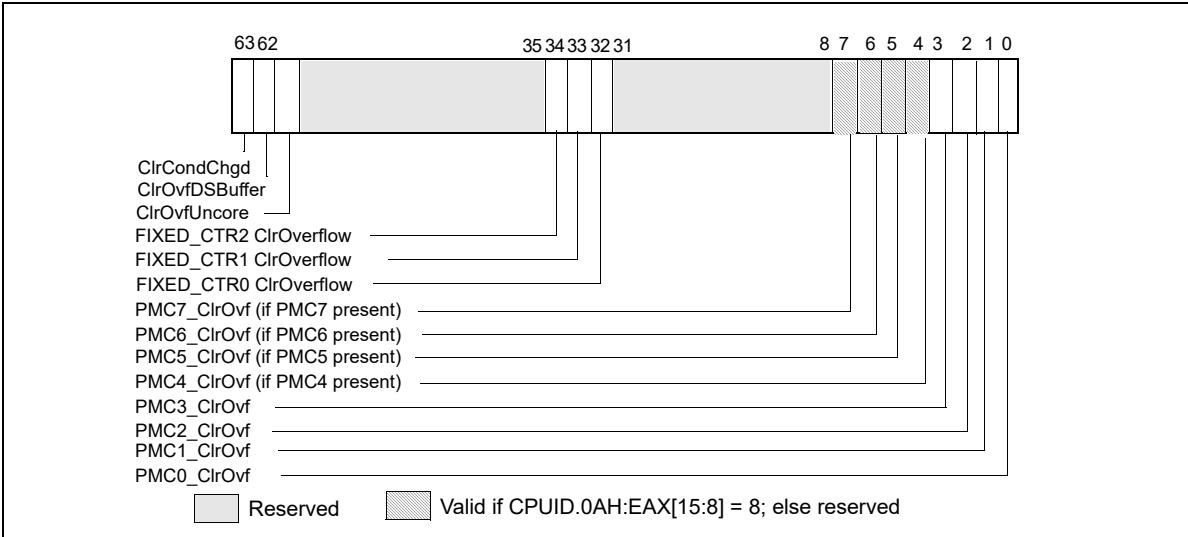


Figure 19-28. IA32_PERF_GLOBAL_OVF_CTRL MSR in Sandy Bridge Microarchitecture

19.3.4.2 Counter Coalescence

In processors based on Sandy Bridge microarchitecture, each processor core implements eight general-purpose counters. CPUID.0AH:EAX[15:8] will report the number of counters visible to software.

If a processor core is shared by two logical processors, each logical processors can access up to four counters (IA32_PMC0-IA32_PMC3). This is the same as in the prior generation for processors based on Nehalem microarchitecture.

If a processor core is not shared by two logical processors, up to eight general-purpose counters are visible. If CPUID.0AH:EAX[15:8] reports 8 counters, then IA32_PMC4-IA32_PMC7 would occupy MSR addresses 0C5H through 0C8H. Each counter is accompanied by an event select MSR (IA32_PERFEVTSEL4-IA32_PERFEVTSEL7).

If CPUID.0AH:EAX[15:8] report 4, access to IA32_PMC4-IA32_PMC7, IA32_PMC4-IA32_PMC7 will cause #GP. Writing 1's to bit position 7:4 of IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, or IA32_PERF_GLOBAL_OVF_CTL will also cause #GP.

19.3.4.3 Full Width Writes to Performance Counters

Processors based on Sandy Bridge microarchitecture support full-width writes to the general-purpose counters, IA32_PMCx. Support of full-width writes are enumerated by IA32_PERF_CAPABILITIES.FW_WRITES[13] (see Section 19.2.4).

The default behavior of IA32_PMCx is unchanged, i.e., WRMSR to IA32_PMCx results in a sign-extended 32-bit value of the input EAX written into IA32_PMCx. Full-width writes must issue WRMSR to a dedicated alias MSR address for each IA32_PMCx.

Software must check the presence of full-width write capability and the presence of the alias address IA32_A_PMCx by testing IA32_PERF_CAPABILITIES[13].

19.3.4.4 PEBS Support in Sandy Bridge Microarchitecture

Processors based on Sandy Bridge microarchitecture support PEBS, similar to those offered in prior generation, with several enhanced features. The key components and differences of PEBS facility relative to Westmere microarchitecture is summarized in Table 19-11.

Table 19-11. PEBS Facility Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 19.3.1.1.1	Section 19.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 19-29	Figure 19-15	
PEBS record layout	Physical Layout same as Table 19-3.	Table 19-3	Enhanced fields at offsets 98H, A0H, A8H.
PEBS Events	See Table 19-12.	See Table 19-84.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 19-13.	Table 19-4	
PEBS-Precise Store	Yes; see Section 19.3.4.4.3.	No	IA32_PMC3 only
PEBS-PDIR	Yes	No	IA32_PMC1 only
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

In IA32_PEBS_ENABLE MSR, bit 63 is defined as PS_ENABLE: When set, this enables IA32_PMC3 to capture precise store information. Only IA32_PMC3 supports the precise store facility. In typical usage of PEBS, the bit fields in IA32_PEBS_ENABLE are written to when the agent software starts PEBS operation; the enabled bit fields should be modified only when re-programming another PEBS event or cleared when the agent uses the performance counters for non-PEBS operations.

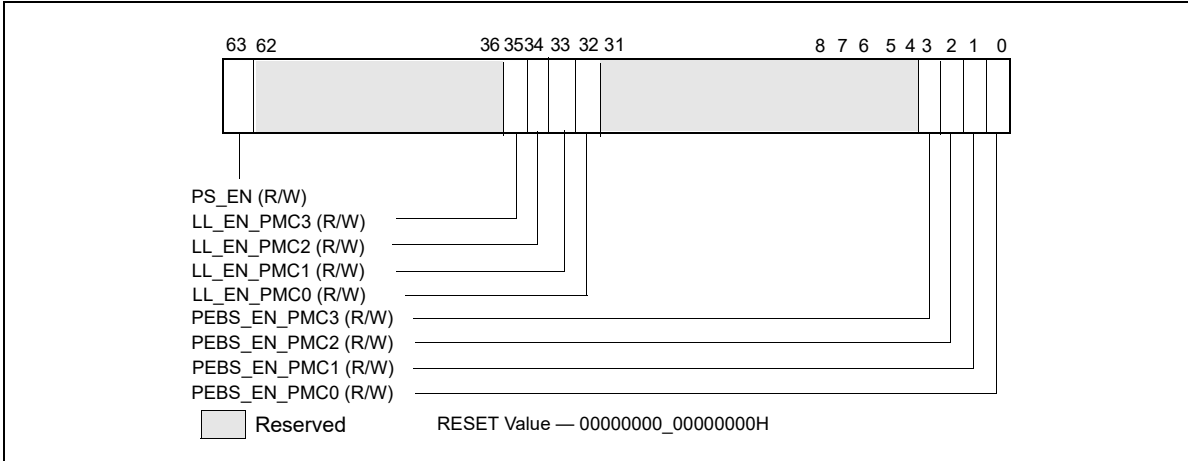


Figure 19-29. Layout of IA32_PEBS_ENABLE MSR

19.3.4.4.1 PEBS Record Format

The layout of PEBS records physically identical to those shown in Table 19-3, but the fields at offset 98H, A0H and A8H have been enhanced to support additional PEBS capabilities.

- Load/Store Data Linear Address (Offset 98H): This field will contain the linear address of the source of the load, or linear address of the destination of the store.
- Data Source /Store Status (Offset A0H): When load latency is enabled, this field will contain three piece of information (including an encoded value indicating the source which satisfied the load operation). The source field encodings are detailed in Table 19-4. When precise store is enabled, this field will contain information indicating the status of the store, as detailed in Table 19.
- Latency Value/0 (Offset A8H): When load latency is enabled, this field contains the latency in cycles to service the load. This field is not meaningful when precise store is enabled and will be written to zero in that case. Upon writing the PEBS record, microcode clears the overflow status bits in the IA32_PERF_GLOBAL_STATUS corresponding to those counters that both overflowed and were enabled in the IA32_PEBS_ENABLE register. The status bits of other counters remain unaffected.

The number PEBS events has expanded. The list of PEBS events supported in Sandy Bridge microarchitecture is shown in Table 19-12.

Table 19-12. PEBS Performance Events for Sandy Bridge Microarchitecture

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST	01H ¹
UOPS_RETIRED	C2H	All	01H
		Retire_Slots	02H
BR_INST_RETIRED	C4H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Near_Return	08H
		Near_Taken	20H
BR_MISP_RETIRED	C5H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Not_Taken	10H
		Taken	20H

Table 19-12. PEBS Performance Events for Sandy Bridge Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_UOPS_RETIRED	DOH	STLB_MISS_LOADS	11H
		STLB_MISS_STORE	12H
		LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H
MEM_LOAD_UOPS_RETIRED	D1H	L1_Hit	01H
		L2_Hit	02H
		L3_Hit	04H
		Hit_LFB	40H
MEM_LOAD_UOPS_LLC_HIT_RETIRED	D2H	XSNP_Miss	01H
		XSNP_Hit	02H
		XSNP_Hitm	04H
		XSNP_None	08H

NOTES:

1. Only available on IA32_PMC1.

19.3.4.4.2 Load Latency Performance Monitoring Facility

The load latency facility in Sandy Bridge microarchitecture is similar to that in prior microarchitectures. It provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table 19-3 and Section 19.3.4.4.1. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_TRANS_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001.00000001H.
- When Load latency event is enabled, no other PEBS event can be configured with other counters.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally. The MEM_TRANS_RETIRED event for load latency counts only tagged retired loads. If a load is cancelled it will not be counted and the internal state of the load latency facility will not be updated. In this case the hardware will tag the next available load.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The physical layout of the PEBS records is the same as shown in Table 19-3. The specificity of Data Source entry at offset A0H has been enhanced to report three pieces of information.

Table 19-13. Layout of Data Source Field of Load Latency Record

Field	Position	Description
Source	3:0	See Table 19-4
STLB_MISS	4	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
Lock	5	0: The load was not part of a locked transaction. 1: The load was part of a locked transaction.
Reserved	63:6	Reserved

The layout of MSR_PEBS_LD_LAT_THRESHOLD is the same as shown in Figure 19-17.

19.3.4.4.3 Precise Store Facility

Processors based on Sandy Bridge microarchitecture offer a precise store capability that complements the load latency facility. It provides a means to profile store memory references in the system.

Precise stores leverage the PEBS facility and provide additional information about sampled stores. Having precise memory reference events with linear address information for both loads and stores can help programmers improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

Only IA32_PMC3 can be used to capture precise store information. After enabling this facility, counter overflows will initiate the generation of PEBS records as previously described in PEBS. Upon counter overflow hardware captures the linear address and other status information of the next store that retires. This information is then written to the PEBS record.

To enable the precise store facility, software must complete the following steps. Please note that the precise store facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture precise store information.

- Complete the PEBS configuration steps.
- Program the MEM_TRANS_RETIRED.PRECISE_STORE event in IA32_PERFVTSEL3. Only counter 3 (IA32_PMC3) supports collection of precise store information.
- Set IA32_PEBS_ENABLE[3] and IA32_PEBS_ENABLE[63]. This enables IA32_PMC3 as a PEBS counter and enables the precise store facility, respectively.

The precise store information written into a PEBS record affects entries at offset 98H, A0H and A8H of Table 19-3. The specificity of Data Source entry at offset A0H has been enhanced to report three piece of information.

Table 19-14. Layout of Precise Store Information In PEBS Record

Field	Offset	Description
Store Data Linear Address	98H	The linear address of the destination of the store.
Store Status	A0H	L1D Hit (Bit 0): The store hit the data cache closest to the core (lowest latency cache) if this bit is set, otherwise the store missed the data cache. STLB Miss (bit 4): The store missed the STLB if set, otherwise the store hit the STLB Locked Access (bit 5): The store was part of a locked access if set, otherwise the store was not part of a locked access.
Reserved	A8H	Reserved

19.3.4.4.4 Precise Distribution of Instructions Retired (PDIR)

Upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. INST_RETIREED is a very common event that is used to sample where performance bottleneck happened and to help identify its location in instruction address space. Even if the delay is constant in core clock space, it invariably manifest as variable “skids” in instruction address space. This creates a challenge for programmers to profile a workload and pinpoint the location of bottlenecks.

The core PMU in processors based on Sandy Bridge microarchitecture include a facility referred to as precise distribution of Instruction Retired (PDIR).

The PDIR facility mitigates the “skid” problem by providing an early indication of when the INST_RETIREED counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow. On processors based on Sandy Bridge microarchitecture, skid is significantly reduced and can be as little as one instruction. On future implementations, PDIR may eliminate skid.

PDIR applies only to the INST_RETIREED.ALL precise event, and processors based on Sandy Bridge microarchitecture must use IA32_PMC1 with PerfEvtSel1 property configured and bit 1 in the IA32_PEBS_ENABLE set to 1. INST_RETIREED.ALL is a non-architectural performance event, it is not supported in prior generation microarchitectures. Additionally, on processors with CPUID DisplayFamily_DisplayModel signatures of 06_2A and 06_2D, the tool that programs PDIR should quiesce the rest of the programmable counters in the core when PDIR is active.

19.3.4.5 Off-core Response Performance Monitoring

The core PMU in processors based on Sandy Bridge microarchitecture provides off-core response facility similar to prior generation. Off-core response can be programmed only with a specific pair of event select and counter MSR, and with specific event codes and predefine mask bit value in a dedicated MSR to specify attributes of the off-core transaction. Two event codes are dedicated for off-core response event programming. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Table 19-15 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 19-15. Off-Core Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMCO-3	B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)
PMCO-3	BBH	01H	MSR_OFFCORE_RSP_1 (address 1A7H)

The layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 are shown in Figure 19-30 and Figure 19-31. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

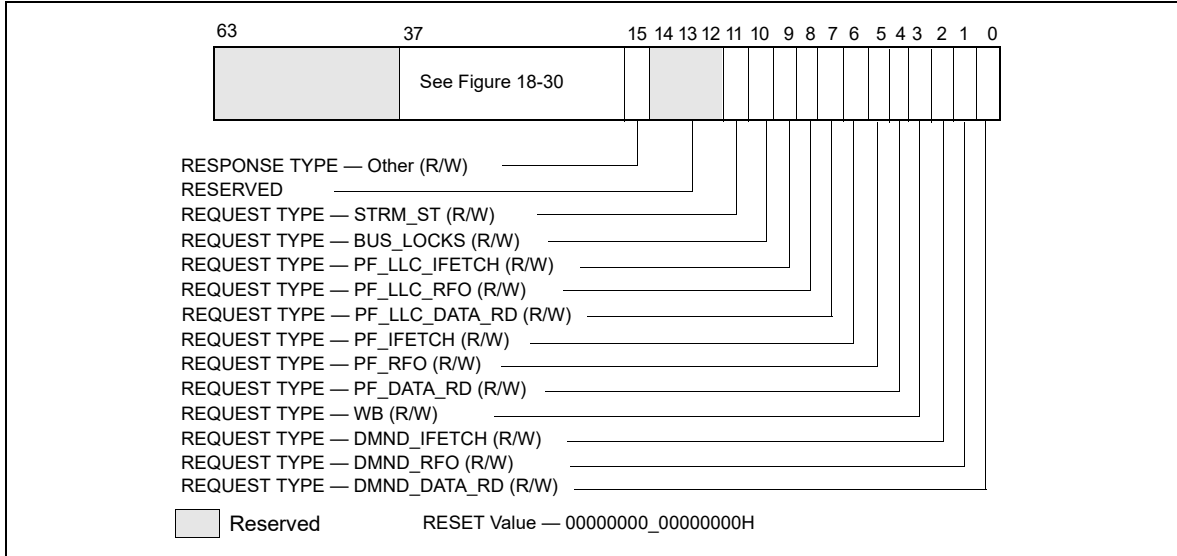


Figure 19-30. Request_Type Fields for MSR_OFFCORE_RSP_x

Table 19-16. MSR_OFFCORE_RSP_x Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_LLC_DATA_RD	7	L2 prefetcher to L3 for loads.
PF_LLC_RFO	8	RFO requests generated by L2 prefetcher
PF_LLC_IFETCH	9	L2 prefetcher to L3 for instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests
STRM_ST	11	Streaming store requests
OTHER	15	Any other request that crosses IDI, including I/O.

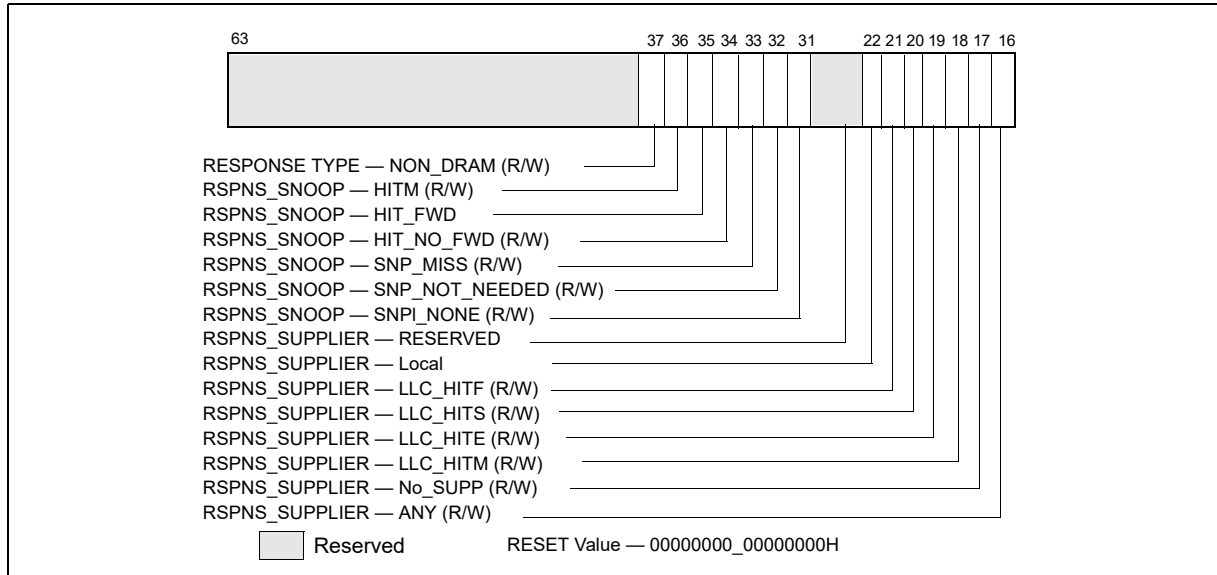


Figure 19-31. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSP_x

To properly program this extra register, software must set at least one request type bit and a valid response type pattern. Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSP_x allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 19-17. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 19-18. MSR_OFFCORE_RSP_x Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	SNP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNP_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO) -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD) -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S) In the LLC Miss case, data is returned from DRAM.
	SNP_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD) -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO) -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

19.3.4.6 Uncore Performance Monitoring Facilities In Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series

The uncore sub-system in Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series provides a unified L3 that can support up to four processor cores. The L3 cache consists multiple slices, each slice interface with a processor via a coherence engine, referred to as a C-Box. Each C-Box provides dedicated facility of MSRs to select uncore performance monitoring events and each C-Box event select MSR is paired with a counter register, similar in style as those described in Section 19.3.1.2.2. The ARB unit in the uncore also provides its local performance counters and event select MSRs. The layout of the event select MSRs in the C-Boxes and the ARB unit are shown in Figure 19-32.

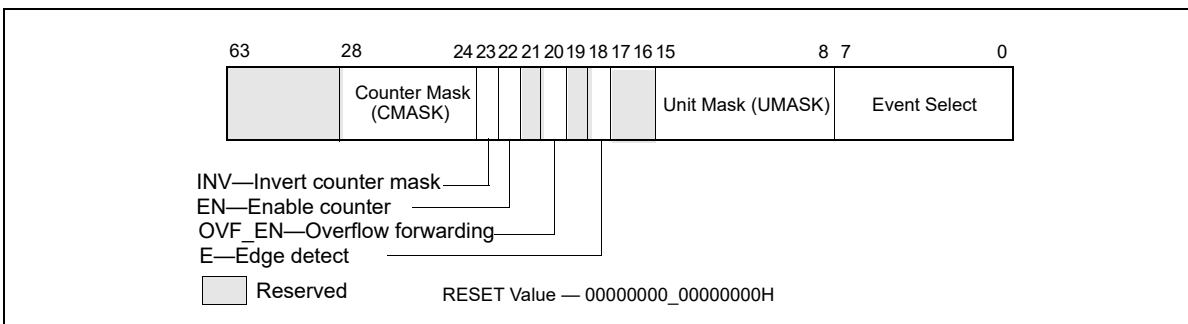


Figure 19-32. Layout of Uncore PERFVTSSEL MSR for a C-Box Unit or the ARB Unit

The bit fields of the uncore event select MSRs for a C-box unit or the ARB unit are summarized below:

- Event_Select (bits 7:0) and UMASK (bits 15:8): Specifies the microarchitectural condition to count in a local uncore PMU counter, see the event list at: <https://perfmon-events.intel.com/>.
- E (bit 18): Enables edge detection filtering, if 1.
- OVF_EN (bit 20): Enables the overflow indicator from the uncore counter forwarded to MSR_UNC_PERF_GLOBAL_CTRL, if 1.
- EN (bit 22): Enables the local counter associated with this event select MSR.
- INV (bit 23): Event count increments with non-negative value if 0, with negated value if 1.
- CMASK (bits 28:24): Specifies a positive threshold value to filter raw event count input.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 19-33 shows the layout of the uncore domain global control.

When an uncore counter overflows, a PMI can be routed to a processor core. Bits 3:0 of MSR_UNC_PERF_GLOBAL_CTRL can be used to select which processor core to handle the uncore PMI. Software must then write to bit 13 of IA32_DEBUGCTL (at address 1D9H) to enable this capability.

- PMI_SEL_Core#: Enables the forwarding of an uncore PMI request to a processor core, if 1. If bit 30 (WakePMI) is '1', a wake request is sent to the respective processor core prior to sending the PMI.
- EN: Enables the fixed uncore counter, the ARB counters, and the CBO counters in the uncore PMU, if 1. This bit is cleared if bit 31 (FREEZE) is set and any enabled uncore counters overflow.
- WakePMI: Controls sending a wake request to any halted processor core before issuing the uncore PMI request. If a processor core was halted and not sent a wake request, the uncore PMI will not be serviced by the processor core.
- FREEZE: Provides the capability to freeze all uncore counters when an overflow condition occurs in a unit counter. When this bit is set, and a counter overflow occurs, the uncore PMU logic will clear the global enable bit (bit 29).

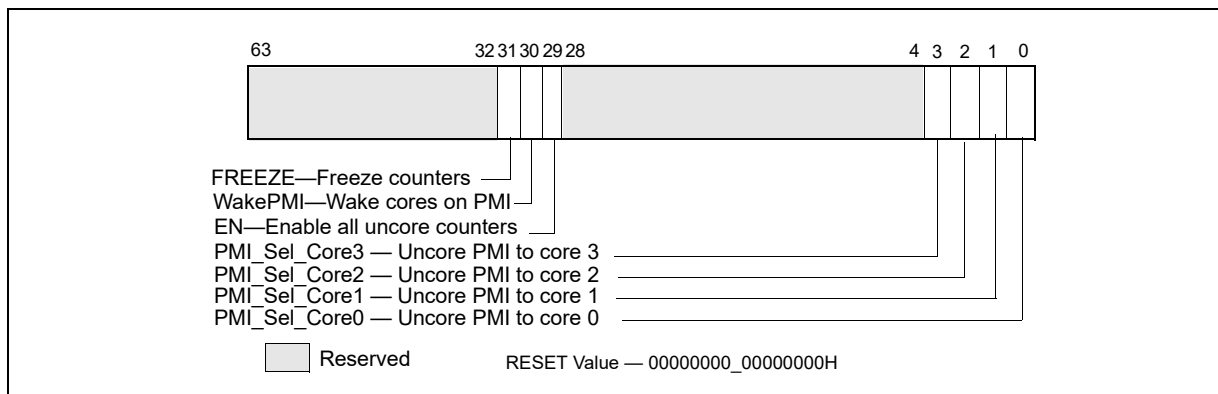


Figure 19-33. Layout of MSR_UNC_PERF_GLOBAL_CTRL MSR for Uncore

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 19-19 summarizes the number MSRs for uncore PMU for each box.

Table 19-19. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, see Table 2-21 MSR_UNC_CBO_CONFIG
ARB	1	2	44	Yes	Uncore	

Table 19-19. Uncore PMU MSR Summary (Contd.)

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
Fixed Counter	N.A.	N.A.	48	No	Uncore	

19.3.4.6.1 Uncore Performance Monitoring Events

There are certain restrictions on the uncore performance counters in each C-Box. Specifically,

- Occupancy events are supported only with counter 0 but not counter 1.
- Other uncore C-Box events can be programmed with either counter 0 or 1.

The C-Box uncore performance events can collect performance characteristics of transactions initiated by processor core. In that respect, they are similar to various sub-events in the OFFCORE_RESPONSE family of performance events in the core PMU. Information such as data supplier locality (LLC HIT/MISS) and snoop responses can be collected via OFFCORE_RESPONSE and qualified on a per-thread basis.

On the other hand, uncore performance event logic cannot associate its counts with the same level of per-thread qualification attributes as the core PMU events can. Therefore, whenever similar event programming capabilities are available from both core PMU and uncore PMU, the recommendation is that utilizing the core PMU events may be less affected by artifacts, complex interactions and other factors.

19.3.4.7 Intel® Xeon® Processor E5 Family Performance Monitoring Facility

The Intel® Xeon® Processor E5 Family (and Intel® Core™ i7-3930K Processor) are based on Sandy Bridge-E micro-architecture. While the processor cores share the same microarchitecture as those of the Intel® Xeon® Processor E3 Family and 2nd generation Intel Core i7-2xxx, Intel Core i5-2xxx, Intel Core i3-2xxx processor series, the uncore subsystems are different. An overview of the uncore performance monitoring facilities of the Intel Xeon processor E5 family (and Intel Core i7-3930K processor) is described in Section 19.3.4.8.

Thus, the performance monitoring facilities in the processor core generally are the same as those described in Section 19.6.3 through Section 19.3.4.5. However, the MSR_OFFCORE_RSP_0/MSR_OFFCORE_RSP_1 Response Supplier Info field shown in Table 19-17 applies to Intel Core Processors with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2AH; Intel Xeon processor with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2DH supports an additional field for remote DRAM controller shown in Table 19-20. Additionally, there are some small differences in the non-architectural performance monitoring events (see event list available at: <https://perfmon-events.intel.com/>).

Table 19-20. MSR_OFFCORE_RSP_x Supplier Info Field Definitions

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Remote	30:23	Remote DRAM Controller (either all 0s or all 1s).

19.3.4.8 Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5-2600 product family has some similarities with those of the Intel Xeon processor E7 family. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope. For example, each Cbox caching agent has a set of local performance counters, and the power controller unit (PCU) has its own local performance counters. Up to 8 C-Box units are supported in the uncore sub-system.

Table 19-21 summarizes the uncore PMU facilities providing MSR interfaces.

Table 19-21. Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	4	44	Yes	per-box	None
PCU	1	4	48	Yes	per-box	Match/Mask
U-Box	1	2	44	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 family is available in “Intel® Xeon® Processor E5 Uncore Performance Monitoring Programming Reference Manual”. The MSR-based uncore PMU interfaces are listed in Table 2-24.

19.3.5 3rd Generation Intel® Core™ Processor Performance Monitoring Facility

The 3rd generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v2 product family are based on the Ivy Bridge microarchitecture. The performance monitoring facilities in the processor core generally are the same as those described in Section 19.6.3 through Section 19.3.4.5. The non-architectural performance monitoring events supported by the processor core can be found at: <https://perfmon-events.intel.com/>.

19.3.5.1 Intel® Xeon® Processor E5 v2 and E7 v2 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5 v2 and Intel Xeon Processor E7 v2 product families are based on the Ivy Bridge-E microarchitecture. There are some similarities with those of the Intel Xeon processor E5 family based on the Sandy Bridge microarchitecture. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope.

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v2 and Intel Xeon Processor E7 v2 families are available in the “Intel® Xeon® Processor E5 v2 and E7 v2 Uncore Performance Monitoring Programming Reference Manual”. The MSR-based uncore PMU interfaces are listed in Table 2-28.

19.3.6 4th Generation Intel® Core™ Processor Performance Monitoring Facility

The 4th generation Intel® Core™ processor and Intel® Xeon® processor E3-1200 v3 product family are based on the Haswell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 19.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 19.2.3.

The core PMU’s capability is similar to those described in Section 19.6.3 through Section 19.3.4.5, with some differences and enhancements summarized in Table 19-22. Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 19.3.6.5. For details of Intel TSX, see Chapter 16, “Programming with Intel® Transactional Synchronization Extensions” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

Table 19-22. Core PMU Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 19.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 19.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 19.2.1.
PMI Overhead Mitigation	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_LBR_on_PMI with legacy semantics for branch profiling. ▪ Freeze_while_SMM. 	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_LBR_on_PMI with legacy semantics for branch profiling. ▪ Freeze_while_SMM. 	See Section 17.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 19-12 and Section 19.3.6.5.1.	See Table 19-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 19.3.4.4.2.	See Section 19.3.4.4.2.	
PEBS-Precise Store	No, replaced by Data Address profiling.	Section 19.3.4.4.3	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL)	Yes (using precise INST_RETIRED.ALL)	
PEBS-EventingIP	Yes	No	
Data Address Profiling	Yes	No	
LBR Profiling	Yes	Yes	
Call Stack Profiling	Yes, see Section 17.11.	No	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; extended request and response types.	MSR 1A6H and 1A7H; extended request and response types.	
Intel TSX support for Perfmon	See Section 19.3.6.5.	No	

19.3.6.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 4th Generation Intel Core processor is similar to those in processors based on Sandy Bridge microarchitecture, with several enhanced features. The key components and differences of PEBS facility relative to Sandy Bridge microarchitecture is summarized in Table 19-23.

Table 19-23. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
Valid IA32_PMCx	PMCO-PMC3	PMCO-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 19.3.1.1.1	Section 19.3.1.1.1	Unchanged
IA32_PEBES_ENABLE Layout	Figure 19-15	Figure 19-29	
PEBS record layout	Table 19-24; enhanced fields at offsets 98H, A0H, A8H, B0H.	Table 19-3; enhanced fields at offsets 98H, A0H, A8H.	

Table 19-23. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
Precise Events	See Table 19-12.	See Table 19-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 19-13.	Table 19-13	
PEBS-Precise Store	No, replaced by data address profiling.	Yes; see Section 19.3.4.4.3.	
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

19.3.6.2 PEBS Data Format

The PEBS record format for the 4th Generation Intel Core processor is shown in Table 19-24. The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 19-24. PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	Data Linear Address
40H	R/EBP	A0H	Data Source Encoding
48H	R/ESP	A8H	Latency value (core cycles)
50H	R8	B0H	EventingIP
58H	R9	B8H	TX Abort Information (Section 19.3.6.5.1)

The layout of PEBS records are almost identical to those shown in Table 19-3. Offset B0H is a new field that records the eventing IP address of the retired instruction that triggered the PEBS assist.

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 19.3.4.4.2), PDIR (Section 19.3.4.4.4), and the equivalent capability of precise store in prior generation (see Section 19.3.6.3).

In the core PMU of the 4th generation Intel Core processor, load latency facility and PDIR capabilities are unchanged. However, precise store is replaced by an enhanced capability, data address profiling, that is not restricted to store address. Data address profiling also records information in PEBS records at offsets 98H, A0H, and ABH.

19.3.6.3 PEBS Data Address Profiling

The Data Linear Address facility is also abbreviated as DataLA. The facility is a replacement or extension of the precise store facility in previous processor generations. The DataLA facility complements the load latency facility by providing a means to profile load and store memory references in the system, leverages the PEBS facility, and provides additional information about sampled loads and stores. Having precise memory reference events with linear address information for both loads and stores provides information to improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

The DataLA facility in the 4th generation processor supports the following events configured to use PEBS:

Table 19-25. Precise Events That Supports Data Linear Address Profiling

Event Name	Event Name
MEM_UOPS_RETIRED.STLB_MISS_LOADS	MEM_UOPS_RETIRED.STLB_MISS_STORES
MEM_UOPS_RETIRED.LOCK_LOADS	MEM_UOPS_RETIRED.SPLIT_STORES
MEM_UOPS_RETIRED.SPLIT_LOADS	MEM_UOPS_RETIRED.ALL_STORES
MEM_UOPS_RETIRED.ALL_LOADS	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM
MEM_LOAD_UOPS_RETIRED.L1_HIT	MEM_LOAD_UOPS_RETIRED.L2_HIT
MEM_LOAD_UOPS_RETIRED.L3_HIT	MEM_LOAD_UOPS_RETIRED.L1_MISS
MEM_LOAD_UOPS_RETIRED.L2_MISS	MEM_LOAD_UOPS_RETIRED.L3_MISS
MEM_LOAD_UOPS_RETIRED.HIT_LFB	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM
UOPS_RETIRED.ALL (if load or store is tagged)	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE

DataLA can use any one of the IA32_PMC0-IA32_PMC3 counters. Counter overflows will initiate the generation of PEBS records. Upon counter overflow, hardware captures the linear address and possible other status information of the retiring memory uop. This information is then written to the PEBS record that is subsequently generated.

To enable the DataLA facility, software must complete the following steps. Please note that the DataLA facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture DataLA information.

- Complete the PEBS configuration steps.
- Program an event listed in Table 19-25 using any one of IA32_PERFEVTSEL0-IA32_PERFEVTSEL3.
- Set the corresponding IA32_PEBS_ENABLE.PEBS_EN_CTRx bit. This enables the corresponding IA32_PMCx as a PEBS counter and enables the DataLA facility.

When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H and A8H, as shown in Table 19-26.

Table 19-26. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	<ul style="list-style-type: none"> ▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRE.ALL (if store is tagged), MEM_UOPS_RETIRE.STLB_MISS_STORES, MEM_UOPS_RETIRE.SPLIT_STORES, MEM_UOPS_RETIRE.ALL_STORES ▪ Other bits are zero, The STLB_MISS, LOCK bit information can be obtained by programming the corresponding store event in Table 19-25.
Reserved	A8H	Always zero.

19.3.6.3.1 EventingIP Record

The PEBS record layout for processors based on Haswell microarchitecture adds a new field at offset 0B0H. This is the eventingIP field that records the IP address of the retired instruction that triggered the PEBS assist. The EIP/RIP field at offset 08H records the IP address of the next instruction to be executed following the PEBS assist.

19.3.6.4 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 19.3.4.5. The event codes are listed in Table 19-15. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 19-27.
- Supplier information (bits 30:16): see Table 19-28.
- Snoop response information (bits 37:31): see Table 19-18.

Table 19-27. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture)

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
COREWB	3	Counts the number of modified cachelines written back.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_L3_DATA_RD	7	Counts the number of data cacheline reads generated by L3 prefetchers.
PF_L3_RFO	8	Counts the number of RFO requests generated by L3 prefetchers.
PF_L3_CODE_RD	9	Counts the number of code reads generated by L3 prefetchers.
SPLIT_LOCK_UC_LOCK	10	Counts the number of lock requests that split across two cachelines or are to UC memory.
STRM_ST	11	Counts the number of streaming store requests electronically.
Reserved	14:12	Reserved

Table 19-27. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture) (Contd.)

Bit Name	Offset	Description
OTHER	15	Any other request that crosses IDI, including I/O.

The supplier information field listed in Table 19-28. The fields vary across products (according to CPUID signatures) and is noted in the description.

Table 19-28. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_3CH, 06_46H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

Table 19-29. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_45H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	L4_HIT_LOCAL_L4	22	L4 Cache
	L4_HIT_REMOTE_HOP0_L4	23	L4 Cache
	L4_HIT_REMOTE_HOP1_L4	24	L4 Cache
	L4_HIT_REMOTE_HOP2P_L4	25	L4 Cache
Reserved	30:26	Reserved	

19.3.6.4.1 Off-core Response Performance Monitoring in Intel Xeon Processors E5 v3 Series

Table 19-28 lists the supplier information field that apply to Intel Xeon processor E5 v3 series (CPUID signature 06_3FH).

Table 19-30. MSR_OFFCORE_RSP_x Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	L3_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	26:23	Reserved
	L3_MISS_REMOTE_HOP0	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

19.3.6.5 Performance Monitoring and Intel® TSX

Chapter 16 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* describes the details of Intel® Transactional Synchronization Extensions (Intel® TSX). This section describes performance monitoring support for Intel TSX.

If a processor supports Intel TSX, the core PMU enhances its IA32_PERFEVTSELx MSR with two additional bit fields for event filtering. Support for Intel TSX is indicated by either (a) CPUID.(EAX=7, ECX=0):RTM[bit 11]=1, or (b) if CPUID.07H.EBX.HLE [bit 4] = 1. The TSX-enhanced layout of IA32_PERFEVTSELx is shown in Figure 19-34. The two additional bit fields are:

- **IN_TX** (bit 32): When set, the counter will only include counts that occurred inside a transactional region, regardless of whether that region was aborted or committed. This bit may only be set if the processor supports HLE or RTM.
- **IN_TXCP** (bit 33): When set, the counter will not include counts that occurred inside of an aborted transactional region. This bit may only be set if the processor supports HLE or RTM. This bit may only be set for IA32_PERFEVTSEL2.

When the IA32_PERFEVTSELx MSR is programmed with both IN_TX=0 and IN_TXCP=0 on a processor that supports Intel TSX, the result in a counter may include detectable conditions associated with a transaction code region for its aborted execution (if any) and completed execution.

In the initial implementation, software may need to take pre-caution when using the IN_TXCP bit. See Table 2-29.

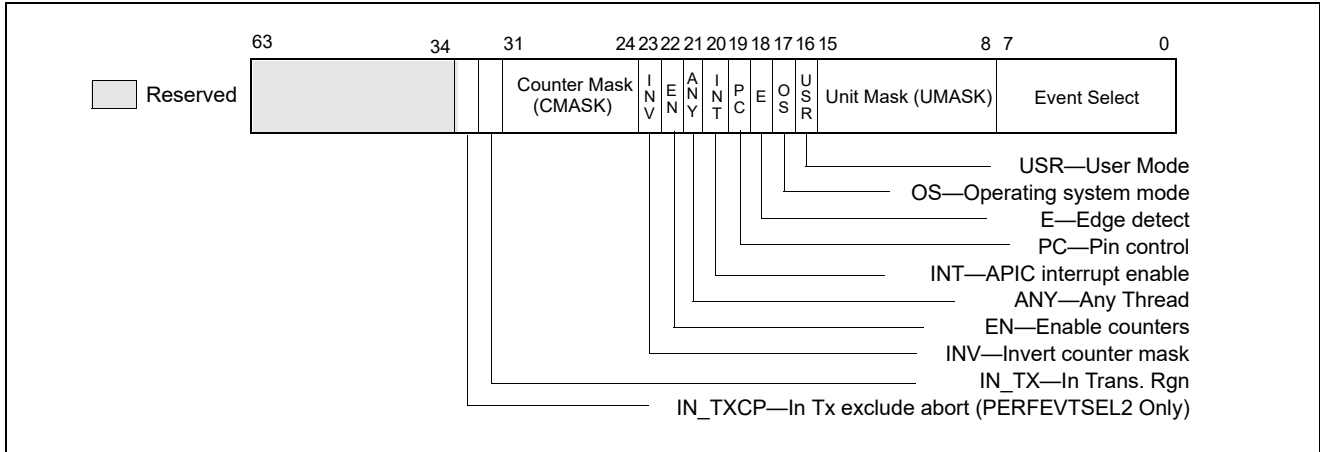


Figure 19-34. Layout of IA32_PERFEVTSELx MSRs Supporting Intel TSX

A common usage of setting IN_TXCP=1 is to capture the number of events that were discarded due to a transactional abort. With IA32_PMC2 configured to count in such a manner, then when a transactional region aborts, the value for that counter is restored to the value it had prior to the aborted transactional region. As a result, any updates performed to the counter during the aborted transactional region are discarded.

On the other hand, setting IN_TX=1 can be used to drill down on the performance characteristics of transactional code regions. When a PMCx is configured with the corresponding IA32_PERFEVTSELx.IN_TX=1, only eventing conditions that occur inside transactional code regions are propagated to the event logic and reflected in the counter result. Eventing conditions specified by IA32_PERFEVTSELx but occurring outside a transactional region are discarded.

Additionally, a number of performance events are solely focused on characterizing the execution of Intel TSX transactional code, they can be found at: <https://perfmon-events.intel.com/>.

19.3.6.5.1 Intel TSX and PEBS Support

If a PEBS event would have occurred inside a transactional region, then the transactional region first aborts, and then the PEBS event is processed.

Two of the TSX performance monitoring events also support using the PEBS facility to capture additional information. They are:

- HLE_RETIREDA.BORTED (encoding C8H mask 04H),
- RTM_RETIREDA.BORTED (encoding C9H mask 04H).

A transactional abort (HLE_RETIREDA.BORTED,RTM_RETIREDA.BORTED) can also be programmed to cause PEBS events. In this scenario, a PEBS event is processed following the abort.

Pending a PEBS record inside of a transactional region will cause a transactional abort. If a PEBS record was pending at the time of the abort or on an overflow of the TSX PEBS events listed above, only the following PEBS entries will be valid (enumerated by PEBS entry offset B8H bits[33:32] to indicate an HLE abort or an RTM abort):

- Offset B0H: EventingIP,
- Offset B8H: TX Abort Information

These fields are set for all PEBS events.

- Offset 08H (RIP/EIP) corresponds to the instruction following the outermost XACQUIRE in HLE or the first instruction of the fallback handler of the outermost XBEGIN instruction in RTM. This is useful to identify the aborted transactional region.

In the case of HLE, an aborted transaction will restart execution deterministically at the start of the HLE region. In the case of RTM, an aborted transaction will transfer execution to the RTM fallback handler.

The layout of the TX Abort Information field is given in Table 19-31.

Table 19-31. TX Abort Information Field Definition

Bit Name	Offset	Description
Cycles_Last_TX	31:0	The number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
HLE_Abort	32	If set, the abort information corresponds to an aborted HLE execution
RTM_Abort	33	If set, the abort information corresponds to an aborted RTM execution
Instruction_Abort	34	If set, the abort was associated with the instruction corresponding to the eventing IP (offset OBOH) within the transactional region.
Non_Instruction_Abort	35	If set, the instruction corresponding to the eventing IP may not necessarily be related to the transactional abort.
Retry	36	If set, retrying the transactional execution may have succeeded.
Data_Conflict	37	If set, another logical processor conflicted with a memory address that was part of the transactional region that aborted.
Capacity Writes	38	If set, the transactional region aborted due to exceeding resources for transactional writes.
Capacity Reads	39	If set, the transactional region aborted due to exceeding resources for transactional reads.
Reserved	63:40	Reserved

19.3.6.6 Uncore Performance Monitoring Facilities in the 4th Generation Intel® Core™ Processors

The uncore sub-system in the 4th Generation Intel® Core™ processors provides its own performance monitoring facility. The uncore PMU facility provides dedicated MSRs to select uncore performance monitoring events in a similar manner as those described in Section 19.3.4.6.

The ARB unit and each C-Box provide local pairs of event select MSR and counter register. The layout of the event select MSRs in the C-Boxes are identical as shown in Figure 19-32.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 19-33 shows the layout of the uncore domain global control.

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 19-19 summarizes the number MSRs for uncore PMU for each box.

Table 19-32. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, see Table 2-21 MSR_UNC_CBO_CONFIG
ARB	1	2	44	Yes	Uncore	
Fixed Counter	N.A.	N.A.	48	No	Uncore	

The uncore performance events for the C-Box and ARB units can be found at: <https://perfmon-events.intel.com/>.

19.3.6.7 Intel® Xeon® Processor E5 v3 Family Uncore Performance Monitoring Facility

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v3 families are available in “Intel® Xeon® Processor E5 v3 Uncore Performance Monitoring Programming Reference Manual”. The MSR-based uncore PMU interfaces are listed in Table 2-33.

19.3.7 5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility

The 5th Generation Intel® Core™ processor and the Intel® Core™ M processor families are based on the Broadwell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 19.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 19.2.3.

The core PMU has the same capability as those described in Section 19.3.6. IA32_PERF_GLOBAL_STATUS provide a bit indicator (bit 55) for PMI handler to distinguish PMI due to output buffer overflow condition due to accumulating packet data from Intel Processor Trace.

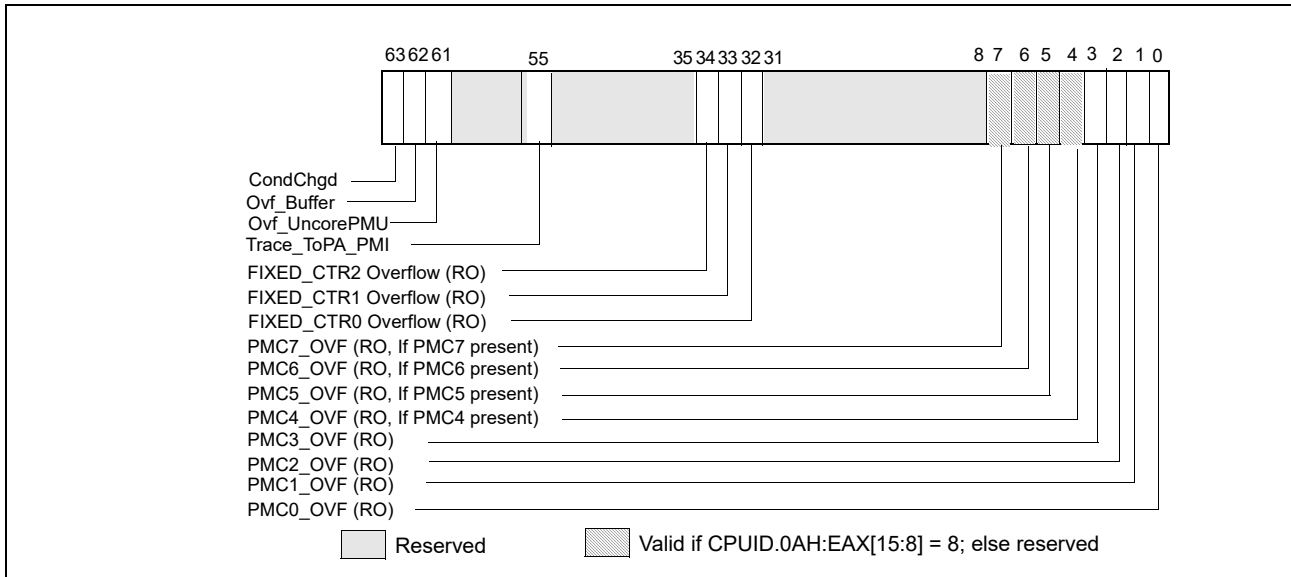


Figure 19-35. IA32_PERF_GLOBAL_STATUS MSR in Broadwell Microarchitecture

Details of Intel Processor Trace is described in Chapter 32, “Intel® Processor Trace”. The IA32_PERF_GLOBAL_OVF_CTRL MSR provides a corresponding reset control bit.

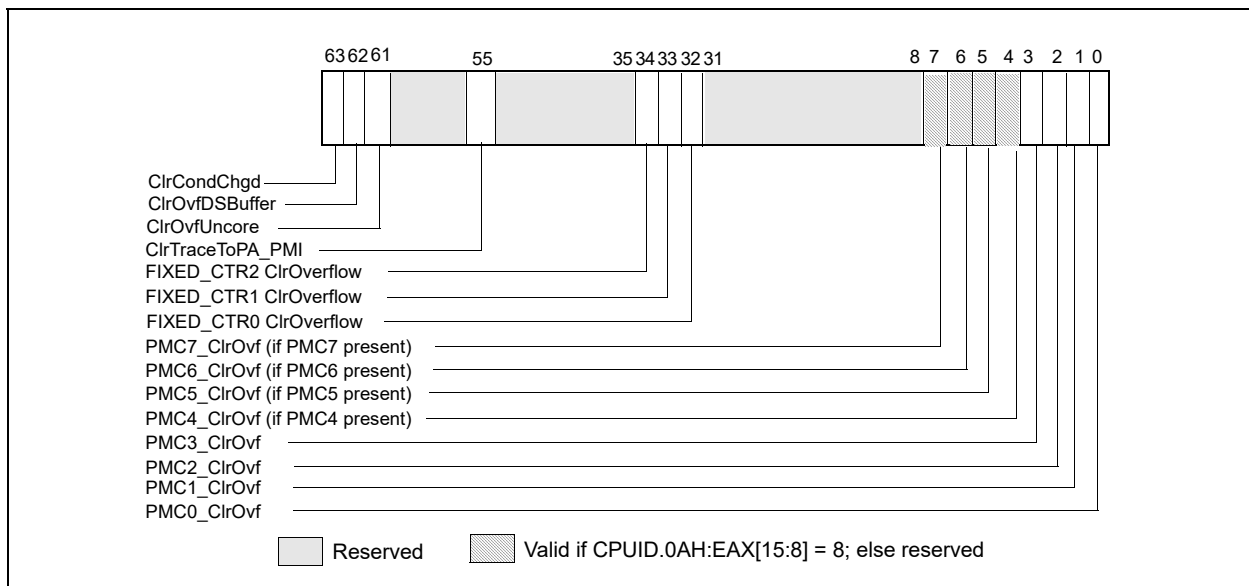


Figure 19-36. IA32_PERF_GLOBAL_OVF_CTRL MSR in Broadwell microarchitecture

The specifics of non-architectural performance events can be found at: <https://perfmon-events.intel.com/>.

19.3.8 6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility

The 6th generation Intel® Core™ processor is based on the Skylake microarchitecture. The 7th generation Intel® Core™ processor is based on the Kaby Lake microarchitecture. The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture. For these microarchitectures, the core PMU supports architectural performance monitoring capability with version ID 4 (see Section 19.2.4) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 4 capabilities are described in Section 19.2.4.

The core PMU's capability is similar to those described in Section 19.6.3 through Section 19.3.4.5, with some differences and enhancements summarized in Table 19-33. Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 19.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® Transactional Synchronization Extensions" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Performance monitoring result may be affected by side-band activity on processors that support Intel SGX, details are described in Chapter 39, "Enclave Code Debug and Profiling".

Table 19-33. Core PMU Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 19.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 19.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 19.2.1.
Architectural Perfmon version	4	3	See Section 19.2.4
PMI Overhead Mitigation	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with streamlined semantics. ▪ Freeze_LBR_on_PMI with streamlined semantics. ▪ Freeze_while_SMM. 	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_LBR_on_PMI with legacy semantics for branch profiling. ▪ Freeze_while_SMM. 	See Section 17.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	<ul style="list-style-type: none"> ▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_STATUS_RESET ▪ Set via IA32_PERF_GLOBAL_STATUS_SET 	<ul style="list-style-type: none"> ▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_OVF_CTRL 	See Section 19.2.4.

Table 19-33. Core PMU Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	<ul style="list-style-type: none"> Individual counter overflow PEBS buffer overflow ToPA buffer overflow CTR_Frz, LBR_Frz, ASCI 	<ul style="list-style-type: none"> Individual counter overflow PEBS buffer overflow ToPA buffer overflow (applicable to Broadwell microarchitecture) 	See Section 19.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	<ul style="list-style-type: none"> CTR_Frz LBR_Frz 	NA	See Section 19.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	NA	See Section 19.2.4.3.
Precise Events	See Table 19-36.	See Table 19-12.	IA32_PMC4-PMC7 do not support PEBS.
PEBS for front end events	See Section 19.3.8.2.	No	
LBR Record Format Encoding	000101b	000100b	Section 17.4.8.1
LBR Size	32 entries	16 entries	
LBR Entry	From_IP/To_IP/LBR_Info triplet	From_IP/To_IP pair	Section 17.12
LBR Timing	Yes	No	Section 17.12.1
Call Stack Profiling	Yes, see Section 17.11	Yes, see Section 17.11	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types.	MSR 1A6H and 1A7H; Extended request and response types.	
Intel TSX support for Perfmon	See Section 19.3.6.5.	See Section 19.3.6.5.	

19.3.8.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 6th generation, 7th generation and 8th generation Intel Core processors provides a number enhancement relative to PEBS in processors based on Haswell/Broadwell microarchitectures. The key components and differences of PEBS facility relative to Haswell/Broadwell microarchitecture is summarized in Table 19-34.

Table 19-34. PEBS Facility Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 19.3.1.1.1	Section 19.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 19-15	Figure 19-15	
PEBS-EventingIP	Yes	Yes	
PEBS record format encoding	0011b	0010b	
PEBS record layout	Table 19-35; enhanced fields at offsets 98H- B8H; and TSC record field at C0H.	Table 19-24; enhanced fields at offsets 98H, A0H, A8H, B0H.	
Multi-counter PEBS resolution	PEBS record 90H resolves the eventing counter overflow.	PEBS record 90H reflects IA32_PERF_GLOBAL_STATUS.	
Precise Events	See Table 19-36.	See Table 19-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.

Table 19-34. PEBS Facility Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS-Load Latency	See Section 19.3.4.4.2.	See Section 19.3.4.4.2.	
Data Address Profiling	Yes	Yes	
FrontEnd event support	FrontEnd_Retried event and MSR_PEBS_FRONTEND.	No	IA32_PMC0-PMC3 only.

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTES

Precise events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

19.3.8.1.1 PEBS Data Format

The PEBS record format for the 6th generation, 7th generation and 8th generation Intel Core processors is reporting with encoding 0011b in IA32_PERF_CAPABILITIES[11:8]. The lay out is shown in Table 19-35. The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 19-35. PEBS Record Format for 6th Generation, 7th Generation and 8th Generation Intel Core Processor Families

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counter
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Data Source Encoding
40H	R/EBP	A8H	Latency value (core cycles)
48H	R/ESP	B0H	EventingIP
50H	R8	B8H	TX Abort Information (Section 19.3.6.5.1)
58H	R9	COH	TSC
60H	R10		

The layout of PEBS records are largely identical to those shown in Table 19-24.

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 19.3.4.4.2), PDIR (Section 19.3.4.4.4), and data address profiling (Section 19.3.6.3).

In the core PMU of the 6th generation, 7th generation and 8th generation Intel Core processors, load latency facility and PDIR capabilities and data address profiling are unchanged relative to the 4th generation and 5th generation Intel Core processors. Similarly, precise store is replaced by data address profiling.

With format 0010b, a snapshot of the IA32_PERF_GLOBAL_STATUS may be useful to resolve the situations when more than one of IA32_PMICx have been configured to collect PEBS data and two consecutive overflows of the PEBS-enabled counters are sufficiently far apart in time. It is also possible for the image at 90H to indicate multiple PEBS-enabled counters have overflowed. In the latter scenario, software cannot to correlate the PEBS record entry to the multiple overflowed bits.

With PEBS record format encoding 0011b, offset 90H reports the “applicable counter” field, which is a multi-counter PEBS resolution index allowing software to correlate the PEBS record entry with the eventing PEBS overflow when multiple counters are configured to record PEBS records. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

19.3.8.1.2 PEBS Events

The list of precise events supported for PEBS in the Skylake, Kaby Lake and Coffee Lake microarchitectures is shown in Table 19-36.

Table 19-36. Precise Events for the Skylake, Kaby Lake and Coffee Lake Microarchitectures

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST ¹	01H
		ALL_CYCLES ²	01H
OTHER_ASSISTS	C1H	ANY	3FH
BR_INST_RETIRED	C4H	CONDITIONAL	01H
		NEAR_CALL	02H
		ALL_BRANCHES	04H
		NEAR_RETURN	08H
		NEAR_TAKEN	20H
		FAR_BRACHES	40H
BR_MISP_RETIRED	C5H	CONDITIONAL	01H
		ALL_BRANCHES	04H
		NEAR_TAKEN	20H
FRONTEND_RETIRED	C6H	<Programmable ³ >	01H
HLE_RETIRED	C8H	ABORTED	04H
RTM_RETIRED	C9H	ABORTED	04H
MEM_INST_RETIRED ²	D0H	LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H

Table 19-36. Precise Events for the Skylake, Kaby Lake and Coffee Lake Microarchitectures (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_LOAD_RETIRED ⁴	D1H	L1_HIT	01H
		L2_HIT	02H
		L3_HIT	04H
		L1_MISS	08H
		L2_MISS	10H
		L3_MISS	20H
		HIT_LFB	40H
MEM_LOAD_L3_HIT_RETIRED ²	D2H	XSNP_MISS	01H
		XSNP_HIT	02H
		XSNP_HITM	04H
		XSNP_NONE	08H

NOTES:

1. Only available on IA32_PMC1.
2. INST_RETIRED.ALL_CYCLES is configured with additional parameters of cmask = 10 and INV = 1
3. Subevents are specified using MSR_PEBS_FRONTEND, see Section 19.3.8.3
4. Instruction with at least one load uop experiencing the condition specified in the UMask.

19.3.8.1.3 Data Address Profiling

The PEBS Data address profiling on the 6th generation, 7th generation and 8th generation Intel Core processors is largely unchanged from the prior generation. When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H and A8H, as shown in Table 19-26.

Table 19-37. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	<ul style="list-style-type: none"> ▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_INST_RETIRED.STLB_MISS_STORES, MEM_INST_RETIRED.ALL_STORES, MEM_INST_RETIRED.SPLIT_STORES. ▪ Other bits are zero.
Reserved	A8H	Always zero.

19.3.8.2 Frontend Retired Facility

The Skylake Core PMU has been extended to cover common microarchitectural conditions related to the front end pipeline in addition to providing a generic latency mechanism that can locate fetch bubbles without necessarily attributing them to a particular condition. The facility counts the events if the associated instruction reaches retirement (architecturally committed). Additionally, the user may opt to enable the PEBS facility to obtain precise information on the context of the event, e.g., EventingIP.

The supported frontend microarchitectural conditions require the following interfaces:

- The IA32_PERFEVTSELx MSR must select the FRONTEND_RETIRED event, EventSelect = C6H and UMASK = 01H.

- This event employs a new MSR, MSR_PEBS_FRONTEND, to specify the supported frontend event details, see Table 19-38.
 - If precise information is desired, program the PEBS_EN_PMCx field of IA32_PEBS_ENABLE MSR as required.
- Note the AnyThread field of IA32_PERFEVTSELx is ignored by the processor for the “FRONTEND_RETIRED” event. The sub-event encodings supported by MSR_PEBS_FRONTEND.EVTSEL is given in Table 19-38.

Table 19-38. FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL

Sub-Event Name	EVTSEL	Description
ANY_DSB_MISS	1H	Retired Instructions which experienced any decode stream buffer (DSB) miss.
DSB_MISS	11H	Retired Instructions which experienced a DSB miss that caused a fetch starvation cycle.
L11_MISS	12H	The fetch of retired Instructions which experienced Instruction L1 Cache true miss ¹ . Additional requests to the same cache line as an in-flight L11 cache miss will not be counted.
L2_MISS	13H	The fetch of retired Instructions which experienced L2 Cache true miss. Additional requests to the same cache line as an in-flight MLC cache miss will not be counted.
ITLB_MISS	14H	The fetch of retired Instructions which experienced ITLB true miss. Additional requests to the same cache line as an in-flight ITLB miss will not be counted.
STLB_MISS	15H	The fetch of retired Instructions which experienced STLB true miss. Additional requests to the same cache line as an in-flight STLB miss will not be counted.
IDQ_READ_BUBBLES	6H	An IDQ read bubble is defined as any one of the 4 allocation slots of IDQ that is not filled by the front-end on any cycle where there is no back end stall. Using the threshold and latency fields in MSR_PEBS_FRONTEND allows counting of IDQ read bubbles of various magnitude and duration. Latency controls the number of cycles and Threshold controls the number of allocation slots that contain bubbles. The event counts if and only if a sequence of at least FE_LATENCY consecutive cycles contain at least FE_TRESHOLD number of bubbles each.

NOTES:

1. A true miss is the first miss for a cacheline/page (excluding secondary misses that fall into same cacheline/page).

The layout of MSR_PEBS_FRONTEND is given in Table 19-39.

Table 19-39. MSR_PEBS_FRONTEND Layout

Bit Name	Offset	Description
EVTSEL	7:0	Encodes the sub-event within FrontEnd_Retired that can use PEBS facility, see Table 19-38.
IDQ_Bubble_Length	19:8	Specifies the threshold of continuously elapsed cycles for the specified width of bubbles when counting IDQ_READ_BUBBLES event.
IDQ_Bubble_Width	22:20	Specifies the threshold of simultaneous bubbles when counting IDQ_READ_BUBBLES event.
Reserved	63:23	Reserved

The FRONTEND_RETIRED event is designed to help software developers identify exact instructions that caused front-end issues. There are some instances in which the event will, by design, the under-counting scenarios include the following:

- The event counts only retired (non-speculative) front-end events, i.e., events from just true program execution path are counted.
- The event will count once per cacheline (at most). If a cacheline contains multiple instructions which caused front-end misses, the count will be only 1 for that line.
- If the multibyte sequence of an instruction spans across two cachelines and causes a miss it will be recorded once. If there were additional misses in the second cacheline, they will not be counted separately.

- If a multi-uop instruction exceeds the allocation width of one cycle, the bubbles associated with these uops will be counted once per that instruction.
- If 2 instructions are fused (macro-fusion), and either of them or both cause front-end misses, it will be counted once for the fused instruction.
- If a front-end (miss) event occurs outside instruction boundary (e.g., due to processor handling of architectural event), it may be reported for the next instruction to retire.

19.3.8.3 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 19.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 19-40.
- Supplier information (bits 29:16): see Table 19-41.
- Snoop response information (bits 37:30): see Table 19-42.

Table 19-40. MSR_OFFCORE_RSP_x Request_Type Definition (Skylake, Kaby Lake and Coffee Lake Microarchitectures)

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DMND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	14:3	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

Table 19-41 lists the supplier information field that applies to 6th generation, 7th generation and 8th generation Intel Core processors. (6th generation Intel Core processor CPUID signatures: 06_4EH, 06_5EH; 7th generation and 8th generation Intel Core processor CPUID signatures: 06_8EH, 06_9EH).

Table 19-41. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures 06_4EH, 06_5EH and 06_8EH, 06_9EH)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	L4_HIT	22	L4 Cache (if L4 is present in the processor).
	Reserved	25:23	Reserved
	DRAM	26	Local Node
	Reserved	29:27	Reserved
	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).

Table 19-42 lists the snoop information field that apply to processors with CPUID signatures 06_4EH, 06_5EH, 06_8EH, 06_9E, and 06_55H.

Table 19-42. MSR_OFFCORE_RSP_x Snoop Info Field Definition (CPUID Signatures 06_4EH, 06_5EH, 06_8EH, 06_9E and 06_55H)

Subtype	Bit Name	Offset	Description
Snoop Info	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).
	SNOOP_NONE	31	No details on snoop-related information.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores. -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNOOP_HIT_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO). -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD). -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S). In the LLC Miss case, data is returned from DRAM.
	SNOOP_HIT_WITH_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	SNOOP_HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD). -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO). -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	SNOOP_NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

19.3.8.3.1 Off-core Response Performance Monitoring for the Intel® Xeon® Processor Scalable Family

The following tables list the requestor and supplier information fields that apply to the Intel® Xeon® Processor Scalable Family.

- Transaction request type encoding (bits 15:0): see Table 19-43.
- Supplier information (bits 29:16): see Table 19-44.
- Supplier information (bits 29:16) with support for Intel® Optane™ DC Persistent Memory support: see Table 19-45.
- Snoop response information has not been changed and is the same as in (bits 37:30): see Table 19-42.

Table 19-43. MSR_OFFCORE_RSP_x Request_Type Definition (Intel® Xeon® Processor Scalable Family)

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DEMAND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DEMAND_CODE_RD	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	3	Reserved.
PF_L2_DATA_RD	4	Counts the number of prefetch data reads into L2.
PF_L2_RFO	5	Counts the number of RFO Requests generated by the MLC prefetches to L2.
Reserved	6	Reserved.
PF_L3_DATA_RD	7	Counts the number of MLC data read prefetches into L3.
PF_L3_RFO	8	Counts the number of RFO requests generated by MLC prefetches to L3.
Reserved	9	Reserved.
PF_L1D_AND_SW	10	Counts data cacheline reads generated by hardware L1 data cache prefetcher or software prefetch requests.
Reserved	14:11	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

Table 19-44 lists the supplier information field that applies to the Intel Xeon Processor Scalable Family (CPUID signature: 06_55H).

Table 19-44. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_55H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	Reserved	25:22	Reserved
	L3_MISS_LOCAL_DRAM	26	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
Reserved	30	Reserved	

Table 19-45 lists the supplier information field that applies to the Intel Xeon Processor Scalable Family (CPUID signature: 06_55H, Steppings 0x5H - 0xFH).

Table 19-45. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_55H, Steppings 0x5H - 0xFH)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	LOCAL_PMM	22	Local home requests that were serviced by local PMM.
	REMOTE_HOP0_PMM	23	Hop 0 Remote supplier.
	REMOTE_HOP1_PMM	24	Hop 1 Remote supplier.
	REMOTE_HOP2P_PMM	25	Hop 2 or more Remote supplier.
	L3_MISS_LOCAL_DRAM	26	L3 Miss: Local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
Reserved		30	Reserved

19.3.8.4 Uncore Performance Monitoring Facilities on Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Cannon Lake microarchitecture introduces LLC support of up to six processor cores. To support six processor cores and eight LLC slices, existing MSR addresses have been rearranged and new CBo MSR addresses have been added. Uncore performance monitoring software drivers from prior generations of Intel Core processors will need to update the MSR addresses. The new MSR addresses and updated MSR addresses have been added to the Uncore PMU listing in Section 2.17.2, “MSRs Specific to 8th Generation Intel® Core™ i3 Processors” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*.

19.3.9 10th Generation Intel® Core™ Processor Performance Monitoring Facility

Some 10th generation Intel® Core™ processors and some 3rd generation Intel® Xeon® Processor Scalable Family are based on Ice Lake microarchitecture. Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture. For these processors, the core PMU supports architectural performance monitoring capability with version Id 5 (see Section 19.2.5) and a host of non-architectural monitoring capabilities.

The core PMU's capability is similar to those described in Section 19.3.1 through Section 19.3.8, with some differences and enhancements summarized in Table 19-46.

Table 19-46. Core PMU Summary of the Ice Lake Microarchitecture

Box	Ice Lake Microarchitecture	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Comment
Architectural Perfmon version	5	4	See Section 19.2.5.
PEBS: Basic functionality	Yes	Yes	See Section 19.3.9.1.

Table 19-46. Core PMU Summary of the Ice Lake Microarchitecture

Box	Ice Lake Microarchitecture	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Comment
PEBS record format encoding	0100b	0011b	See Section 19.6.2.4.2.
Extended PEBS	PEBS is extended to all Fixed and General Purpose counters and to all performance monitoring events.	No	See Section 19.9.1.
Adaptive PEBS	Yes	No	See Section 19.9.2.
Performance Metrics	Yes (4)	No	See Section 19.3.9.3.
PEBS-PDIR	IA32_FIXED0 only (Corresponding counter control MSRs must be enabled.)	IA32_PMC1 only.	

19.3.9.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 10th generation Intel Core processors provides a number of enhancements relative to PEBS in processors based on the Skylake, Kaby Lake, and Coffee Lake microarchitectures. Enhancement of the PEBS facility with Extended PEBS and Adaptive PEBS features is described in detail in Section 19.9.

The 3rd generation Intel Xeon Scalable Family of processors based on the Ice Lake microarchitecture introduce EPT-friendly PEBS. This allows EPT violations and other VM Exits to be taken on PEBS accesses to the DS Area. See Section 19.9.5 for details.

19.3.9.2 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 19.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 18-[N1].
- Response type encoding (bits 16-37) of
 - Supplier information: see Table [18-N2].
 - Snoop response information: see Table [18-N3].
- All transactions are tracked at cacheline granularity except some in request type OTHER.

**Table 19-47. MSR_OFFCORE_RSP_x Request_Type Definition
(Processors Based on Ice Lake Microarchitecture)**

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data and page table entry reads.
DEMAND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DEMAND_CODE_RD	2	Counts demand instruction fetches and instruction prefetches targeting the L1 instruction cache.
Reserved	3	Reserved
HWPF_L2_DATA_RD	4	Counts hardware generated data read prefetches targeting the L2 cache.
HWPF_L2_RFO	5	Counts hardware generated prefetches for exclusive ownership (RFO) targeting the L2 cache.
Reserved	6	Reserved
HWPF_L3	9:7 and 13 ¹	Counts hardware generated prefetches of any type targeting the L3 cache.

**Table 19-47. MSR_OFFCORE_RSP_x Request_Type Definition
(Processors Based on Ice Lake Microarchitecture)**

Bit Name	Offset	Description
HWPF_L1D_AND_SWPF	10	Counts hardware generated data read prefetches targeting the L1 data cache and the following software prefetches (PREFETCHNTA, PREFETCHT0/1/2).
STREAMING_WR	11	Counts streaming stores.
Reserved	12	Reserved
Reserved	14	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

NOTES:

1. All bits need to be set to 1 to count this type.

Ice Lake microarchitecture has added a new category of Response subtype, called a Combined Response Info. To count a feature in this type, all the bits specified must be set to 1.

A valid response type must be a non-zero value of the following expression:

Any | ['OR' of Combined Response Info Bits | (('OR' of Supplier Info Bits) & ('OR' of Snoop Info Bits)]]

If "ANY" bit[16] is set, other response type bits [17-39] are ignored.

Table 19-48 lists the supplier information field that applies to processors based on Ice Lake microarchitecture.

**Table 19-48. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Combined Response Info	DRAM	26, 31, 32 ¹	Requests that are satisfied by DRAM.
	NON_DRAM	26, 37 ¹	Requests that are satisfied by a NON_DRAM system component. This includes MMIO transactions.
	L3_MISS	22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 ¹	Requests that were not supplied by the L3 Cache. The event includes some currently reserved bits in anticipation of future memory designs.
Supplier Info	L3_HIT	18,19, 20 ¹	Requests that hit in L3 cache. Depending on the snoop response the L3 cache may have retrieved the cacheline from another core's cache.
Reserved		17, 21:25, 27:29	Reserved.

NOTES:

1. All bits need to be set to 1 to count this type.

Table 19-49 lists the snoop information field that applies to processors based on Ice Lake microarchitecture.

**Table 19-49. MSR_OFFCORE_RSP_x Snoop Info Field Definition
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Snoop Info	Reserved	30	Reserved.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was sent and none of the snooped caches contained the cacheline.
	SNOOP_HIT_NO_FWD	34	A snoop was sent and hit in at least one snooped cache. The unmodified cacheline was not forwarded back, because the L3 already has a valid copy.
	Reserved	35	Reserved.
	SNOOP_HITM	36	A snoop was sent and the cacheline was found modified in another core's caches. The modified cacheline was forwarded to the requesting core.

19.3.9.3 Performance Metrics

The Ice Lake core PMU provides built-in support for Top-down Microarchitecture Analysis (TMA) method level 1 metrics. These metrics are always available to cross-validate performance observations, freeing general purpose counters to count other events in high counter utilization scenarios. For more details about the method, refer to Top-Down Analysis Method chapter (Appendix B.1) of the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

A new MSR called MSR_PERF_METRICS reports the metrics directly. Software can check (and/or expose to its guests) the availability of the PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15). For additional details on this MSR, refer to Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*.

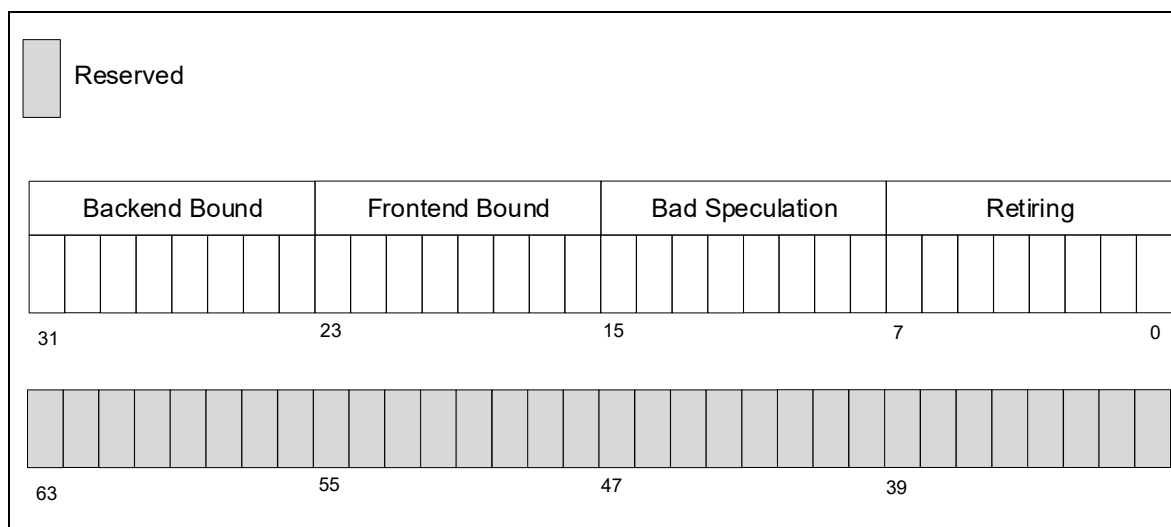


Figure 19-37. MSR_PERF_METRICS Definition

This register exposes the four TMA Level 1 metrics. The lower 32 bits are divided into four 8-bit fields, as shown by the above figure, each of which is an integer fraction of 255.

To support built-in performance metrics, new bits have been added to the following MSRs:

- IA32_PERF_GLOBAL_CTRL. EN_PERF_METRICS[48]: If this bit is set and fixed-function performance-monitoring counter 3 is enabled, built-in performance metrics are enabled.
- IA32_PERF_GLOBAL_STATUS_SET. SET_OVF_PERF_METRICS[48]: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS_RESET. RESET_OVF_PERF_METRICS[48]: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS. OVF_PERF_METRICS[48]: If this bit is set, it indicates that a PERF_METRICS-related resource has overflowed and a PMI is triggered¹. If this bit is clear, no such overflow has occurred.

NOTE

Software has to synchronize, e.g., re-start, fixed-function performance-monitoring counter 3 as well as PERF_METRICS when either bit 35 or 48 in IA32_PERF_GLOBAL_STATUS is set. Otherwise, PERF_METRICS may return undefined values.

The values in MSR_PERF_METRICS are derived from fixed-function performance-monitoring counter 3. Software should start both registers, PERF_METRICS and fixed-function performance-monitoring counter 3, from zero. Additionally, software is recommended to periodically clear both registers in order to maintain accurate measurements for certain scenarios that involve sampling metrics at high rates.

In order to save/restore PERF_METRICS, software should follow these guidelines:

- PERF_METRICS and fixed-function performance-monitoring counter 3 should be saved and restored together.
- To ensure that PERF_METRICS and fixed-function performance-monitoring counter 3 remain synchronized, both should be disabled during both save and restore. Software should enable/disable them atomically, with a single write to IA32_PERF_GLOBAL_CTRL to set/clear both EN_PERF_METRICS[bit 48] and EN_FIXED_CTR3[bit 35].
- On state restore, fixed-function performance-monitoring counter 3 must be restored **before** PERF_METRICS, otherwise undefined results may be observed.

19.3.10 12th Generation Intel® Core™ Processor Performance Monitoring Facility

The 12th generation Intel® Core™ processor supports Alder Lake performance hybrid architecture. These processors offer a unique combination of Performance and Efficient-cores (P-core and E-core). The P-core is based on Golden Cove microarchitecture and the E-core is based on Gracemont microarchitecture. They report architectural performance monitoring version ID = 5 and support non-architectural monitoring capabilities described in this section.

19.3.10.1 P-core Performance Monitoring Unit

The P-core PMU's capability is similar to those described in Section 19.3.1 through Section 19.3.9, with some differences and enhancements summarized in Table 19-50.

1. An overflow of fixed-function performance-monitoring counter 3 should normally happen first if software follows Intel's recommendations.

Table 19-50. Core PMU Summary of the Golden Cove Microarchitecture

Box	Golden Cove Microarchitecture	Ice Lake Microarchitecture	Comment
Architectural Perfmon version	5	5	See Section 19.2.5.
Event-Counter Restrictions	Simplified identification		Counters 4-7 support a subset of events. See Section 19.3.10.1.2.
Performance Metrics	Yes (12)	Yes (4)	See Section 19.3.9.3.
PEBS: Baseline, record format	Yes 0100b	Yes 0100b	See Section 19.3.9.
PEBS: EPT-friendly	Yes	No; debuts in Ice Lake server microarchitecture	See Section 19.6.2.4.2.
PEBS: Precise Distribution	IA32_FIXED0 instruction-granularity PDist on IA32_PMC0	IA32_FIXED0 cycle-granularity No PDist	See Section 19.9.6.
PEBS: Load Latency	Instruction latency Cache latency Access info fields (5)	Instruction latency Access info fields (3)	See Section 19.9.7.
PEBS: Store Latency	Cache latency Access info fields (3)	None	See Section 19.9.8.

19.3.10.1.1 P-core Perf Metrics Extensions

For 12th generation Intel Core processor P-cores, the core PMU supports the built-in metrics that were introduced in the Ice Lake microarchitecture PMU. This core PMU extends the PERF_METRICS MSR to feature TMA method level 2 metrics, as shown in Figure 19-38.

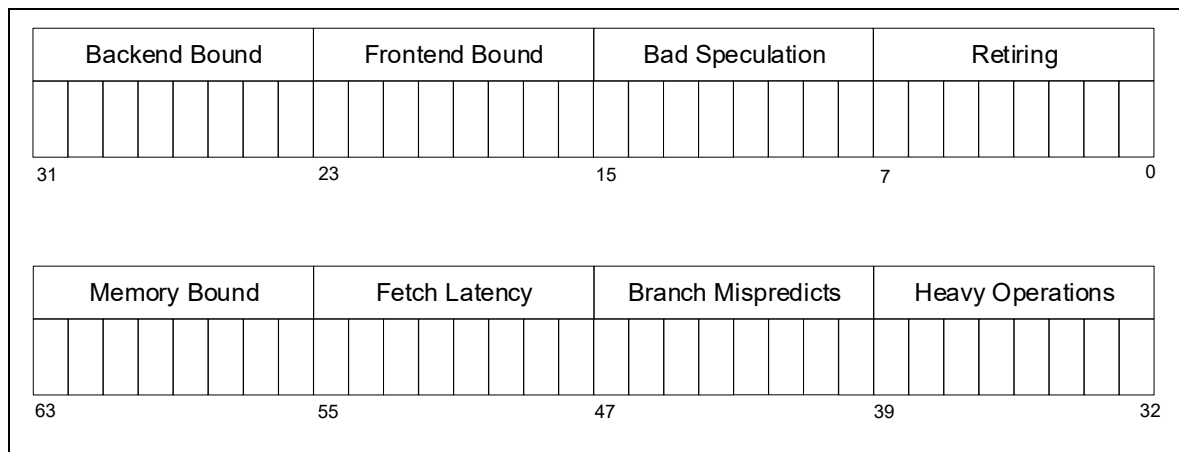


Figure 19-38. PERF_METRICS MSR Definition for 12th Generation Intel® Core™ Processor P-core

The lower half of the register is the TMA level 1 metrics (legacy). The upper half is also divided into four 8-bit fields, each of which is an integer fraction of 255. Additionally, each of the new level 2 metrics in the upper half is a subset of the corresponding level 1 metric in the lower half (that is, its parent node per the TMA hierarchy). This enables software to deduce the other four level 2 metrics by subtracting corresponding metrics as shown in Figure 19-39.

$\begin{aligned} \text{Light_Operations} &= \text{Retiring} - \text{Heavy_Operations} \\ \text{Machine_Clears} &= \text{Bad_Speculation} - \text{Branch_Mispredicts} \\ \text{Fetch_Bandwidth} &= \text{Frontend_Bound} - \text{Fetch_Latency} \\ \text{Core_Bound} &= \text{Backend_Bound} - \text{Memory_Bound} \end{aligned}$

Figure 19-39. Deducing Implied Level 2 Metrics in the Core PMU for 12th Generation Intel® Core™ Processor P-core

The PERF_METRICS MSR and fixed-function performance-monitoring counter 3 of the core PMU feature 12 metrics in total that cover all level 1 and level 2 nodes of the TMA hierarchy.

19.3.10.1.2 P-core Counter Restrictions Simplification

The 12th generation Intel Core processor P-core allows identification of performance monitoring events with counter restrictions based on event encodings. The general rule is: Event Codes < 0x90 are restricted to general-purpose performance-monitoring counters 0-3. Event Codes ≥ 0x90 are likely to have no restrictions. Table 19-51 lists the exceptions to this rule.

Table 19-51. Special Performance Monitoring Events with Counter Restrictions

Event Encoding ¹	Event Name	Counter Restriction
xx3C	CPU_CLK_UNHALTED.*	0-7 (No restriction for all architectural events.)
xx2E	LONGEST_LAT_CACHE.*	
xxDx	MEM*_RETIRED.*	0-3
01A3, 02A3, 08A3	Some CYCLE_ACTIVITY sub-events	0-3
02CD	MEM_TRANS_RETIRED.STORE_SAMPLE	0
04A4	TOPDOWN.BAD_SPEC_SLOTS	0
08A4	TOPDOWN.BR_MISPREDICT_SLOTS	
xxCE	AMX_OPS_RETIRED	0

NOTES:

1. Linux perf rUUEE syntax, where UU is the Unit Mask field and EE is the Event Select (also known as Event Code) field in the IA32_PERFEVTSELx MSRs.

19.3.10.1.3 P-core Off-core Response Facility

For the 12th generation Intel Core processor P-core, the Off-core Response (OCR) Facility is similar to that described in Section 19.3.9.2.

The following enhancements are introduced for the Request_Type of MSR_OFFCORE_RSP_x:

- WB (bits 3 and 12): Count writeback (modified or non-modified) transactions by core caches.
- HWPF_L1D (bit 10): Counts hardware generated data read prefetches targeting the L1 data cache (only).
- SWPF_READ (bit 14): Counts software generated data read prefetches by the PREFETCHNTA and PREFETCHT0/1/2 instructions.

19.3.10.2 E-core Performance Monitoring Unit

The core PMU capabilities on the 12th generation Intel Core processor E-core are summarized in Table 19-52 below.

Table 19-52. Core PMU Summary of the Gracemont Microarchitecture

Box	Gracemont Microarchitecture	Tremont Microarchitecture	Comment
Number of fixed-function performance-monitoring counters per core	3	3	Use CPUID to enumerate number of counters. See Section 19.2.1.
Number of general-purpose counters per core	6	4	Use CPUID to enumerate number of counters. See Section 19.2.1.
Architectural Performance Monitoring version ID	5	5	See Section 19.2.5.
PEBS record format encoding	0100b	0100b	See Section 19.5.5.
EPT-friendly PEBS support	Yes	No	See Section 19.9.5.
Extended PEBS	Yes	Yes	See Section 19.9.1.
Adaptive PEBS	Yes	Yes	See Section 19.9.2.
Precise distribution (PDist) PEBS	IA32_PMC0 and IA32_FIXED_CTRO	IA32_PMC0 and IA32_FIXED_CTRO	PDist eliminates skid, see Section 19.9.3, Section 19.9.4, and Section 19.9.6.
PEBS Latency	Load and Store Latency	No	See Section 19.3.10.2.1, Section 19.3.10.2.2, Section 19.9.7, and Section 19.9.8.
PEBS Output	DS Save Area or Intel® Processor Trace	DS Save Area or Intel® Processor Trace	See Section 19.5.5.2.1.
Offcore Response	MSR 01A6H and 01A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	See Section 19.5.5.4.

19.3.10.2.1 E-core PEBS Load Latency

The 12th generation Intel Core processor E-core includes PEBS Load Latency support similar to that described in Section 19.9.7.

When a programmable counter is configured to count MEM_UOPS_RETIRED.LOAD_LATENCY_ABOVE_THRESHOLD (IA32_PERFEVTSELx[15:0] = 0xD005, with CMASK=0 and INV=0), selected load operations whose latency exceeds the threshold provided in MSR_PEBS_LD_LAT_THRESHOLD (MSR 03F6H) will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 19.9.2.2.2). The formats of these fields are shown in Table 19-53 and Table 19-94.

Table 19-53. E-core PEBS Memory Access Info Encoding

Bit(s)	Field	Description
3:0	Data Source	The source of the data; see Table 19-54.
4	Lock	0: The operation was not part of a locked transaction. 1: The operation was part of a locked transaction.

Table 19-53. E-core PEBS Memory Access Info Encoding (Contd.)

Bit(s)	Field	Description
5	STLB_MISS	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
6	ST_FWD_BLK	0: Load did not get a store forward block. 1: Load got a store forward block.
63:7	Reserved	Reserved

For details on E-core PEBS memory access latency encoding, see the Access Latency Field in Table 19-94.

Table 19-54. E-core PEBS Data Source Encodings

Encoding	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request) and MMIO. Memory mapped I/O hit.
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. Outstanding core cache miss to same cache-line address was already underway. (Pending core cache hit.)
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HITE. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found (clean).
06H	L3 HITM. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a modified copy was found.
07H	Reserved.
08H	L3 HITF. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a shared or forwarding copy was found.
09H	Reserved.
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	Reserved.
0CH	Reserved.
0DH	Reserved.
0EH	I/O. Request of input/output operation.
0FH	The request was to un-cacheable memory.

19.3.10.2.2 E-core PEBS Store Latency

The 12th generation Intel Core processor E-core includes PEBS Store Latency support. When a programmable counter is configured to count MEM_UOPS_RETIRED.STORE_LATENCY (IA32_PERFEVTSELx[15:0] = 0xD006, with CMASK=0 and INV=0), all store operations will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 18.9.2.2.2). The formats of these fields are shown in Table 19-53 and Table 19-94.

19.3.10.2.3 E-core Precise Distribution (PDist) Support

The 12th generation Intel Core processor E-core supports PEBS with Precise Distribution (PDist) on IA32_PMC0 and IA32_FIXED_CTR0. All precise events support PDist save for UOPS_RETIRED. See Section 19.9.6 for additional details on PDist.

19.3.10.2.4 E-core Enhanced Off-core Response

Event number 0B7H support off-core response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. There are unique pairs of MSR_OFFCORE_RSPx registers per core. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 and bits 49:44 specify the request type of a transaction request to the uncore. This is described in Table 19-55.
- Bits 30:16 specify Response Type information or an L2 Hit, and is described in Table 19-75.
- If L2 misses, then bits 37:31 can be used to specify snoop response information and is described in Table 19-76.
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 19.5.2.3 for details.

Table 19-55. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPF_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPF_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPF_L2_CODE_RD	6	Counts all prefetch (that bring data to MLC only) code reads.
HWPF_L3_DATA_RD	7	Counts L3 cache hardware prefetch data reads (written to the L3 cache only).
HWPF_L3_RFO	8	Counts L3 cache hardware prefetch RFOs (written to the L3 cache only).
HWPF_L3_CODE_RD	9	Counts L3 cache hardware prefetch code reads (written to the L3 cache only).
HWPF_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
RSVD	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WIL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (WCIL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (WCILF).
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

19.3.10.3 Unhalted Reference Cycles

The Unhalted Reference Cycles architectural performance monitoring event is enhanced to count at TSC-rate in the 12th generation Intel Core processor P-core when used on a general-purpose PMC. This enhancement makes it consistent with the fixed-function counter 2 and the E-core. As a result, this event is kept enumerated in CPUID leaf 0AH.EBX (unlike prior hybrid parts).

19.4 PERFORMANCE MONITORING (INTEL® XEON™ PHI PROCESSORS)

NOTE

This section also applies to the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture.

19.4.1 Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring

The Intel® Xeon Phi™ processor 7200/5200/3200 series are based on the Knights Landing microarchitecture. The performance monitoring capabilities are distributed between its tiles (pair of processor cores) and untile (connecting many tiles in a physical processor package). Functional details of the tiles and untile of the Knights Landing microarchitecture can be found in Chapter 16 of *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

A complete description of the tile and untile PMU programming interfaces for Intel Xeon Phi processors based on the Knights Landing microarchitecture can be found in the Technical Document section at <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>.

A tile contains a pair of cores attached to a shared L2 cache and is similar to those found in Intel Atom® processors based on the Silvermont microarchitecture. The processor provides several new capabilities on top of the Silvermont performance monitoring facilities.

The processor supports architectural performance monitoring capability with version ID 3 (see Section 19.2.3) and a host of non-architectural performance monitoring capabilities. The processor provides two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

Non-architectural performance monitoring in the processor also uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 19-6 and described in Section 19.2.1.1 and Section 19.2.3. The processor supports AnyThread counting in three architectural performance monitoring events.

19.4.1.1 Enhancements of Performance Monitoring in the Intel® Xeon Phi™ processor Tile

The Intel® Xeon Phi™ processor tile includes the following enhancements to the Silvermont microarchitecture.

- AnyThread support. This facility is limited to following three architectural events: Instructions Retired, Unhalted Core Cycles, Unhalted Reference Cycles using IA32_FIXED_CTR0-2 and Unhalted Core Cycles, Unhalted Reference Cycles using IA32_PERFEVTSELx.
- PEBS-DLA (Processor Event-Based Sampling-Data Linear Address) fields. The processor provides memory address in addition to the Silvermont PEBS record support on select events. The PEBS recording format as reported by IA32_PERF_CAPABILITIES [11:8] is 2.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor tile to subsystems outside the tile (untile). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx. Two cores do not share the off-core response MSRs. Knights Landing expands off-core response capability to match the processor untile changes.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests. This facility is updated to match the processor untile changes.

19.4.1.1.1 Processor Event-Based Sampling

The processor supports processor event based sampling (PEBS). PEBS is supported using IA32_PMC0 (see also Section 17.4.9, "BTS and DS Save Area").

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 19.6.2.4).

The list of PEBS events supported in the processor is shown in the following table.

Table 19-56. PEBS Performance Events for the Knights Landing Microarchitecture

Event Name	Event Select	Sub-event	UMask	Data Linear Address Support
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		CALL	F9H	No
		REL_CALL	FDH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		FAR_BRANCH	BFH	No
		RETURN	F7H	No
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		RETURN	F7H	No
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H	Yes
		L2_MISS_LOADS	04H	Yes
		DLTB_MISS_LOADS	08H	Yes
RECYCLEQ	03H	LD_BLOCK_ST_FORWARD	01H	Yes
		LD_SPLITS	08H	Yes

The PEBS record format 2 supported by processors based on the Knights Landing microarchitecture is shown in Table 19-57, and each field in the PEBS record is 64 bits long.

Table 19-57. PEBS Record Format for the Knights Landing Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	PSDLA
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP

Table 19-57. PEBS Record Format for the Knights Landing Microarchitecture (Contd.)

Byte Offset	Field	Byte Offset	Field
58H	R9	B8H	Reserved

19.4.1.1.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 19-58 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 19-58. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMCO-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMCO-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

Some of the MSR_OFFCORE_RESP [0,1] register bits are not valid in this processor and their use is reserved. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 registers are defined in Table 19-59. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 19.5.2.3 for details.

Table 19-59. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers

Main	Sub-field	Bit	Name	Description
Request Type		0	DEMAND_DATA_RD	Demand cacheable data and L1 prefetch data reads.
		1	DEMAND_RFO	Demand cacheable data writes.
		2	DEMAND_CODE_RD	Demand code reads and prefetch code reads.
		3	Reserved	Reserved.
		4	Reserved	Reserved.
		5	PF_L2_RFO	L2 data RFO prefetches (includes PREFETCHW instruction).
		6	PF_L2_CODE_RD	L2 code HW prefetches.
		7	PARTIAL_READS	Partial reads (UC or WC).
		8	PARTIAL_WRITES	Partial writes (UC or WT or WP). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		9	UC_CODE_READS	UC code reads.
		10	BUS_LOCKS	Bus locks and split lock requests.
		11	FULL_STREAMING_STORES	Full streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		12	SW_PREFETCH	Software prefetches.
		13	PF_L1_DATA_RD	L1 data HW prefetches.
		14	PARTIAL_STREAMING_STORES	Partial streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
15	ANY_REQUEST	Account for any requests.		

Table 19-59. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers (Contd.)

Main	Sub-field	Bit	Name	Description	
Response Type	Any	16	ANY_RESPONSE	Account for any response.	
		Data Supply from Untile	17	NO_SUPP	No Supplier Details.
			18	Reserved	Reserved.
			19	L2_HIT_OTHER_TILE_NEAR	Other tile L2 hit E Near.
			20	Reserved	Reserved.
			21	MCDRAM_NEAR	MCDRAM Local.
			22	MCDRAM_FAR_OR_L2_HIT_OTHER_TILE_FAR	MCDRAM Far or Other tile L2 hit far.
			23	DRAM_NEAR	DRAM Local.
	24	DRAM_FAR	DRAM Far.		
	Data Supply from within same tile	25	L2_HITM_THIS_TILE	M-state.	
		26	L2_HITE_THIS_TILE	E-state.	
		27	L2_HITS_THIS_TILE	S-state.	
		28	L2_HITF_THIS_TILE	F-state.	
		29	Reserved	Reserved.	
		30	Reserved	Reserved.	
	Snoop Info; Only Valid in case of Data Supply from Untile	31	SNOOP_NONE	None of the cores were snooped.	
		32	NO_SNOOP_NEEDED	No snoop was needed to satisfy the request.	
		33	Reserved	Reserved.	
		34	Reserved	Reserved.	
		35	HIT_OTHER_TILE_FWD	Snoop request hit in the other tile with data forwarded.	
		36	HITM_OTHER_TILE	A snoop was needed and it HitM-ed in other core's L1 cache. HitM denotes a cache-line was in modified state before effect as a result of snoop.	
		37	NON_DRAM	Target was non-DRAM system address. This includes MMIO transactions.	
Outstanding requests	Weighted cycles	38	OUTSTANDING (Valid only for MSR_OFFCORE_RESP0. Should only be used on PMCO. This bit is reserved for MSR_OFFCORE_RESP1).	If set, counts total number of weighted cycles of any outstanding offcore requests with data response. Valid only for OFFCORE_RESP_0 event. Should only be used on PMCO. This bit is reserved for OFFCORE_RESP_1 event.	

19.4.1.1.3 Average Offcore Request Latency Measurement

Measurement of average latency of offcore transaction requests can be enabled using MSR_OFFCORE_RSP0.[bit 38] with the choice of request type specified in MSR_OFFCORE_RSP0.[bit 15:0].

Refer to Section 19.5.2.3, "Average Offcore Request Latency Measurement," for typical usage. Note that MSR_OFFCORE_RESPx registers are not shared between cores in Knights Landing. This allows one core to measure average latency while other core is measuring different offcore response events.

19.5 PERFORMANCE MONITORING (INTEL ATOM® PROCESSORS)

19.5.1 Performance Monitoring (45 nm and 32 nm Intel Atom® Processors)

45 nm and 32 nm Intel Atom processors report architectural performance monitoring versionID = 3 (supporting the aggregate capabilities of versionID 1, 2, and 3; see Section 19.2.3) and a host of non-architectural monitoring capabilities. These 45 nm and 32 nm Intel Atom processors provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters. CPUID.0AH:EAX[15:8] reports the MSRs available to software; see Section 19.2.1.

Non-architectural performance monitoring in Intel Atom processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

Architectural and non-architectural performance monitoring events in 45 nm and 32 nm Intel Atom processors support thread qualification using bit 21 (AnyThread) of IA32_PERFEVTSELx MSR, i.e., if IA32_PERFEVTSELx.AnyThread = 1, event counts include monitored conditions due to either logical processors in the same processor core.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 19-6 and described in Section 19.2.1.1 and Section 19.2.3.

Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields that provide the same qualifying actions like those listed in Table 19-77, Table 19-78, Table 19-79, and Table 19-80. One or more of these sub-fields may apply to specific events on an event-by-event basis. Precise Event Based Monitoring is supported using IA32_PMC0 (see also Section 17.4.9, "BTS and DS Save Area").

19.5.2 Performance Monitoring for Silvermont Microarchitecture

Intel processors based on the Silvermont microarchitecture report architectural performance monitoring versionID = 3 (see Section 19.2.3) and a host of non-architectural monitoring capabilities. Intel processors based on the Silvermont microarchitecture provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2). Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture.

Non-architectural performance monitoring in the Silvermont microarchitecture uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 19-6 and described in Section 19.2.1.1 and Section 19.2.3. Architectural and non-architectural performance monitoring events in the Silvermont microarchitecture ignore the AnyThread qualification regardless of its setting in IA32_PERFEVTSELx MSR.

19.5.2.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- The width of counter reported by CPUID.0AH:EAX[23:16] is 40 bits.

- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests.

19.5.2.1.1 Processor Event Based Sampling (PEBS)

In the Silvermont microarchitecture, the PEBS facility can be used with precise events. PEBS is supported using IA32_PMC0 (see also Section 17.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 19.6.2.4).

The list of precise events supported in the Silvermont microarchitecture is shown in Table 19-60.

Table 19-60. PEBS Performance Events for the Silvermont Microarchitecture

Event Name	Event Select	Sub-event	UMask
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
		RETURN	F7H
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H
		L2_MISS_LOADS	04H
		DLTB_MISS_LOADS	08H
		HITM	20H
REHABQ	03H	LD_BLOCK_ST_FORWARD	01H
		LD_SPLITS	08H

PEBS Record Format The PEBS record format supported by processors based on the Intel Silvermont microarchitecture is shown in Table 19-61, and each field in the PEBS record is 64 bits long.

Table 19-61. PEBS Record Format for the Silvermont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	Reserved
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP
58H	R9	B8H	Reserved

19.5.2.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 19-62 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

In the Silvermont microarchitecture, each MSR_OFFCORE_RSPx is shared by two processor cores.

Table 19-62. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMC0-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are shown in Figure 19-40 and Figure 19-41. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 19.5.2.3 for details.

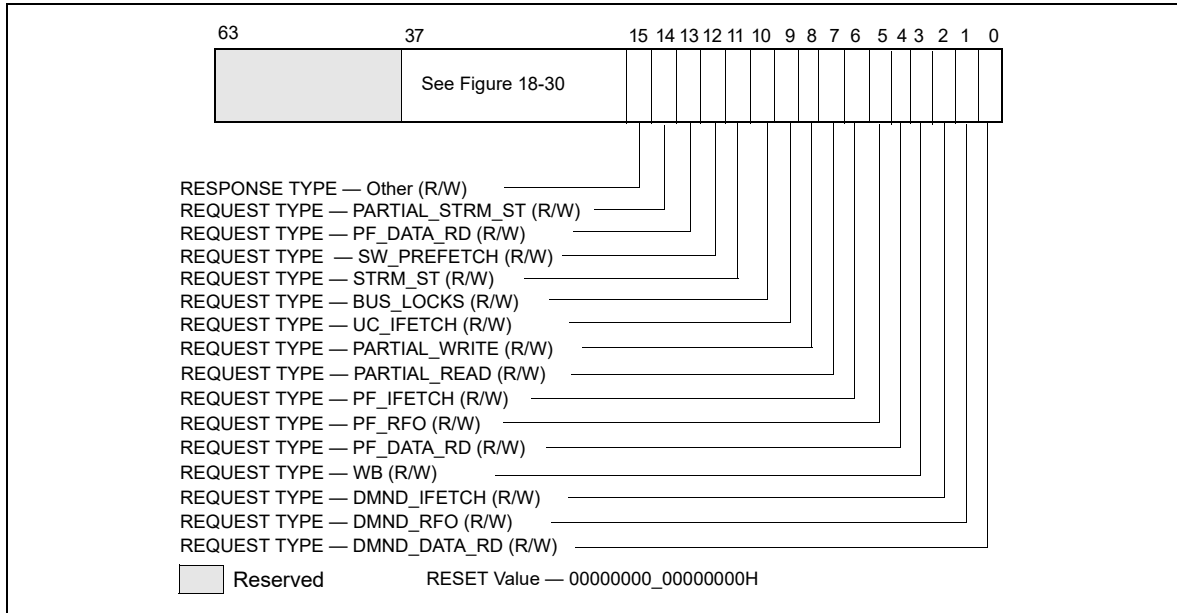


Figure 19-40. Request_Type Fields for MSR_OFFCORE_RSPx

Table 19-63. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PARTIAL_READ	7	Counts the number of demand reads of partial cache lines (including UC and WC).
PARTIAL_WRITE	8	Counts the number of demand RFO requests to write to partial cache lines (includes UC, WT and WP)
UC_IFETCH	9	Counts the number of UC instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests
STRM_ST	11	Streaming store requests
Sw_PREFETCH	12	Counts software prefetch requests
PF_DATA_RD	13	Counts DCU hardware prefetcher data read requests
PARTIAL_STRM_ST	14	Streaming store requests
ANY	15	Any request that crosses IDI, including I/O.

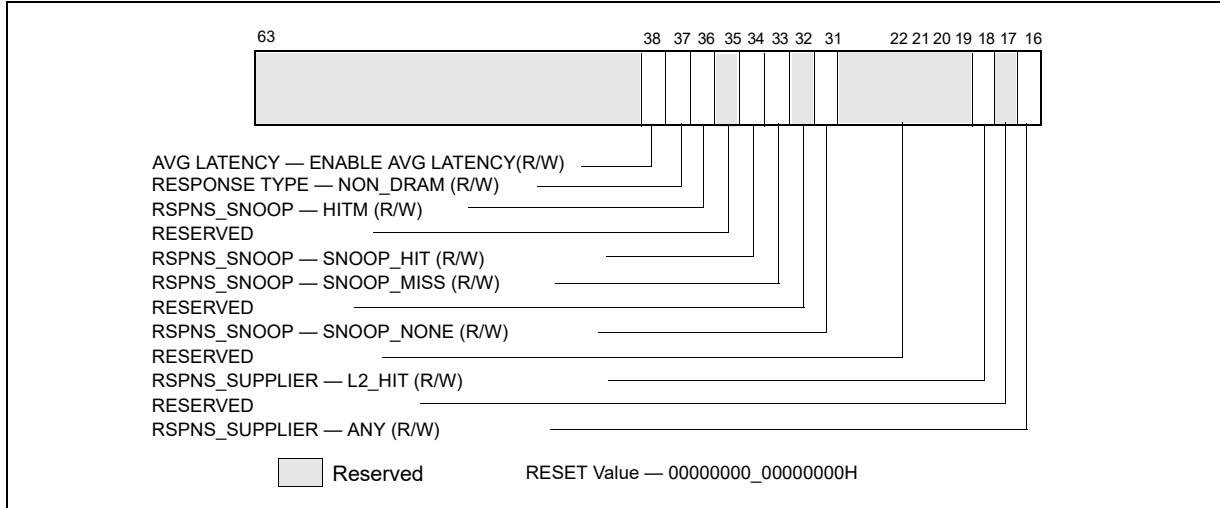


Figure 19-41. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSPx

To properly program this extra register, software must set at least one request type bit (Table 19-63) and a valid response type pattern (Table 19-64, Table 19-65). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 19-64. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	ANY_RESPONSE	16	Catch all value for any response types.
Supplier Info	Reserved	17	Reserved
	L2_HIT	18	Cache reference hit L2 in either M/E/S states.
	Reserved	30:19	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 19-65. MSR_OFFCORE_RSPx Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	Reserved	32	Reserved
	SNOOP_MISS	33	Counts the number of snoop misses when L2 misses.
	SNOOP_HIT	34	Counts the number of snoops hit in the other module where no modified copies were found.
	Reserved	35	Reserved

Table 19-65. MSR_OFFCORE_RSPx Snoop Info Field Definition (Contd.)

Subtype	Bit Name	Offset	Description
	HITM	36	Counts the number of snoops hit in the other module where modified copies were found in other core's L1 cache.
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.
	AVG_LATENCY	38	Enable average latency measurement by counting weighted cycles of outstanding offcore requests of the request type specified in bits 15:0 and any response (bits 37:16 cleared to 0). This bit is available in MSR_OFFCORE_RESP0. The weighted cycles is accumulated in the specified programmable counter IA32_PMCx and the occurrence of specified requests are counted in the other programmable counter.

19.5.2.3 Average Offcore Request Latency Measurement

Average latency for offcore transactions can be determined by using both MSR_OFFCORE_RSP registers. Using two performance monitoring counters, program the two OFFCORE_RESPONSE event encodings into the corresponding IA32_PERFEVTSELx MSRs. Count the weighted cycles via MSR_OFFCORE_RSP0 by programming a request type in MSR_OFFCORE_RSP0.[15:0] and setting MSR_OFFCORE_RSP0.OUTSTANDING[38] to 1, while setting the remaining bits to 0. Count the number of requests via MSR_OFFCORE_RSP1 by programming the same request type from MSR_OFFCORE_RSP0 into MSR_OFFCORE_RSP1[bit 15:0], and setting MSR_OFFCORE_RSP1.ANY_RESPONSE[16] = 1, while setting the remaining bits to 0. The average latency can be obtained by dividing the value of the IA32_PMCx register that counted weight cycles by the register that counted requests.

19.5.3 Performance Monitoring for Goldmont Microarchitecture

Intel Atom processors based on the Goldmont microarchitecture report architectural performance monitoring versionID = 4 (see Section 19.2.4) and support non-architectural monitoring capabilities described in this section. Architectural performance monitoring version 4 capabilities are described in Section 19.2.4.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 19-6 and described in Section 19.2.1.1 and Section 19.2.3. The Goldmont microarchitecture does not support Hyper-Threading and thus architectural and non-architectural performance monitoring events ignore the AnyThread qualification regardless of its setting in the IA32_PERFEVTSELx MSR. However, Goldmont does not set the AnyThread deprecation bit (CPUID.0AH:EDX[15]).

The core PMU's capability is similar to that of the Silvermont microarchitecture described in Section 19.5.2, with some differences and enhancements summarized in Table 19-66.

Table 19-66. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 19.2.1.
# of general-purpose counters per core	4	2	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:40, W:32	See Section 19.2.2.
Architectural Performance Monitoring version ID	4	3	Use CPUID to determine # of counters. See Section 19.2.1.

Table 19-66. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
PMI Overhead Mitigation	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with streamlined semantics. ▪ Freeze_LBR_on_PMI with streamlined semantics for branch profiling. 	<ul style="list-style-type: none"> ▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_LBR_on_PMI with legacy semantics for branch profiling. 	See Section 17.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	<ul style="list-style-type: none"> ▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_STATUS_RESET ▪ Set via IA32_PERF_GLOBAL_STATUS_SET 	<ul style="list-style-type: none"> ▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_OVF_CTRL 	See Section 19.2.4.
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	<ul style="list-style-type: none"> ▪ Individual counter overflow ▪ PEBS buffer overflow ▪ ToPA buffer overflow ▪ CTR_Frz, LBR_Frz 	<ul style="list-style-type: none"> ▪ Individual counter overflow ▪ PEBS buffer overflow 	See Section 19.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	<ul style="list-style-type: none"> ▪ CTR_Frz, ▪ LBR_Frz 	No	See Section 19.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	No	See Section 19.2.4.3.
Processor Event Based Sampling (PEBS) Events	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 19-67.	See Section 19.5.2.1.1. General-Purpose Counter 0 only. Only supports precise events (see Table 19-60).	IA32_PMC0 only.
PEBS record format encoding	0011b	0010b	
Reduce skid PEBS	IA32_PMC0 only	No	
Data Address Profiling	Yes	No	
PEBS record layout	Table 19-68; enhanced fields at offsets 90H- 98H; and TSC record field at C0H.	Table 19-61.	
PEBS EventingIP	Yes	Yes	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register.	MSR 1A6H and 1A7H, shared by a pair of cores.	Nehalem supports 1A6H only.

19.5.3.1 Processor Event Based Sampling (PEBS)

Processor event based sampling (PEBS) on the Goldmont microarchitecture is enhanced over prior generations with respect to sampling support of precise events and non-precise events. In the Goldmont microarchitecture, PEBS is supported using IA32_PMC0 for all events (see Section 17.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor at the time the sample was generated.

Precise events work the same way on Goldmont microarchitecture as on the Silvermont microarchitecture. The record will be generated after an instruction that causes the event when the counter is already overflowed and will capture the architectural state at this point (see Section 19.6.2.4 and Section 17.4.9). The eventingIP in the record will indicate the instruction that caused the event. The list of precise events supported in the Goldmont microarchitecture is shown in Table 19-67.

In the Goldmont microarchitecture, the PEBS facility also supports the use of non-precise events to record processor state information into PEBS records with the same format as with precise events.

However, a non-precise event may not be attributable to a particular retired instruction or the time of instruction execution. When the counter overflows, a PEBS record will be generated at the next opportunity. Consider the event ICACHE.HIT. When the counter overflows, the processor is fetching future instructions. The PEBS record will be generated at the next opportunity and capture the state at the processor's current retirement point. It is likely that the instruction fetch that caused the event to increment was beyond that current retirement point. Other examples of non-precise events are CPU_CLK_UNHALTED.CORE_P and HARDWARE_INTERRUPTS.RECEIVED. CPU_CLK_UNHALTED.CORE_P will increment each cycle that the processor is awake. When the counter over-flows, there may be many instructions in various stages of execution. Additionally, zero, one or multiple instructions may be retired the cycle that the counter overflows. HARDWARE_INTERRUPTS.RECEIVED increments independent of any instructions being executed. For all non-precise events, the PEBS record will be generated at the next opportunity, after the counter has overflowed. The PEBS facility thus allows for identification of the instructions which were executing when the event overflowed.

After generating a record for a non-precise event, the PEBS facility reloads the counter and resumes execution, just as is done for precise events. Unlike interrupt-based sampling, which requires an interrupt service routine to collect the sample and reload the counter, the PEBS facility can collect samples even when interrupts are masked and without using NMI. Since a PEBS record is generated immediately when a counter for a non-precise event is enabled, it may also be generated after an overflow is set by an MSR write to IA32_PERF_GLOBAL_STATUS_SET.

Table 19-67. Precise Events Supported by the Goldmont Microarchitecture

Event Name	Event Select	Sub-event	UMask
LD_BLOCKS	03H	DATA_UNKNOWN	01H
		STORE_FORWARD	02H
		4K_ALIAS	04H
		UTLB_MISS	08H
		ALL_BLOCK	10H
MISALIGN_MEM_REF	13H	LOAD_PAGE_SPLIT	02H
		STORE_PAGE_SPLIT	04H
INST_RETIRED	C0H	ANY	00H
UOPS_RETIRED	C2H	ANY	00H
		LD_SPLITSMS	01H
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
RETURN	F7H		
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H

Table 19-67. Precise Events Supported by the Goldmont Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_UOPS_RETIRED	DOH	ALL_LOADS	81H
		ALL_STORES	82H
		ALL	83H
		DLTB_MISS_LOADS	11H
		DLTB_MISS_STORES	12H
		DLTB_MISS	13H
MEM_LOAD_UOPS_RETIRED	D1H	L1_HIT	01H
		L2_HIT	02H
		L1_MISS	08H
		L2_MISS	10H
		HITM	20H
		WCB_HIT	40H
		DRAM_HIT	80H

The PEBS record format supported by processors based on the Goldmont microarchitecture is shown in Table 19-68, and each field in the PEBS record is 64 bits long.

Table 19-68. PEBS Record Format for the Goldmont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counters
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Reserved
40H	R/EBP	A8H	Reserved
48H	R/ESP	B0H	EventingRIP
50H	R8	B8H	Reserved
58H	R9	C0H	TSC
60H	R10		

On Goldmont microarchitecture, all 64 bits of architectural registers are written into the PEBS record regardless of processor mode.

With PEBS record format encoding 0011b, offset 90H reports the “Applicable Counter” field, which indicates which counters actually requested generating a PEBS record. This allows software to correlate the PEBS record entry properly with the instruction that caused the event even when multiple counters are configured to record PEBS records and multiple bits are set in the field. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

19.5.3.1.1 PEBS Data Linear Address Profiling

Goldmont supports the Data Linear Address field introduced in Haswell. It does not support the Data Source Encoding or Latency Value fields that are also part of Data Address Profiling; those fields are present in the record but are reserved.

For Goldmont microarchitecture, the Data Linear Address field will record the linear address of memory accesses in the previous instruction (e.g., the one that triggered a precise event that caused the PEBS record to be generated). Goldmont microarchitecture may record a Data Linear Address for the instruction that caused the event even for events not related to memory accesses. This may differ from other microarchitectures.

19.5.3.1.2 Reduced Skid PEBS

Processors based on Goldmont Plus microarchitecture support the Reduced Skid PEBS feature described in Section 19.9.4 on the IA32_PMC0 counter. Although Extended PEBS adds support for generating PEBS records for precise events on additional general-purpose and fixed-function performance counters, those counters do not support the Reduced Skid PEBS feature.

19.5.3.1.3 Enhancements to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62]

In addition to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62] being set when PEBS_Index reaches the PEBS_Interrupt_Threshold, the bit is also set when PEBS_Index is out of bounds. That is, the bit will be set when PEBS_Index < PEBS_Buffer_Base or PEBS_Index > PEBS_Absolute_Maximum. Note that when an out of bound condition is encountered, the overflow bits in IA32_PERF_GLOBAL_STATUS will be cleared according to Applicable Counters, however the IA32_PMCx values will not be reloaded with the Reset values stored in the DS_AREA.

19.5.3.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 19-62 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

The Goldmont microarchitecture provides unique pairs of MSR_OFFCORE_RSPx registers per core.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 specifies the request type of a transaction request to the uncore. This is described in Table 19-69.
- Bits 30:16 specifies common supplier information or an L2 Hit, and is described in Table 19-64.
- If L2 misses, then Bits 37:31 can be used to specify snoop response information and is described in Table 19-70.
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 19.5.2.3 for details.

Table 19-69. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts cacheline read requests due to demand reads (excludes prefetches).
DEMAND_RFO	1	Counts cacheline read for ownership (RFO) requests due to demand writes (excludes prefetches).
DEMAND_CODE_RD	2	Counts demand instruction cacheline and I-side prefetch requests that miss the instruction cache.
COREWB	3	Counts writeback transactions caused by L1 or L2 cache evictions.
PF_L2_DATA_RD	4	Counts data cacheline reads generated by hardware L2 cache prefetcher.
PF_L2_RFO	5	Counts reads for ownership (RFO) requests generated by L2 prefetcher.
Reserved	6	Reserved.

Table 19-69. MSR_OFFCORE_RSPx Request_Type Field Definition (Contd.)

Bit Name	Offset	Description
PARTIAL_READS	7	Counts demand data partial reads, including data in uncacheable (UC) or uncacheable (WC) write combining memory types.
PARTIAL_WRITES	8	Counts partial writes, including uncacheable (UC), write through (WT) and write protected (WP) memory type writes.
UC_CODE_READS	9	Counts code reads in uncacheable (UC) memory region.
BUS_LOCKS	10	Counts bus lock and split lock requests.
FULL_STREAMING_STORES	11	Counts full cacheline writes due to streaming stores.
SW_PREFETCH	12	Counts cacheline requests due to software prefetch instructions.
PF_L1_DATA_RD	13	Counts data cacheline reads generated by hardware L1 data cache prefetcher.
PARTIAL_STREAMING_STORES	14	Counts partial cacheline writes due to streaming stores.
ANY_REQUEST	15	Counts requests to the uncore subsystem.

To properly program this extra register, software must set at least one request type bit (Table 19-63) and a valid response type pattern (either Table 19-64 or Table 19-70). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 19-70. MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests

Subtype	Bit Name	Offset	Description
L2_MISS (Snoop Info)	Reserved	32:31	Reserved
	L2_MISS.SNOOP_MISS_OR_NO_SNOOP_NEEDED	33	A true miss to this module, for which a snoop request missed the other module or no snoop was performed/needed.
	L2_MISS.HIT_OTHER_CORE_NO_FWD	34	A snoop hit in the other processor module, but no data forwarding is required.
	Reserved	35	Reserved
	L2_MISS.HITM_OTHER_CORE	36	Counts the number of snoops hit in the other module or other core's L1 where modified copies were found.
	L2_MISS.NON_DRAM	37	Target was a non-DRAM system address. This includes MMIO transactions.
Outstanding requests ¹	OUTSTANDING	38	Counts weighted cycles of outstanding offcore requests of the request type specified in bits 15:0, from the time the XQ receives the request and any response is received. Bits 37:16 must be set to 0. This bit is only available in MSR_OFFCORE_RESPO.

NOTES:

1. See Section 19.5.2.3, "Average Offcore Request Latency Measurement" for details on how to use this bit to extract average latency.

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

Any_Response Bit | L2 Hit | 'OR' of Snoop Info Bits | Outstanding Bit

19.5.3.3 Average Offcore Request Latency Measurement

In Goldmont microarchitecture, measurement of average latency of offcore transaction requests is the same as described in Section 19.5.2.3.

19.5.4 Performance Monitoring for Goldmont Plus Microarchitecture

Intel Atom processors based on the Goldmont Plus microarchitecture report architectural performance monitoring versionID = 4 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 19.2.4.

Goldmont Plus performance monitoring capabilities are similar to Goldmont capabilities. The differences are in specific events and in which counters support PEBS. Goldmont Plus introduces the ability for fixed performance monitoring counters to generate PEBS records.

Goldmont Plus will set the AnyThread deprecation CPUID bit (CPUID.0AH:EDX[15]) to indicate that the Any-Thread bits in IA32_PERFVTSELx and IA32_FIXED_CTR_CTRL have no effect.

The core PMU's capability is similar to that of the Goldmont microarchitecture described in Section 19.6.3, with some differences and enhancements summarized in Table 19-71.

Table 19-71. Core PMU Comparison Between the Goldmont Plus and Goldmont Microarchitectures

Box	Goldmont Plus Microarchitecture	Goldmont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 19.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change.
Architectural Performance Monitoring version ID	4	4	No change.
Processor Event Based Sampling (PEBS) Events	All General-Purpose and Fixed counters. Each General-Purpose counter supports all events (precise and non-precise).	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 19-67.	Goldmont Plus supports PEBS on all counters.
PEBS record format encoding	0011b	0011b	No change.

19.5.4.1 Extended PEBS

The PEBS facility in Goldmont Plus microarchitecture provides a number of enhancements relative to PEBS in processors from previous generations. Enhancement of PEBS facility with the Extended PEBS feature are described in detail in section 18.9.

19.5.5 Performance Monitoring for Tremont Microarchitecture

Intel Atom processors based on the Tremont microarchitecture report architectural performance monitoring versionID = 5 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 5 capabilities are described in Section 19.2.5.

Tremont performance monitoring capabilities are similar to Goldmont Plus capabilities, with the following extensions:

- Support for Adaptive PEBS.
- Support for PEBS output to Intel® Processor Trace.
- Precise Distribution support on Fixed Counter0.
- Compatibility enhancements to off-core response MSRs, MSR_OFFCORE_RSPx.

The differences and enhancements between Tremont microarchitecture and Goldmont Plus microarchitecture are summarized in Table 19-72.

Table 19-72. Core PMU Comparison Between the Tremont and Goldmont Plus Microarchitectures

Box	Tremont Microarchitecture	Goldmont Plus Microarchitecture	Comment
# of fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 19.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change. See Section 19.2.2.
Architectural Performance Monitoring version ID	5	4	
PEBS record format encoding	0100b	0011b	See Section 19.6.2.4.2.
Reduce skid PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 only	
Extended PEBS	Yes	Yes	See Section 19.5.4.1.
Adaptive PEBS	Yes	No	See Section 19.9.2.
PEBS output	DS Save Area or Intel® Processor Trace	DS Save Area only	See Section 19.5.5.2.1.
PEBS record layout	See Section 19.9.2.3 for output to DS, Section 19.5.5.2.2 for output to Intel PT.	Table 19-68; enhanced fields at offsets 90H- 98H; and TSC record field at C0H.	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register.	

19.5.5.1 Adaptive PEBS

The PEBS record format and configuration interface has changed versus Goldmont Plus, as the Tremont microarchitecture includes support for the configurable Adaptive PEBS records; see Section 19.9.2.

19.5.5.2 PEBS output to Intel® Processor Trace

Intel Atom processors based on the Tremont microarchitecture introduce the following Precise Event-Based Sampling (PEBS) extensions:

- A mechanism to direct PEBS output into the Intel® Processor Trace (Intel® PT) output stream. In this scenario, the PEBS record is written in packetized form, in order to co-exist with other Intel PT trace data.
- New Performance Monitoring counter reload MSRs, which are used by PEBS in place of the counter reload values stored in the DS Management area when PEBS output is directed into the Intel PT output stream.

Processors that indicate support for Intel PT by setting CPUID.07H.0.EBX[25]=1, and set the new IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] bit, support these extensions.

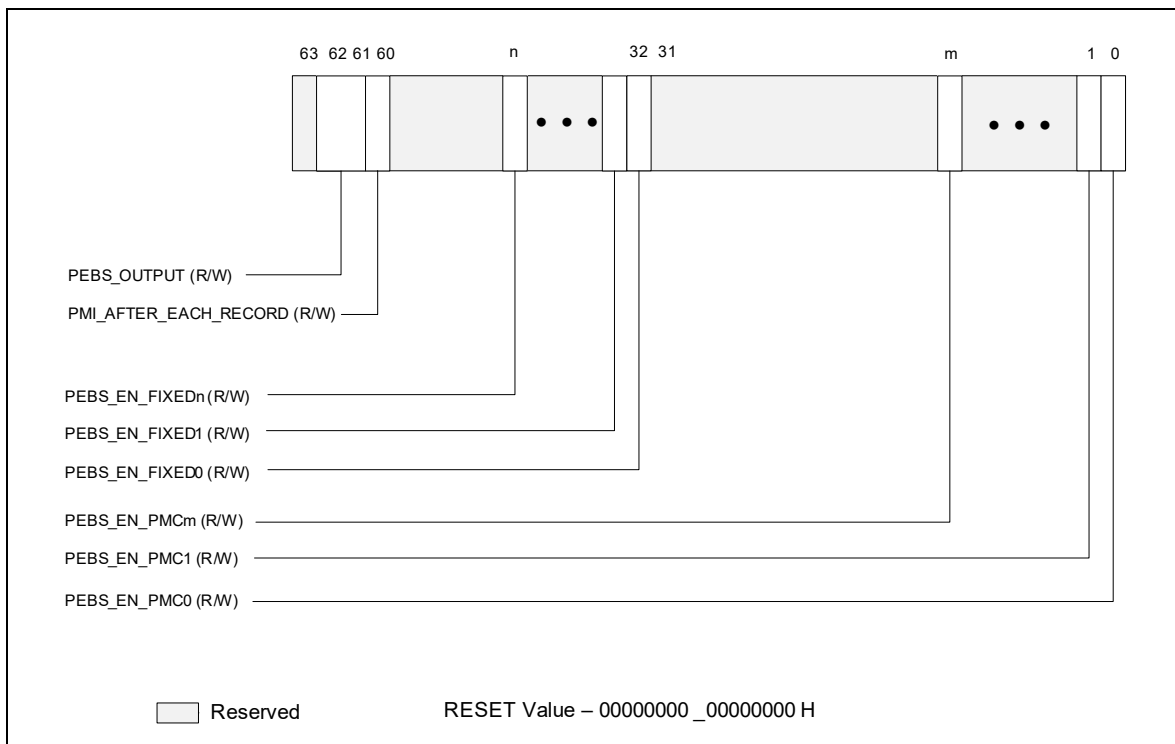
19.5.5.2.1 PEBS Configuration

PEBS output to Intel Processor Trace includes support for two new fields in IA32_PEBS_ENABLE.

Table 19-73. New Fields in IA32_PEBS_ENABLE

Field	Description
PMI_AFTER_EACH_RECORD[60]	Pend a PerfMon Interrupt (PMI) after each PEBS event.
PEBS_OUTPUT[62:61]	Specifies PEBS output destination. Encodings: 00B: DS Save Area. Matches legacy PEBS behavior, output location defined by IA32_DS_AREA. 01B: Intel PT trace output. 10B: Reserved. 11B: Reserved.

When PEBS_OUTPUT is set to 01B, the DS Management Area is not used and need not be configured. Instead, the output mechanism is configured through IA32_RTIT_CTL and other Intel PT MSRs, while counter reload values are configured in the MSR_RELOAD_PMCx MSRs. Details on configuring Intel PT can be found in Section 32.2.7.

**Figure 19-42. IA32_PEBS_ENABLE MSR with PEBS Output to Intel® Processor Trace**

19.5.5.2.2 PEBS Record Format in Intel® Processor Trace

The format of the PEBS record changes when output to Intel PT, as the PEBS state is packetized. Each PEBS grouping is emitted as a Block Begin (BBP) and following Block Item (BIP) packets. A PEBS grouping ends when either a new PEBS grouping begins (indicated by a BBP packet) or a Block End (BEP) packet is encountered. See Section 32.4.1.1 for details of these Intel PT packets.

Because the packet headers describe the state held in the packet payload, PEBS state ordering is not fixed. PEBS state groupings may be emitted in any order, and the PEBS state elements within those groupings may be emitted in any order. Further, there is no packet that provides indication of “Record Format” or “Record Size”.

If Intel PT tracing is not enabled (IA32_RTIT_STATUS.TriggerEn=0), any PEBS records triggered will be dropped. PEBS packets do not depend on ContextEn or FilterEn in IA32_RTIT_STATUS, any filtering of PEBS must be enabled from within the PerfMon configuration. Counter reload will occur in all scenarios where PEBS is triggered, regardless of TriggerEn.

The PEBS threshold mechanism for generating PerfMon Interrupts (PMIs) is not available in this mode. However, there exist other means to generate PMIs based on PEBS output. When the Intel PT ToPA output mechanism is chosen, a PMI can optionally be pended when a ToPA region is filled; see Section 32.2.7.2 for details. Further, software can opt to generate a PMI on each PEBS record by setting the new IA32_PEBS_ENABLE.PMI_AFTER_EACH_RECORD[60] bit.

The IA32_PERF_GLOBAL_STATUS.OvfDSBuffer bit will not be set in this mode.

19.5.5.2.3 PEBS Counter Reload

When PEBS output is directed into Intel PT (IA32_PEBS_ENABLE.PEBS_OUTPUT = 01B), new MSR_RELOAD_PMCx MSRs are used by the PEBS routine to reload PerfMon counters. The value from the associated reload MSR will be loaded to the appropriate counter on each PEBS event.

19.5.5.3 Precise Distribution Support on Fixed Counter 0

The Tremont microarchitecture supports the PDIR (Precise Distribution of Retired Instructions) facility, as described in Section 19.3.4.4.4, on Fixed Counter 0. Fixed Counter 0 counts the INST_RETIRED.ALL event. PEBS skid for Fixed Counter 0 will be precisely one instruction.

This is in addition to the reduced skid PEBS behavior on IA32_PMC0; see Section 19.5.3.1.2.

19.5.5.4 Compatibility Enhancements to Offcore Response MSRs

The Off-core Response facility is similar to that described in Section 19.5.3.2.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as shown below. RequestType bits are defined in Table 19-74, ResponseType bits in Table 19-75, and SnoopInfo bits in Table 19-76.

Table 19-74. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPf_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPf_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPf_L2_CODE_RD	6	Counts all prefetch (that bring data to L2 only) code reads.
Reserved	9:7	Reserved.
HWPf_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
Reserved	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WiL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (WCiL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (WCiLF).

Table 19-74. MSR_OFFCORE_RSPx Request Type Definition (Contd.)

Bit Name	Offset	Description
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

Table 19-75. MSR_OFFCORE_RSPx Response Type Definition

Bit Name	Offset	Description
ANY_RESPONSE	16	Catch all value for any response types.
L3_HIT_M	18	LLC/L3 Hit - M-state.
L3_HIT_E	19	LLC/L3 Hit - E-state.
L3_HIT_S	20	LLC/L3 Hit - S-state.
L3_HIT_F	21	LLC/L3 Hit - I-state.
LOCAL_DRAM	26	LLC/L3 Miss, DRAM Hit.
OUTSTANDING	63	Average latency of outstanding requests with the other counter counting number of occurrences; can also can be used to count occupancy.

Table 19-76. MSR_OFFCORE_RSPx Snoop Info Definition

Bit Name	Offset	Description
SNOOP_NONE	31	None of the cores were snooped. <ul style="list-style-type: none"> ▪ LLC miss and Dram data returned directly to the core.
SNOOP_NOT_NEEDED	32	No snoop needed to satisfy the request. <ul style="list-style-type: none"> ▪ LLC hit and CV bit(s) (core valid) was not set. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_MISS	33	A snoop was sent but missed. <ul style="list-style-type: none"> ▪ LLC hit and CV bit(s) was set but snoop missed (silent data drop in core), data returned from LLC. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_HIT_NO_FWD	34	A snoop was sent but no data forward. <ul style="list-style-type: none"> ▪ LLC hit and CV bit(s) was set but no data forward from the core, data returned from LLC. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_HIT_WITH_FWD	35	A snoop was sent and non-modified data was forward. <ul style="list-style-type: none"> ▪ LLC hit and CV bit(s) was set, non-modified data was forward from core.
SNOOP_HITM	36	A snoop was sent and modified data was forward. <ul style="list-style-type: none"> ▪ LLC hit E or M and the CV bit(s) was set, modified data was forward from core.
NON_DRAM_BIT	37	Target was non-DRAM system address, MMIO access. <ul style="list-style-type: none"> ▪ LLC miss and Non-Dram data returned.

The Off-core Response capability behaves as follows:

- To specify a complete offcore response filter, software must properly program at least one RequestType and one ResponseType. A valid request type must have at least one bit set in the non-reserved bits of 15:0 or 49:44. A valid response type must be a non-zero value of one the following expressions:
 - Read requests:
Any_Response Bit | ('OR' of Supplier Info Bits) 'AND' ('OR' of Snoop Info Bits) | Outstanding Bit
 - Write requests:
Any_Response Bit | ('OR' of Supplier Info Bits) | Outstanding Bit
- When the ANY_RESPONSE bit in the ResponseType is set, all other response type bits will be ignored.
- True Demand Cacheable Loads include neither L1 Prefetches nor Software Prefetches.
- Bits 15:0 and Bits 49:44 specifies the request type of a transaction request to the uncore. This is described in Table 19-74.
- Bits 30:16 specifies common supplier information.
- "Outstanding Requests" (bit 63) is only available on MSR_OFFCORE_RSP0; a #GP fault will occur if software attempts to write a 1 to this bit in MSR_OFFCORE_RSP1. It is mutually exclusive with any ResponseType. Software must guarantee that all other ResponseType bits are set to 0 when the "Outstanding Requests" bit is set.
- "Outstanding Requests" bit 63 can enable measurement of the average latency of a specific type of off-core transaction; two programmable counters must be used simultaneously and the RequestType programming for MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 must be the same when using this Average Latency feature. See Section 19.5.2.3 for further details.

19.6 PERFORMANCE MONITORING (LEGACY INTEL PROCESSORS)

19.6.1 Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)

In Intel Core Solo and Intel Core Duo processors, non-architectural performance monitoring events are programmed using the same facilities (see Figure 19-1) used for architectural performance events.

Non-architectural performance events use event select values that are model-specific. Event mask (Umask) values are also specific to event logic units. Some microarchitectural conditions detectable by a Umask value may have specificity related to processor topology (see Section 8.6, "Detecting Hardware Multi-Threading Support and Topology," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*). As a result, the unit mask field (for example, IA32_PERFEVTSELx[bits 15:8]) may contain sub-fields that specify topology information of processor cores.

The sub-field layout within the Umask field may support two-bit encoding that qualifies the relationship between a microarchitectural condition and the originating core. This data is shown in Table 19-77. The two-bit encoding for core-specificity is only supported for a subset of Umask values (see: <https://perfmon-events.intel.com/>) and for Intel Core Duo processors. Such events are referred to as core-specific events.

Table 19-77. Core Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 15:14 Encoding	Description
11B	All cores
10B	Reserved
01B	This core
00B	Reserved

Some microarchitectural conditions allow detection specificity only at the boundary of physical processors. Some bus events belong to this category, providing specificity between the originating physical processor (a bus agent) versus other agents on the bus. Sub-field encoding for agent specificity is shown in Table 19-78.

Table 19-78. Agent Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13 Encoding	Description
0	This agent
1	Include all agents

Some microarchitectural conditions are detectable only from the originating core. In such cases, unit mask does not support core-specificity or agent-specificity encodings. These are referred to as core-only conditions.

Some microarchitectural conditions allow detection specificity that includes or excludes the action of hardware prefetches. A two-bit encoding may be supported to qualify hardware prefetch actions. Typically, this applies only to some L2 or bus events. The sub-field encoding for hardware prefetch qualification is shown in Table 19-79.

Table 19-79. HW Prefetch Qualification Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13:12 Encoding	Description
11B	All inclusive
10B	Reserved
01B	Hardware prefetch only
00B	Exclude hardware prefetch

Some performance events may (a) support none of the three event-specific qualification encodings (b) may support core-specificity and agent specificity simultaneously (c) or may support core-specificity and hardware prefetch qualification simultaneously. Agent-specificity and hardware prefetch qualification are mutually exclusive.

In addition, some L2 events permit qualifications that distinguish cache coherent states. The sub-field definition for cache coherency state qualification is shown in Table 19-80. If no bits in the MESI qualification sub-field are set for an event that requires setting MESI qualification bits, the event count will not increment.

Table 19-80. MESI Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	Description
Bit 11	Counts modified state
Bit 10	Counts exclusive state
Bit 9	Counts shared state
Bit 8	Counts Invalid state

19.6.2 Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)

In addition to architectural performance monitoring, processors based on the Intel Core microarchitecture support non-architectural performance monitoring events.

Architectural performance events can be collected using general-purpose performance counters. Non-architectural performance events can be collected using general-purpose performance counters (coupled with two IA32_PERFEVTSELx MSRs for detailed event configurations), or fixed-function performance counters (see Section 19.6.2.1). IA32_PERFEVTSELx MSRs are architectural; their layout is shown in Figure 19-1. Starting with Intel

Core 2 processor T 7700, fixed-function performance counters and associated counter control and status MSR becomes part of architectural performance monitoring version 2 facilities (see also Section 19.2.2).

Non-architectural performance events in processors based on Intel Core microarchitecture use event select values that are model-specific. Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields identical to those listed in Table 19-77, Table 19-78, Table 19-79, and Table 19-80. One or more of these sub-fields may apply to specific events on an event-by-event basis.

In addition, the UMASK field may also contain a sub-field that allows detection specificity related to snoop responses. Bits of the snoop response qualification sub-field are defined in Table 19-81.

Table 19-81. Bus Snoop Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	Description
Bit 11	HITM response
Bit 10	Reserved
Bit 9	HIT response
Bit 8	CLEAN response

There are also non-architectural events that support qualification of different types of snoop operation. The corresponding bit field for snoop type qualification are listed in Table 19-82.

Table 19-82. Snoop Type Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 9:8	Description
Bit 9	CMP2I snoops
Bit 8	CMP2S snoops

No more than one sub-field of MESI, snoop response, and snoop type qualification sub-fields can be supported in a performance event.

NOTE

Software must write known values to the performance counters prior to enabling the counters. The content of general-purpose counters and fixed-function counters are undefined after INIT or RESET.

19.6.2.1 Fixed-function Performance Counters

Processors based on Intel Core microarchitecture provide three fixed-function performance counters. Bits beyond the width of the fixed counter are reserved and must be written as zeros. Model-specific fixed-function performance counters on processors that support Architectural Perfmon version 1 are 40 bits wide.

Each of the fixed-function counter is dedicated to count a pre-defined performance monitoring events. See Table 19-2 for details of the PMC addresses and what these events count.

Programming the fixed-function performance counters does not involve any of the IA32_PERFEVTSELx MSRs, and does not require specifying any event masks. Instead, the MSR IA32_FIXED_CTR_CTRL provides multiple sets of 4-bit fields; each 4-bit field controls the operation of a fixed-function performance counter (PMC). See Figures 19-43. Two sub-fields are defined for each control. See Figure 19-43; bit fields are:

- **Enable field (low 2 bits in each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring 0.

When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring greater than 0.

Writing 0 to both bits stops the performance counter. Writing 11B causes the counter to increment irrespective of privilege levels.

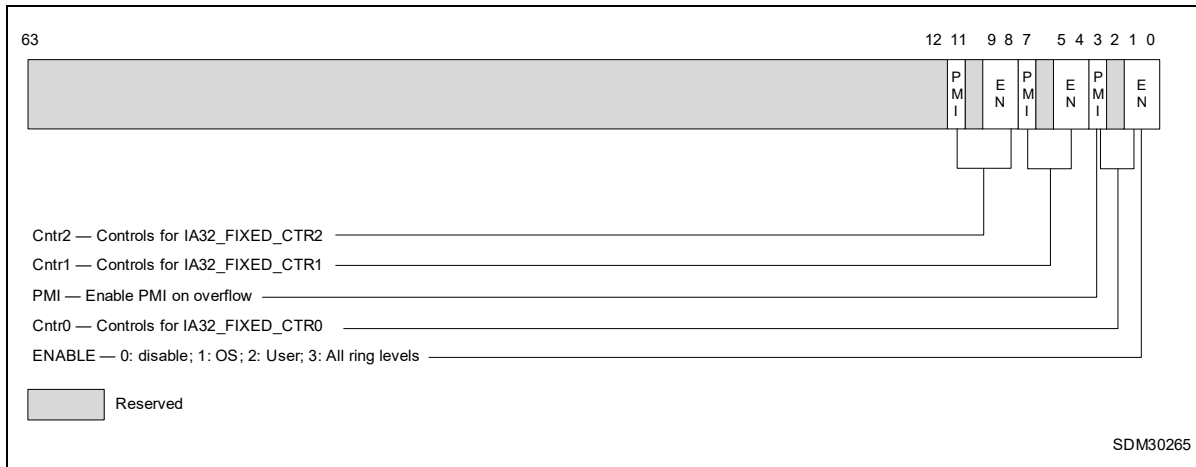


Figure 19-43. Layout of IA32_FIXED_CTR_CTRL MSR

- **PMI field (fourth bit in each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

19.6.2.2 Global Counter Control Facilities

Processors based on Intel Core microarchitecture provides simplified performance counter control that simplifies the most frequent operations in programming performance events, i.e., enabling/disabling event counting and checking the status of counter overflows. This is done by the following three MSRs:

- MSR_PERF_GLOBAL_CTRL enables/disables event counting for all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.
- MSR_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single RDMSR.
- MSR_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.

MSR_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting in each performance counter (see Figure 19-44). Each enable bit in MSR_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

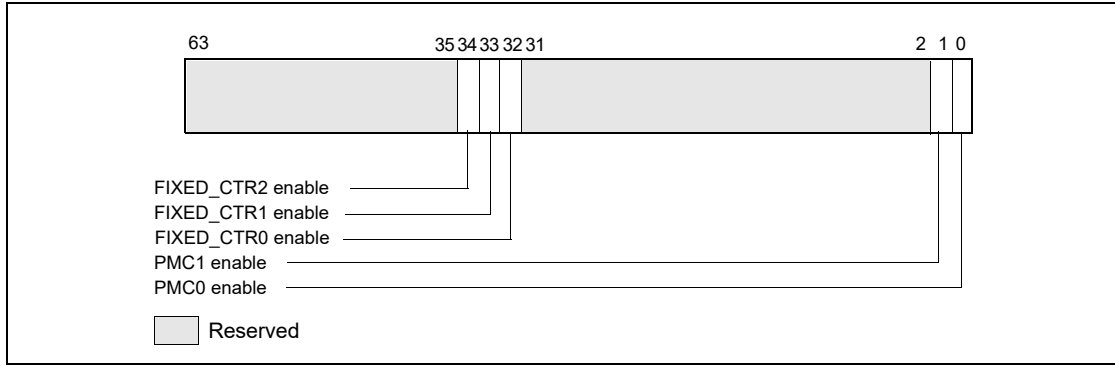


Figure 19-44. Layout of MSR_PERF_GLOBAL_CTRL MSR

MSR_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. MSR_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. MSR_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware (see Figure 19-45). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

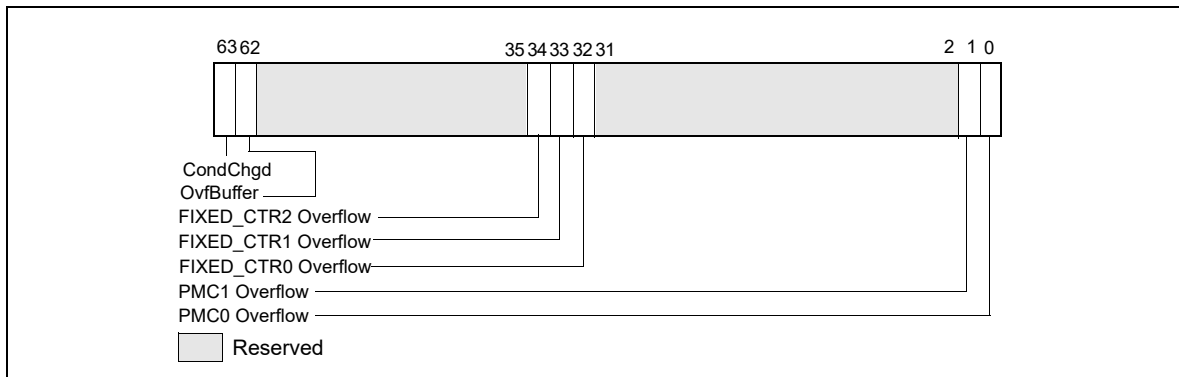


Figure 19-45. Layout of MSR_PERF_GLOBAL_STATUS MSR

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 17.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

MSR_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 19-46). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

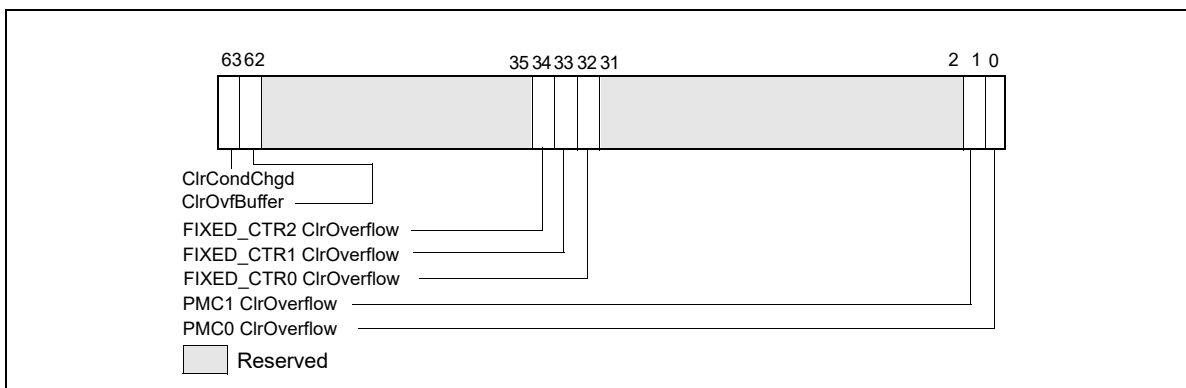


Figure 19-46. Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

19.6.2.3 At-Retirement Events

Many non-architectural performance events are impacted by the speculative nature of out-of-order execution. A subset of non-architectural performance events on processors based on Intel Core microarchitecture are enhanced with a tagging mechanism (similar to that found in Intel NetBurst[®] microarchitecture) that exclude contributions that arise from speculative execution. The at-retirement events available in processors based on Intel Core microarchitecture does not require special MSR programming control (see Section 19.6.3.6, “At-Retirement Counting”), but is limited to IA32_PMC0. See Table 19-83 for a list of events available to processors based on Intel Core microarchitecture.

Table 19-83. At-Retirement Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
ITLB_MISS_RETIRED	00H	C9H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

19.6.2.4 Processor Event Based Sampling (PEBS)

Processors based on Intel Core microarchitecture also support processor event based sampling (PEBS). This feature was introduced by processors based on Intel NetBurst microarchitecture.

PEBS uses a debug store mechanism and a performance monitoring interrupt to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 19.6.2.4.2 and Section 17.4.9).

In cases where the same instruction causes BTS and PEBS to be activated, PEBS is processed before BTS are processed. The PMI request is held until the processor completes processing of PEBS and BTS.

For processors based on Intel Core microarchitecture, precise events that can be used with PEBS are listed in Table 19-84. The procedure for detecting availability of PEBS is the same as described in Section 19.6.3.8.1.

Table 19-84. PEBS Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
INSTR_RETIRED.ANY_P	00H	C0H
X87_OPS_RETIRED.ANY	FEH	C1H
BR_INST_RETIRED.MISPRED	00H	C5H
SIMD_INST_RETIRED.ANY	1FH	C7H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

19.6.2.4.1 Setting up the PEBS Buffer

For processors based on Intel Core microarchitecture, PEBS is available using IA32_PMC0 only. Use the following procedure to set up the processor and IA32_PMC0 counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area. In processors based on Intel Core microarchitecture, PEBS records consist of 64-bit address entries. See Figure 17-8 to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS on PMC0 flag (bit 0) in IA32_PEBS_ENABLE MSR.
3. Set up the IA32_PMC0 performance counter and IA32_PERFEVTSEL0 for an event listed in Table 19-84.

19.6.2.4.2 PEBS Record Format

The PEBS record format may be extended across different processor implementations. The IA32_PERF_CAPABILITIES MSR defines a mechanism for software to handle the evolution of PEBS record format in processors that support architectural performance monitoring with version ID equals 2 or higher. The bit fields of IA32_PERF_CAPABILITIES are defined in Table 2-2 of Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*. The relevant bit fields that governs PEBS are:

- PEBSTrap [bit 6]: When set, PEBS recording is trap-like. After the PEBS-enabled counter has overflowed, PEBS record is recorded for the next PEBS-able event at the completion of the sampled instruction causing the PEBS event. When clear, PEBS recording is fault-like. The PEBS record is recorded before the sampled instruction causing the PEBS event.
- PEBSaveArchRegs [bit 7]: When set, PEBS will save architectural register and state information according to the encoded value of the PEBSRecordFormat field. When clear, only the return instruction pointer and flags are recorded. On processors based on Intel Core microarchitecture, this bit is always 1.
- PEBSRecordFormat [bits 11:8]: Valid encodings are:
 - 0000B: Only general-purpose registers, instruction pointer and RFLAGS registers are saved in each PEBS record (See Section 19.6.3.8).
 - 0001B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS and load latency data. (See Section 19.3.1.1.1).
 - 0010B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS, load latency data, and TSX tuning information. (See Section 19.3.6.2).
 - 0011B: PEBS record includes additional information of load latency data, TSX tuning information, TSC data, and the applicable counter field replaces IA32_PERF_GLOBAL_STATUS at offset 90H. (See Section 19.3.8.1.1).
 - 0100B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 19.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 19-64 with Counter Reset fields allocation for 8 general-purpose counters followed by 4 fixed-function counters.

- 0101B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 19.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 19-64 with Counter Reset fields allocation for 32 general-purpose counters followed by 16 fixed-function counters.

19.6.2.4.3 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the Interrupt-based event sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 17.4.9.1, “64 Bit Format of the DS Save Area,” for guidelines when writing the DS ISR.

The service routine can query MSR_PERF_GLOBAL_STATUS to determine which counter(s) caused of overflow condition. The service routine should clear overflow indicator by writing to MSR_PERF_GLOBAL_OVF_CTL.

A comparison of the sequence of requirements to program PEBS for processors based on Intel Core and Intel NetBurst microarchitectures is listed in Table 19-85.

Table 19-85. Requirements to Program PEBS

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Verify PEBS support of processor/OS.	<ul style="list-style-type: none"> ▪ IA32_MISC_ENABLE.EMON_AVAILABE (bit 7) is set. ▪ IA32_MISC_ENABLE.PEBS_UNAVAILABE (bit 12) is clear. 	
Ensure counters are in disabled.	<p>On initial set up or changing event configurations, write MSR_PERF_GLOBAL_CTRL MSR (38FH) with 0.</p> <p>On subsequent entries:</p> <ul style="list-style-type: none"> ▪ Clear all counters if “Counter Freeze on PMI” is not enabled. ▪ If IA32_DebugCTL.Freeze is enabled, counters are automatically disabled. <p>Counters MUST be stopped before writing.¹</p>	Optional
Disable PEBS.	Clear ENABLE PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Optional
Check overflow conditions.	Check MSR_PERF_GLOBAL_STATUS MSR (38EH) handle any overflow conditions.	Check OVF flag of each CCCR for overflow condition
Clear overflow status.	Clear MSR_PERF_GLOBAL_STATUS MSR (38EH) using IA32_PERF_GLOBAL_OVF_CTRL MSR (390H).	Clear OVF flag of each CCCR.
Write “sample-after” values.	Configure the counter(s) with the sample after value.	
Configure specific counter configuration MSR.	<ul style="list-style-type: none"> ▪ Set local enable bit 22 - 1. ▪ Do NOT set local counter PMI/INT bit, bit 20 - 0. ▪ Event programmed must be PEBS capable. 	<ul style="list-style-type: none"> ▪ Set appropriate OVF_PMI bits - 1. ▪ Only CCCR for MSR_IQ_COUNTER4 support PEBS.
Allocate buffer for PEBS states.	Allocate a buffer in memory for the precise information.	
Program the IA32_DS_AREA MSR.	Program the IA32_DS_AREA MSR.	
Configure the PEBS buffer management records.	Configure the PEBS buffer management records in the DS buffer management area.	
Configure/Enable PEBS.	Set Enable PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Configure MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT and MSR_PEBS_MATRIX_HORZ as needed.
Enable counters.	Set Enable bits in MSR_PERF_GLOBAL_CTRL MSR (38FH).	Set each CCCR enable bit 12 - 1.

NOTES:

1. Counters read while enabled are not guaranteed to be precise with event counts that occur in timing proximity to the RDMSR.

19.6.2.4.4 Re-configuring PEBS Facilities

When software needs to reconfigure PEBS facilities, it should allow a quiescent period between stopping the prior event counting and setting up a new PEBS event. The quiescent period is to allow any latent residual PEBS records to complete its capture at their previously specified buffer address (provided by IA32_DS_AREA).

19.6.3 Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)

The performance monitoring mechanism provided in processors based on Intel NetBurst microarchitecture is different from that provided in the P6 family and Pentium processors. While the general concept of selecting, filtering, counting, and reading performance events through the WRMSR, RDMSR, and RDPMSR instructions is unchanged, the setup mechanism and MSR layouts are incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMSR instruction has been extended to support faster reading of counters and to read all performance counters available in processors based on Intel NetBurst microarchitecture.

The event monitoring mechanism consists of the following facilities:

- The IA32_MISC_ENABLE MSR, which indicates the availability in an Intel 64 or IA-32 processor of the performance monitoring and processor event-based sampling (PEBS) facilities.
- Event selection control (ESCR) MSRs for selecting events to be monitored with specific performance counters. The number available differs by family and model (43 to 45).
- 18 performance counter MSRs for counting events.
- 18 counter configuration control (CCCR) MSRs, with one CCCR associated with each performance counter. CCCRs sets up an associated performance counter for a specific method of counting.
- A debug store (DS) save area in memory for storing PEBS records.
- The IA32_DS_AREA MSR, which establishes the location of the DS save area.
- The debug store (DS) feature flag (bit 21) returned by the CPUID instruction, which indicates the availability of the DS mechanism.
- The MSR_PEBS_ENABLE MSR, which enables the PEBS facilities and replay tagging used in at-retirement event counting.
- A set of predefined events and event metrics that simplify the setting up of the performance counters to count specific events.

Table 19-86 lists the performance counters and their associated CCCRs, along with the ESCRs that select events to be counted for each performance counter. Predefined event metrics and events can be found at: <https://perfmon-events.intel.com/>.

Table 19-86. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Processors Based on Intel NetBurst Microarchitecture)

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCRO	7	3A0H
					MSR_FSB_ESCRO	6	3A2H
					MSR_MOB_ESCRO	2	3AAH
					MSR_PMH_ESCRO	4	3ACH
					MSR_BPU_ESCRO	0	3B2H
					MSR_IS_ESCRO	1	3B4H
					MSR_ITLB_ESCRO	3	3B6H
					MSR_IX_ESCRO	5	3C8H

Table 19-86. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Processors Based on Intel NetBurst Microarchitecture) (Contd.)

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_FLAME_COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAA_T_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_FLAME_COUNTER3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAA_T_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H

Table 19-86. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Processors Based on Intel NetBurst Microarchitecture) (Contd.)

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0	4	3B8H
					MSR_CRU_ESCR2	5	3CCH
					MSR_CRU_ESCR4	6	3E0H
					MSR_IQ_ESCR0 ¹	0	3BAH
					MSR_RAT_ESCR0	2	3BCH
					MSR_SSU_ESCR0	3	3BEH
					MSR_ALF_ESCR0	1	3CAH
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0	4	3B8H
					MSR_CRU_ESCR2	5	3CCH
					MSR_CRU_ESCR4	6	3E0H
					MSR_IQ_ESCR0 ¹	0	3BAH
					MSR_RAT_ESCR0	2	3BCH
					MSR_SSU_ESCR0	3	3BEH
					MSR_ALF_ESCR0	1	3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1	4	3B9H
					MSR_CRU_ESCR3	5	3CDH
					MSR_CRU_ESCR5	6	3E1H
					MSR_IQ_ESCR1 ¹	0	3BBH
					MSR_RAT_ESCR1	2	3BDH
					MSR_ALF_ESCR1	1	3CBH
					MSR_IQ_COUNTER3	15	30FH
MSR_CRU_ESCR3	5	3CDH					
MSR_CRU_ESCR5	6	3E1H					
MSR_IQ_ESCR1 ¹	0	3BBH					
MSR_RAT_ESCR1	2	3BDH					
MSR_ALF_ESCR1	1	3CBH					
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H			
					MSR_CRU_ESCR2	5	3CCH
					MSR_CRU_ESCR4	6	3E0H
					MSR_IQ_ESCR0 ¹	0	3BAH
					MSR_RAT_ESCR0	2	3BCH
					MSR_SSU_ESCR0	3	3BEH
					MSR_ALF_ESCR0	1	3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1	4	3B9H
					MSR_CRU_ESCR3	5	3CDH
					MSR_CRU_ESCR5	6	3E1H
					MSR_IQ_ESCR1 ¹	0	3BBH
					MSR_RAT_ESCR1	2	3BDH
					MSR_ALF_ESCR1	1	3CBH

NOTES:

1. MSR_IQ_ESCR0 and MSR_IQ_ESCR1 are available only on early processor builds (family 0FH, models 01H-02H). These MSRs are not available on later versions.

The types of events that can be counted with these performance monitoring facilities are divided into two classes: non-retirement events and at-retirement events.

- Non-retirement events are events that occur any time during instruction execution (such as bus transactions or cache transactions).
- At-retirement events are events that are counted at the retirement stage of instruction execution, which allows finer granularity in counting events and capturing machine state.

The at-retirement counting mechanism includes facilities for tagging μ ops that have encountered a particular performance event during instruction execution. Tagging allows events to be sorted between those that occurred on an execution path that resulted in architectural state being committed at retirement as well as events that occurred on an execution path where the results were eventually cancelled and never committed to architectural state (such as, the execution of a mispredicted branch).

The Pentium 4 and Intel Xeon processor performance monitoring facilities support the three usage models described below. The first two models can be used to count both non-retirement and at-retirement events; the third model is used to count a subset of at-retirement events:

- **Event counting** — A performance counter is configured to count one or more types of events. While the counter is counting, software reads the counter at selected intervals to determine the number of events that have been counted between the intervals.
- **Interrupt-based event sampling** — A performance counter is configured to count one or more types of events and to generate an interrupt when it overflows. To trigger an overflow, the counter is preset to a modulus value that will cause the counter to overflow after a specific number of events have been counted. When the counter overflows, the processor generates a performance monitoring interrupt (PMI). The interrupt service routine for the PMI then records the return instruction pointer (RIP), resets the modulus, and restarts the counter. Code performance can be analyzed by examining the distribution of RIPs with a tool like the VTune™ Performance Analyzer.
- **Processor event-based sampling (PEBS)** — In PEBS, the processor writes a record of the architectural state of the processor to a memory buffer after the counter overflows. The records of architectural state provide additional information for use in performance tuning. Processor-based event sampling can be used to count only a subset of at-retirement events. PEBS captures more precise processor state information compared to interrupt based event sampling, because the latter need to use the interrupt service routine to re-construct the architectural states of processor.

The following sections describe the MSR and data structures used for performance monitoring in the Pentium 4 and Intel Xeon processors.

19.6.3.1 ESCR MSRs

The 45 ESCR MSRs (see Table 19-86) allow software to select specific events to be countered. Each ESCR is usually associated with a pair of performance counters (see Table 19-86) and each performance counter has several ESCRs associated with it (allowing the events counted to be selected from a variety of events).

Figure 19-47 shows the layout of an ESCR MSR. The functions of the flags and fields are:

- **USR flag, bit 2** — When set, events are counted when the processor is operating at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.
- **OS flag, bit 3** — When set, events are counted when the processor is operating at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the OS and USR flags are set, events are counted at all privilege levels.)

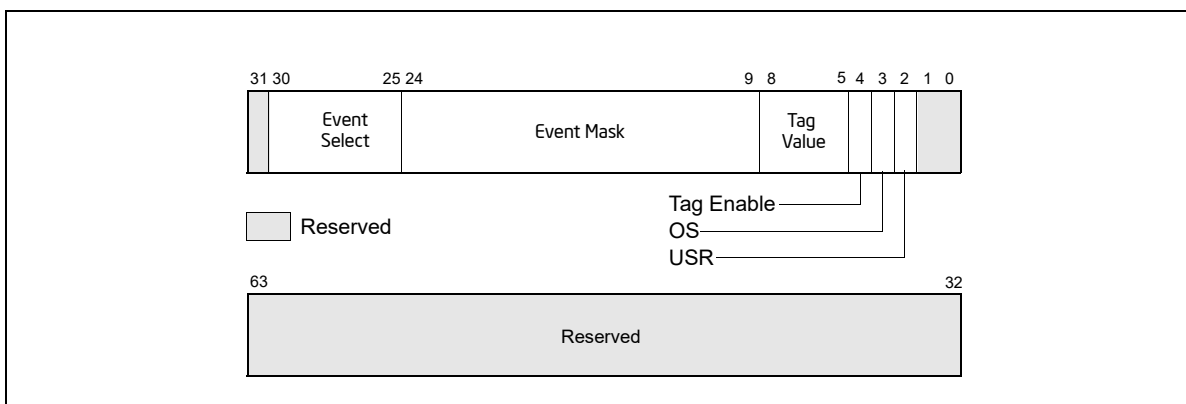


Figure 19-47. Event Selection Control Register (ESCR) for Pentium 4 and Intel® Xeon® Processors without Intel HT Technology Support

- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 19.6.3.6, "At-Retirement Counting."

- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

When setting up an ESCR, the event select field is used to select a specific class of events to count, such as retired branches. The event mask field is then used to select one or more of the specific events within the class to be counted. For example, when counting retired branches, four different events can be counted: branch not taken predicted, branch not taken mispredicted, branch taken predicted, and branch taken mispredicted. The OS and USR flags allow counts to be enabled for events that occur when operating system code and/or application code are being executed. If neither the OS nor USR flag is set, no events will be counted.

The ESCRs are initialized to all 0s on reset. The flags and fields of an ESCR are configured by writing to the ESCR using the WRMSR instruction. Table 19-86 gives the addresses of the ESCR MSRs.

Writing to an ESCR MSR does not enable counting with its associated performance counter; it only selects the event or events to be counted. The CCCR for the selected performance counter must also be configured. Configuration of the CCCR includes selecting the ESCR and enabling the counter.

19.6.3.2 Performance Counters

The performance counters in conjunction with the counter configuration control registers (CCCRs) are used for filtering and counting the events selected by the ESCRs. Processors based on Intel NetBurst microarchitecture provide 18 performance counters organized into 9 pairs. A pair of performance counters is associated with a particular subset of events and ESCR's (see Table 19-86). The counter pairs are partitioned into four groups:

- The BPU group, includes two performance counter pairs:
 - MSR_BPU_COUNTER0 and MSR_BPU_COUNTER1.
 - MSR_BPU_COUNTER2 and MSR_BPU_COUNTER3.
- The MS group, includes two performance counter pairs:
 - MSR_MS_COUNTER0 and MSR_MS_COUNTER1.
 - MSR_MS_COUNTER2 and MSR_MS_COUNTER3.
- The FLAME group, includes two performance counter pairs:
 - MSR_FLAME_COUNTER0 and MSR_FLAME_COUNTER1.
 - MSR_FLAME_COUNTER2 and MSR_FLAME_COUNTER3.
- The IQ group, includes three performance counter pairs:
 - MSR_IQ_COUNTER0 and MSR_IQ_COUNTER1.
 - MSR_IQ_COUNTER2 and MSR_IQ_COUNTER3.
 - MSR_IQ_COUNTER4 and MSR_IQ_COUNTER5.

The MSR_IQ_COUNTER4 counter in the IQ group provides support for the PEBS.

Alternate counters in each group can be cascaded: the first counter in one pair can start the first counter in the second pair and vice versa. A similar cascading is possible for the second counters in each pair. For example, within the BPU group of counters, MSR_BPU_COUNTER0 can start MSR_BPU_COUNTER2 and vice versa, and MSR_BPU_COUNTER1 can start MSR_BPU_COUNTER3 and vice versa (see Section 19.6.3.5.6, "Cascading Counters"). The cascade flag in the CCCR register for the performance counter enables the cascading of counters.

Each performance counter is 40-bits wide (see Figure 19-48). The RDPMC instruction is intended to allow reading of either the full counter-width (40-bits) or, if ECX[31] is set to 1, the low 32-bits of the counter. Reading the low 32-bits is faster than reading the full counter width and is appropriate in situations where the count is small enough to be contained in 32 bits. In such cases, counter bits 31:0 are written to EAX, while 0 is written to EDX.

The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

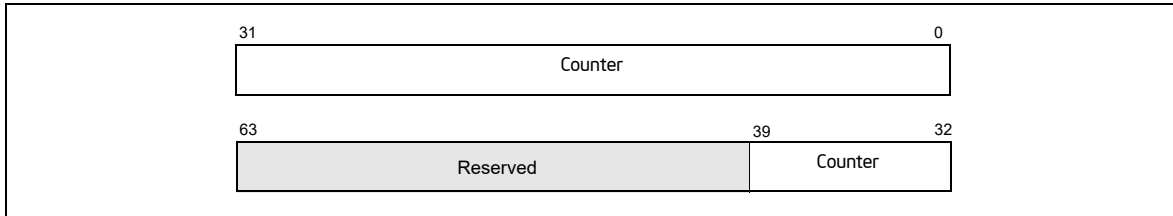


Figure 19-48. Performance Counter (Pentium 4 and Intel® Xeon® Processors)

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

Some uses of the performance counters require the counters to be preset before counting begins (that is, before the counter is enabled). This can be accomplished by writing to the counter using the WRMSR instruction. To set a counter to a specified number of counts before overflow, enter a 2s complement negative integer in the counter. The counter will then count from the preset value up to -1 and overflow. Writing to a performance counter in a Pentium 4 or Intel Xeon processor with the WRMSR instruction causes all 40 bits of the counter to be written.

19.6.3.3 CCCR MSRs

Each of the 18 performance counters has one CCCR MSR associated with it (see Table 19-86). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 19-49 shows the layout of an CCCR MSR. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset.
- **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.
- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 19.6.3.5.2, "Filtering Events"). The complement flag is not active unless the compare flag is set.
- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 19.6.3.5.2, "Filtering Events").
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

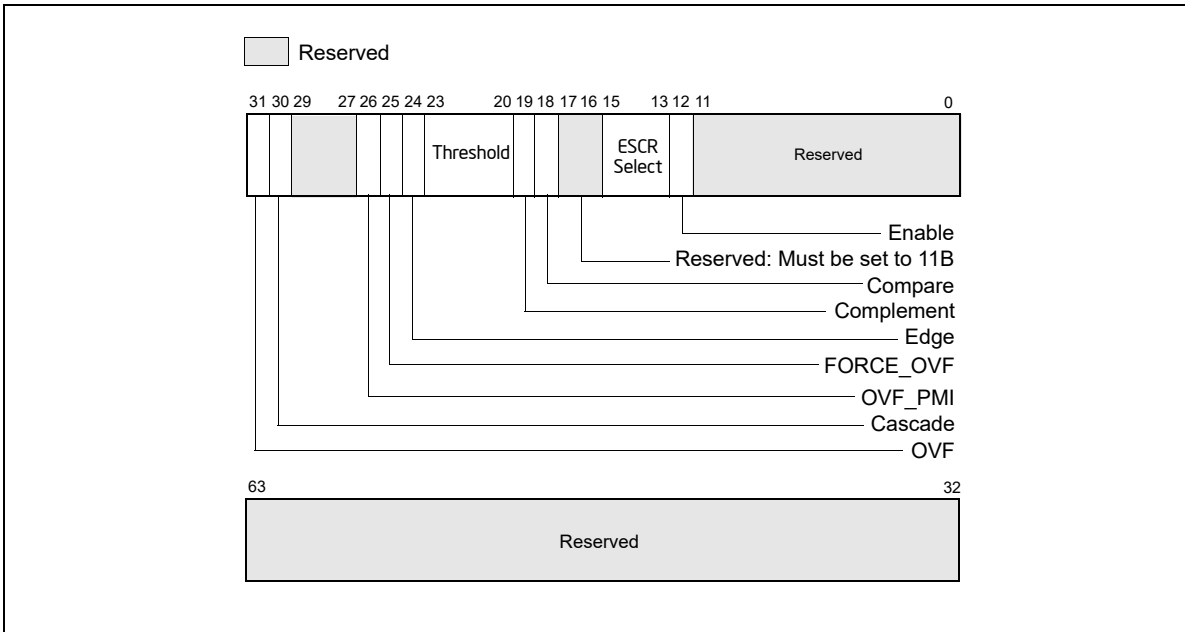


Figure 19-49. Counter Configuration Control Register (CCCR)

- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be generated when the counter overflows occurs; when clear, disables PMI generation. Note that the PMI is generated on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 19.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

The CCCRs are initialized to all 0s on reset.

The events that an enabled performance counter actually counts are selected and filtered by the following flags and fields in the ESCR and CCCR registers and in the qualification order given:

1. The event select and event mask fields in the ESCR select a class of events to be counted and one or more event types within the class, respectively.
2. The OS and USR flags in the ESCR selected the privilege levels at which events will be counted.
3. The ESCR select field of the CCCR selects the ESCR. Since each counter has several ESCRs associated with it, one ESCR must be chosen to select the classes of events that may be counted.
4. The compare and complement flags and the threshold field of the CCCR select an optional threshold to be used in qualifying an event count.
5. The edge flag in the CCCR allows events to be counted only on rising-edge transitions.

The qualification order in the above list implies that the filtered output of one “stage” forms the input for the next. For instance, events filtered using the privilege level flags can be further qualified by the compare and complement flags and the threshold field, and an event that matched the threshold criteria, can be further qualified by edge detection.

The uses of the flags and fields in the CCCRs are discussed in greater detail in Section 19.6.3.5, “Programming the Performance Counters for Non-Retirement Events.”

19.6.3.4 Debug Store (DS) Mechanism

The debug store (DS) mechanism was introduced with processors based on Intel NetBurst microarchitecture to allow various types of information to be collected in memory-resident buffers for use in debugging and tuning programs. The DS mechanism can be used to collect two types of information: branch records and processor event-based sampling (PEBS) records. The availability of the DS mechanism in a processor is indicated with the DS feature flag (bit 21) returned by the CPUID instruction.

See Section 17.4.5, “Branch Trace Store (BTS),” and Section 19.6.3.8, “Processor Event-Based Sampling (PEBS),” for a description of these facilities. Records collected with the DS mechanism are saved in the DS save area. See Section 17.4.9, “BTS and DS Save Area.”

19.6.3.5 Programming the Performance Counters for Non-Retirement Events

The basic steps to program a performance counter and to count events include the following:

1. Select the event or events to be counted.
2. For each event, select an ESCR that supports the event.
3. Match the CCCR Select value and ESCR name to a value listed in Table 19-86; select a CCCR and performance counter.
4. Set up an ESCR for the specific event or events to be counted and the privilege levels at which they are to be counted.
5. Set up the CCCR for the performance counter by selecting the ESCR and the desired event filters.
6. Set up the CCCR for optional cascading of event counts, so that when the selected counter overflows its alternate counter starts.
7. Set up the CCCR to generate an optional performance monitor interrupt (PMI) when the counter overflows. If PMI generation is enabled, the local APIC must be set up to deliver the interrupt to the processor and a handler for the interrupt must be in place.
8. Enable the counter to begin counting.

19.6.3.5.1 Selecting Events to Count

There is a set of at-retirement events for processors based on Intel NetBurst microarchitecture. For each event, setup information is provided. Table 19-87 gives an example of one of the events.

Table 19-87. Event Example

Event Name	Event Parameters	Parameter Value	Description
branch_retired			Counts the retirement of a branch. Specify one or more mask bits to select any combination of branch taken, not-taken, predicted and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 15-3 for the addresses of the ESCR MSRs.
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 15-3.
	ESCR Event Select	06H	ESCR[31:25]
	ESCR Event Mask	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9] Branch Not-taken Predicted Branch Not-taken Mispredicted Branch Taken Predicted Branch Taken Mispredicted
	CCCR Select	05H	CCCR[15:13]

Table 19-87. Event Example (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	Requires Additional MSRs for Tagging	No	

Event Parameters are described below.

- **ESCR restrictions** — Lists the ESCRs that can be used to program the event. Typically only one ESCR is needed to count an event.
- **Counter numbers per ESCR** — Lists which performance counters are associated with each ESCR. Table 19-86 gives the name of the counter and CCCR for each counter number. Typically only one counter is needed to count the event.
- **ESCR event select** — Gives the value to be placed in the event select field of the ESCR to select the event.
- **ESCR event mask** — Gives the value to be placed in the Event Mask field of the ESCR to select sub-events to be counted. The parameter value column defines the documented bits with relative bit position offset starting from 0, where the absolute bit position of relative offset 0 is bit 9 of the ESCR. All undocumented bits are reserved and should be set to 0.
- **CCCR select** — Gives the value to be placed in the ESCR select field of the CCCR associated with the counter to select the ESCR to be used to define the event. This value is not the address of the ESCR; it is the number of the ESCR from the Number column in Table 19-86.
- **Event specific notes** — Gives additional information about the event, such as the name of the same or a similar event defined for the P6 family processors.
- **Can support PEBS** — Indicates if PEBS is supported for the event (only supplied for at-retirement events).
- **Requires additional MSR for tagging** — Indicates which if any additional MSRs must be programmed to count the events (only supplied for the at-retirement events).

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The following procedure shows how to set up a performance counter for basic counting; that is, the counter is set up to count a specified event indefinitely, wrapping around whenever it reaches its maximum count. This procedure is continued through the following four sections.

An event to be counted can be selected as follows:

1. Select the event to be counted.
2. Select the ESCR to be used to select events to be counted from the ESCRs field.
3. Select the number of the counter to be used to count the event from the Counter Numbers Per ESCR field.
4. Determine the name of the counter and the CCCR associated with the counter, and determine the MSR addresses of the counter, CCCR, and ESCR from Table 19-86.
5. Use the WRMSR instruction to write the ESCR Event Select and ESCR Event Mask values into the appropriate fields in the ESCR. At the same time set or clear the USR and OS flags in the ESCR as desired.
6. Use the WRMSR instruction to write the CCCR Select value into the appropriate field in the CCCR.

NOTE

Typically all the fields and flags of the CCCR will be written with one WRMSR instruction; however, in this procedure, several WRMSR writes are used to more clearly demonstrate the uses of the various CCCR fields and flags.

This setup procedure is continued in the next section, Section 19.6.3.5.2, "Filtering Events."

19.6.3.5.2 Filtering Events

Each counter receives up to 4 input lines from the processor hardware from which it is counting events. The counter treats these inputs as binary inputs (input 0 has a value of 1, input 1 has a value of 2, input 3 has a value of 4, and input 3 has a value of 8). When a counter is enabled, it adds this binary input value to the counter value on each clock cycle. For each clock cycle, the value added to the counter can then range from 0 (no event) to 15.

For many events, only the 0 input line is active, so the counter is merely counting the clock cycles during which the 0 input is asserted. However, for some events two or more input lines are used. Here, the counter's threshold setting can be used to filter events. The compare, complement, threshold, and edge fields control the filtering of counter increments by input value.

If the compare flag is set, then a "greater than" or a "less than or equal to" comparison of the input value vs. a threshold value can be made. The complement flag selects "less than or equal to" (flag set) or "greater than" (flag clear). The threshold field selects a threshold value of from 0 to 15. For example, if the complement flag is cleared and the threshold field is set to 6, then any input value of 7 or greater on the 4 inputs to the counter will cause the counter to be incremented by 1, and any value less than 7 will cause an increment of 0 (or no increment) of the counter. Conversely, if the complement flag is set, any value from 0 to 6 will increment the counter and any value from 7 to 15 will not increment the counter. Note that when a threshold condition has been satisfied, the input to the counter is always 1, not the input value that is presented to the threshold filter.

The edge flag provides further filtering of the counter inputs when a threshold comparison is being made. The edge flag is only active when the compare flag is set. When the edge flag is set, the resulting output from the threshold filter (a value of 0 or 1) is used as an input to the edge filter. Each clock cycle, the edge filter examines the last and current input values and sends a count to the counter only when it detects a "rising edge" event; that is, a false-to-true transition. Figure 19-50 illustrates rising edge filtering.

The following procedure shows how to configure a CCCR to filter events using the threshold filter and the edge filter. This procedure is a continuation of the setup procedure introduced in Section 19.6.3.5.1, "Selecting Events to Count."

7. (Optional) To set up the counter for threshold filtering, use the WRMSR instruction to write values in the CCCR compare and complement flags and the threshold field:
 - Set the compare flag.
 - Set or clear the complement flag for less than or equal to or greater than comparisons, respectively.
 - Enter a value from 0 to 15 in the threshold field.
8. (Optional) Select rising edge filtering by setting the CCCR edge flag.

This setup procedure is continued in the next section, Section 19.6.3.5.3, "Starting Event Counting."

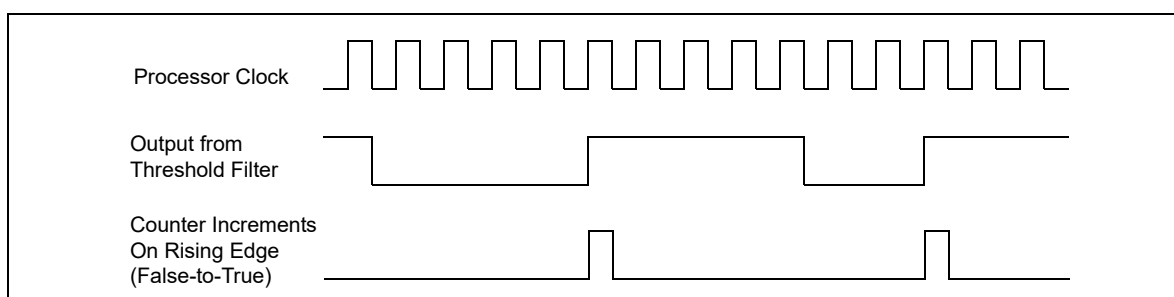


Figure 19-50. Effects of Edge Filtering

19.6.3.5.3 Starting Event Counting

Event counting by a performance counter can be initiated in either of two ways. The typical way is to set the enable flag in the counter's CCCR. Following the instruction to set the enable flag, event counting begins and continues until it is stopped (see Section 19.6.3.5.5, "Halting Event Counting").

The following procedural step shows how to start event counting. This step is a continuation of the setup procedure introduced in Section 19.6.3.5.2, "Filtering Events."

9. To start event counting, use the WRMSR instruction to set the CCCR enable flag for the performance counter.

This setup procedure is continued in the next section, Section 19.6.3.5.4, "Reading a Performance Counter's Count."

The second way that a counter can be started by using the cascade feature. Here, the overflow of one counter automatically starts its alternate counter (see Section 19.6.3.5.6, "Cascading Counters").

19.6.3.5.4 Reading a Performance Counter's Count

Performance counters can be read using either the RDPMC or RDMSR instructions. The enhanced functions of the RDPMC instruction (including fast read) are described in Section 19.6.3.2, "Performance Counters." These instructions can be used to read a performance counter while it is counting or when it is stopped.

The following procedural step shows how to read the event counter. This step is a continuation of the setup procedure introduced in Section 19.6.3.5.3, "Starting Event Counting."

10. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 19-86 used as an operand.

This setup procedure is continued in the next section, Section 19.6.3.5.5, "Halting Event Counting."

19.6.3.5.5 Halting Event Counting

After a performance counter has been started (enabled), it continues counting indefinitely. If the counter overflows (goes one count past its maximum count), it wraps around and continues counting. When the counter wraps around, it sets its OVF flag to indicate that the counter has overflowed. The OVF flag is a sticky flag that indicates that the counter has overflowed at least once since the OVF bit was last cleared.

To halt counting, the CCCR enable flag for the counter must be cleared.

The following procedural step shows how to stop event counting. This step is a continuation of the setup procedure introduced in Section 19.6.3.5.4, "Reading a Performance Counter's Count."

11. To stop event counting, execute a WRMSR instruction to clear the CCCR enable flag for the performance counter.

To halt a cascaded counter (a counter that was started when its alternate counter overflowed), either clear the Cascade flag in the cascaded counter's CCCR MSR or clear the OVF flag in the alternate counter's CCCR MSR.

19.6.3.5.6 Cascading Counters

As described in Section 19.6.3.2, "Performance Counters," eighteen performance counters are implemented in pairs. Nine pairs of counters and associated CCCRs are further organized as four blocks: BPU, MS, FLAME, and IQ (see Table 19-86). The first three blocks contain two pairs each. The IQ block contains three pairs of counters (12 through 17) with associated CCCRs (MSR_IQ_CCCR0 through MSR_IQ_CCCR5).

The first 8 counter pairs (0 through 15) can be programmed using ESCRs to detect performance monitoring events. Pairs of ESCRs in each of the four blocks allow many different types of events to be counted. The cascade flag in the CCCR MSR allows nested monitoring of events to be performed by cascading one counter to a second counter located in another pair in the same block (see Figure 19-49 for the location of the flag).

Counters 0 and 1 form the first pair in the BPU block. Either counter 0 or 1 can be programmed to detect an event via MSR_MO B_ESCR0. Counters 0 and 2 can be cascaded in any order, as can counters 1 and 3. It's possible to set up 4 counters in the same block to cascade on two pairs of independent events. The pairing described also applies to subsequent blocks. Since the IQ PUB has two extra counters, cascading operates somewhat differently if 16 and 17 are involved. In the IQ block, counter 16 can only be cascaded from counter 14 (not from 12); counter 14

cannot be cascaded from counter 16 using the CCCR cascade bit mechanism. Similar restrictions apply to counter 17.

Example 19-1. Counting Events

Assume a scenario where counter X is set up to count 200 occurrences of event A; then counter Y is set up to count 400 occurrences of event B. Each counter is set up to count a specific event and overflow to the next counter. In the above example, counter X is preset for a count of -200 and counter Y for a count of -400; this setup causes the counters to overflow on the 200th and 400th counts respectively.

Continuing this scenario, counter X is set up to count indefinitely and wraparound on overflow. This is described in the basic performance counter setup procedure that begins in Section 19.6.3.5.1, "Selecting Events to Count." Counter Y is set up with the cascade flag in its associated CCCR MSR set to 1 and its enable flag set to 0.

To begin the nested counting, the enable bit for the counter X is set. Once enabled, counter X counts until it overflows. At this point, counter Y is automatically enabled and begins counting. Thus counter X overflows after 200 occurrences of event A. Counter Y then starts, counting 400 occurrences of event B before overflowing. When performance counters are cascaded, the counter Y would typically be set up to generate an interrupt on overflow. This is described in Section 19.6.3.5.8, "Generating an Interrupt on Overflow."

The cascading counters mechanism can be used to count a single event. The counting begins on one counter then continues on the second counter after the first counter overflows. This technique doubles the number of event counts that can be recorded, since the contents of the two counters can be added together.

19.6.3.5.7 EXTENDED CASCADING

Extended cascading is a model-specific feature in the Intel NetBurst microarchitecture with CPUID DisplayFamily_DisplayModel 0F_02, 0F_03, 0F_04, 0F_06. This feature uses bit 11 in CCCRs associated with the IQ block. See Table 19-88.

Table 19-88. CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1 2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INT00	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INT03	Allow counter 5 to cascade into counter 3
MSR_IQ_CCCR4:11	CASCNT5INT04	Allow counter 5 to cascade into counter 4
MSR_IQ_CCCR5:11	CASCNT4INT05	Allow counter 4 to cascade into counter 5

The extended cascading feature can be adapted to the Interrupt based sampling usage model for performance monitoring. However, it is known that performance counters do not generate PMI in cascade mode or extended cascade mode due to an erratum. This erratum applies to processors with CPUID DisplayFamily_DisplayModel signature of 0F_02. For processors with CPUID DisplayFamily_DisplayModel signature of 0F_00 and 0F_01, the erratum applies to processors with stepping encoding greater than 09H.

Counters 16 and 17 in the IQ block are frequently used in processor event-based sampling or at-retirement counting of events indicating a stalled condition in the pipeline. Neither counter 16 or 17 can initiate the cascading of counter pairs using the cascade bit in a CCCR.

Extended cascading permits performance monitoring tools to use counters 16 and 17 to initiate cascading of two counters in the IQ block. Extended cascading from counter 16 and 17 is conceptually similar to cascading other counters, but instead of using CASCADE bit of a CCCR, one of the four CASCNTxINT0y bits is used.

Example 19-2. Scenario for Extended Cascading

A usage scenario for extended cascading is to sample instructions retired on logical processor 1 after the first 4096 instructions retired on logical processor 0. A procedure to program extended cascading in this scenario is outlined below:

1. Write the value 0 to counter 12.
2. Write the value 04000603H to MSR_CRU_ESCR0 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 1).
3. Write the value 04038800H to MSR_IQ_CCCR0. This enables CASCNT4INT00 and OVF_PMI. An ISR can sample on instruction addresses in this case (do not set ENABLE, or CASCADE).
4. Write the value FFFF000H into counter 16.1.
5. Write the value 0400060CH to MSR_CRU_ESCR2 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 0).
6. Write the value 00039000H to MSR_IQ_CCCR4 (set ENABLE bit, but not OVF_PMI).

Another use for cascading is to locate stalled execution in a multithreaded application. Assume MOB replays in thread B cause thread A to stall. Getting a sample of the stalled execution in this scenario could be accomplished by:

1. Set up counter B to count MOB replays on thread B.
2. Set up counter A to count resource stalls on thread A; set its force overflow bit and the appropriate CASCNTx-INTOy bit.
3. Use the performance monitoring interrupt to capture the program execution data of the stalled thread.

19.6.3.5.8 Generating an Interrupt on Overflow

Any performance counter can be configured to generate a performance monitor interrupt (PMI) if the counter overflows. The PMI interrupt service routine can then collect information about the state of the processor or program when overflow occurred. This information can then be used with a tool like the Intel® VTune™ Performance Analyzer to analyze and tune program performance.

To enable an interrupt on counter overflow, the OVR_PMI flag in the counter's associated CCCR MSR must be set. When overflow occurs, a PMI is generated through the local APIC. (Here, the performance counter entry in the local vector table [LVT] is set up to deliver the interrupt generated by the PMI to the processor.)

The PMI service routine can use the OVF flag to determine which counter overflowed when multiple counters have been configured to generate PMIs. Also, note that these processors mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

When generating interrupts on overflow, the performance counter being used should be preset to value that will cause an overflow after a specified number of events are counted plus 1. The simplest way to select the preset value is to write a negative number into the counter, as described in Section 19.6.3.5.6, "Cascading Counters." Here, however, if an interrupt is to be generated after 100 event counts, the counter should be preset to minus 100 plus 1 (-100 + 1), or -99. The counter will then overflow after it counts 99 events and generate an interrupt on the next (100th) event counted. The difference of 1 for this count enables the interrupt to be generated immediately after the selected event count has been reached, instead of waiting for the overflow to be propagation through the counter.

Because of latency in the microarchitecture between the generation of events and the generation of interrupts on overflow, it is sometimes difficult to generate an interrupt close to an event that caused it. In these situations, the FORCE_OVF flag in the CCCR can be used to improve reporting. Setting this flag causes the counter to overflow on every counter increment, which in turn triggers an interrupt after every counter increment.

19.6.3.5.9 Counter Usage Guideline

There are some instances where the user must take care to configure counting logic properly, so that it is not powered down. To use any ESCR, even when it is being used just for tagging, (any) one of the counters that the particular ESCR (or its paired ESCR) can be connected to should be enabled. If this is not done, 0 counts may result. Likewise, to use any counter, there must be some event selected in a corresponding ESCR (other than no_event, which generally has a select value of 0).

19.6.3.6 At-Retirement Counting

At-retirement counting provides a means counting only events that represent work committed to architectural state and ignoring work that was performed speculatively and later discarded.

One example of this speculative activity is branch prediction. When a branch misprediction occurs, the results of instructions that were decoded and executed down the mispredicted path are canceled. If a performance counter was set up to count all executed instructions, the count would include instructions whose results were canceled as well as those whose results committed to architectural state.

To provide finer granularity in event counting in these situations, the performance monitoring facilities provided in the Pentium 4 and Intel Xeon processors provide a mechanism for tagging events and then counting only those tagged events that represent committed results. This mechanism is called “at-retirement counting.”

There are predefined at-retirement events and event metrics that can be used to for tagging events when using at retirement counting. The following terminology is used in describing at-retirement counting:

- **Bogus, non-bogus, retire** — In at-retirement event descriptions, the term “bogus” refers to instructions or μ ops that must be canceled because they are on a path taken from a mispredicted branch. The terms “retired” and “non-bogus” refer to instructions or μ ops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and μ ops are either bogus or non-bogus, but not both. Several of the Pentium 4 and Intel Xeon processors’ performance monitoring events (such as, `Instruction_Retired` and `Uops_Retired`) can count instructions or μ ops that are retired based on the characterization of bogus” versus non-bogus.
- **Tagging** — Tagging is a means of marking μ ops that have encountered a particular performance event so they can be counted at retirement. During the course of execution, the same event can happen more than once per μ op and a direct count of the event would not provide an indication of how many μ ops encountered that event. The tagging mechanisms allow a μ op to be tagged once during its lifetime and thus counted once at retirement. The retired suffix is used for performance metrics that increment a count once per μ op, rather than once per event. For example, a μ op may encounter a cache miss more than once during its life time, but a “Miss Retired” metric (that counts the number of retired μ ops that encountered a cache miss) will increment only once for that μ op. A “Miss Retired” metric would be useful for characterizing the performance of the cache hierarchy for a particular instruction sequence. Details of various performance metrics and how these can be constructed using the Pentium 4 and Intel Xeon processors performance events are provided in the *Intel Pentium 4 Processor Optimization Reference Manual* (see Section 1.4, “Related Literature”).
- **Replay** — To maximize performance for the common case, the Intel NetBurst microarchitecture aggressively schedules μ ops for execution before all the conditions for correct execution are guaranteed to be satisfied. In the event that all of these conditions are not satisfied, μ ops must be reissued. The mechanism that the Pentium 4 and Intel Xeon processors use for this reissuing of μ ops is called replay. Some examples of replay causes are cache misses, dependence violations, and unforeseen resource constraints. In normal operation, some number of replays is common and unavoidable. An excessive number of replays is an indication of a performance problem.
- **Assist** — When the hardware needs the assistance of microcode to deal with some event, the machine takes an assist. One example of this is an underflow condition in the input operands of a floating-point operation. The hardware must internally modify the format of the operands in order to perform the computation. Assists clear the entire machine of μ ops before they begin and are costly.

19.6.3.6.1 Using At-Retirement Counting

Processors based on Intel NetBurst microarchitecture allow counting both events and μ ops that encountered a specified event. For a subset of the at-retirement events, a μ op may be tagged when it encounters that event. The tagging mechanisms can be used in Interrupt-based event sampling, and a subset of these mechanisms can be used in PEBS. There are four independent tagging mechanisms, and each mechanism uses a different event to count μ ops tagged with that mechanism:

- **Front-end tagging** — This mechanism pertains to the tagging of μ ops that encountered front-end events (for example, trace cache and instruction counts) and are counted with the `Front_end_event` event.
- **Execution tagging** — This mechanism pertains to the tagging of μ ops that encountered execution events (for example, instruction types) and are counted with the `Execution_Event` event.

- **Replay tagging** — This mechanism pertains to tagging of μ ops whose retirement is replayed (for example, a cache miss) and are counted with the `Replay_event` event. Branch mispredictions are also tagged with this mechanism.
- **No tags** — This mechanism does not use tags. It uses the `Instr_retired` and the `Uops_retired` events.

Each tagging mechanism is independent from all others; that is, a μ op that has been tagged using one mechanism will not be detected with another mechanism's tagged- μ op detector. For example, if μ ops are tagged using the front-end tagging mechanisms, the `Replay_event` will not count those as tagged μ ops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of μ ops to be counted at retirement through execution tagging.

The independence of tagging mechanisms does not hold when using PEBS. When using PEBS, only one tagging mechanism should be used at a time.

Certain kinds of μ ops that cannot be tagged, including I/O, uncacheable and locked accesses, returns, and far transfers.

There are performance monitoring events that support at-retirement counting: specifically the `Front_end_event`, `Execution_event`, `Replay_event`, `Inst_retired` and `Uops_retired` events. The following sections describe the tagging mechanisms for using these events to tag μ op and count tagged μ ops.

19.6.3.6.2 Tagging Mechanism for `Front_end_event`

The `Front_end_event` counts μ ops that have been tagged as encountering any of the following events:

- **μ op decode events** — Tagging μ ops for μ op decode events requires specifying bits in the `ESCR` associated with the performance-monitoring event, `Uop_type`.
- **Trace cache events** — Tagging μ ops for trace cache events may require specifying certain bits in the `MSR_TC_PRECISE_EVENT` MSR.

The MSRs that are supported by the front-end tagging mechanism must be set and one or both of the `NBOGUS` and `BOGUS` bits in the `Front_end_event` event mask must be set to count events. None of the events currently supported requires the use of the `MSR_TC_PRECISE_EVENT` MSR.

19.6.3.6.3 Tagging Mechanism For `Execution_event`

The execution tagging mechanism differs from other tagging mechanisms in how it causes tagging. One *upstream* `ESCR` is used to specify an event to detect and to specify a tag value (bits 5 through 8) to identify that event. A second *downstream* `ESCR` is used to detect μ ops that have been tagged with that tag value identifier using `Execution_event` for the event selection.

The upstream `ESCR` that counts the event must have its tag enable flag (bit 4) set and must have an appropriate tag value mask entered in its tag value field. The 4-bit tag value mask specifies which of tag bits should be set for a particular μ op. The value selected for the tag value should coincide with the event mask selected in the downstream `ESCR`. For example, if a tag value of 1 is set, then the event mask of `NBOGUS0` should be enabled, correspondingly in the downstream `ESCR`. The downstream `ESCR` detects and counts tagged μ ops. The normal (not tag value) mask bits in the downstream `ESCR` specify which tag bits to count. If any one of the tag bits selected by the mask is set, the related counter is incremented by one. The tag enable and tag value bits are irrelevant for the downstream `ESCR` used to select the `Execution_event`.

The four separate tag bits allow the user to simultaneously but distinctly count up to four execution events at retirement. (This applies for interrupt-based event sampling. There are additional restrictions for PEBS as noted in Section 19.6.3.8.3, "Setting Up the PEBS Buffer.") It is also possible to detect or count combinations of events by setting multiple tag value bits in the upstream `ESCR` or multiple mask bits in the downstream `ESCR`. For example, use a tag value of 3H in the upstream `ESCR` and use `NBOGUS0/NBOGUS1` in the downstream `ESCR` event mask.

19.6.3.7 Tagging Mechanism for `Replay_event`

The replay mechanism enables tagging of μ ops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of μ op that may experience the replay in the `MSR_PEBS_MATRIX_VERT` MSR and selecting the type of event in the `MSR_PEBS_ENABLE` MSR. Replay tagging must also be enabled with the `UOP_Tag` flag (bit 24) in the `MSR_PEBS_ENABLE` MSR.

The replay tags defined in Table A-5 also enable Processor Event-Based Sampling (PEBS, see Section 17.4.9). Each of these replay tags can also be used in normal sampling by not setting Bit 24 nor Bit 25 in IA_32_PEBS_ENABLE_MSR. Each of these metrics requires that the Replay_Event be used to count the tagged μ ops.

19.6.3.8 Processor Event-Based Sampling (PEBS)

The debug store (DS) mechanism in processors based on Intel NetBurst microarchitecture allow two types of information to be collected for use in debugging and tuning programs: PEBS records and BTS records. See Section 17.4.5, “Branch Trace Store (BTS),” for a description of the BTS mechanism.

PEBS permits the saving of precise architectural information associated with one or more performance events in the precise event records buffer, which is part of the DS save area (see Section 17.4.9, “BTS and DS Save Area”). To use this mechanism, a counter is configured to overflow after it has counted a preset number of events. After the counter overflows, the processor copies the current state of the general-purpose and EFLAGS registers and instruction pointer into a record in the precise event records buffer. The processor then resets the count in the performance counter and restarts the counter. When the precise event records buffer is nearly full, an interrupt is generated, allowing the precise event records to be saved. A circular buffer is not supported for precise event records.

PEBS is supported only for a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can only be carried out using the one performance counter, the MSR_IQ_COUNTER4 MSR.

In processors based on Intel Core microarchitecture, a similar PEBS mechanism is also supported using IA32_PMC0 and IA32_PERFEVTSEL0 MSRs (See Section 19.6.2.4).

19.6.3.8.1 Detection of the Availability of the PEBS Facilities

The DS feature flag (bit 21) returned by the CPUID instruction indicates (when set) the availability of the DS mechanism in the processor, which supports the PEBS (and BTS) facilities. When this bit is set, the following PEBS facilities are available:

- The PEBS_UNAVAILABLE flag in the IA32_MISC_ENABLE MSR indicates (when clear) the availability of the PEBS facilities, including the MSR_PEBS_ENABLE MSR.
- The enable PEBS flag (bit 24) in the MSR_PEBS_ENABLE MSR allows PEBS to be enabled (set) or disabled (clear).
- The IA32_DS_AREA MSR can be programmed to point to the DS save area.

19.6.3.8.2 Setting Up the DS Save Area

Section 17.4.9.2, “Setting Up the DS Save Area,” describes how to set up and enable the DS save area. This procedure is common for PEBS and BTS.

19.6.3.8.3 Setting Up the PEBS Buffer

Only the MSR_IQ_COUNTER4 performance counter can be used for PEBS. Use the following procedure to set up the processor and this counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, and precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area (see Figure 17-5) to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS flag (bit 24) in MSR_PEBS_ENABLE MSR.
3. Set up the MSR_IQ_COUNTER4 performance counter and its associated CCCR and one or more ESCRs for PEBS.

19.6.3.8.4 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the non-precise event-based sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 17.4.9.5, “Writing the DS Interrupt Service Routine,” for guidelines for writing the DS ISR.

19.6.3.8.5 Other DS Mechanism Implications

The DS mechanism is not available in the SMM. It is disabled on transition to the SMM mode. Similarly the DS mechanism is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT. The DS mechanism is available in real address mode.

19.6.3.9 Operating System Implications

The DS mechanism can be used by the operating system as a debugging extension to facilitate failure analysis. When using this facility, a 25 to 30 times slowdown can be expected due to the effects of the trace store occurring on every taken branch.

Depending upon intended usage, the instruction pointers that are part of the branch records or the PEBS records need to have an association with the corresponding process. One solution requires the ability for the DS specific operating system module to be chained to the context switch. A separate buffer can then be maintained for each process of interest and the MSR pointing to the configuration area saved and setup appropriately on each context switch.

If the BTS facility has been enabled, then it must be disabled and state stored on transition of the system to a sleep state in which processor context is lost. The state must be restored on return from the sleep state.

It is required that an interrupt gate be used for the DS interrupt as opposed to a trap gate to prevent the generation of an endless interrupt loop.

Pages that contain buffers must have mappings to the same physical address for all processes/logical processors, such that any change to CR3 will not change DS addresses. If this requirement cannot be satisfied (that is, the feature is enabled on a per thread/process basis), then the operating system must ensure that the feature is enabled/disabled appropriately in the context switch code.

19.6.4 Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

The performance monitoring capability of processors based on Intel NetBurst microarchitecture and supporting Intel Hyper-Threading Technology is similar to that described in Section 19.6.3. However, the capability is extended so that:

- Performance counters can be programmed to select events qualified by logical processor IDs.
- Performance monitoring interrupts can be directed to a specific logical processor within the physical processor.

The sections below describe performance counters, event qualification by logical processor ID, and special purpose bits in ESCRs/CCCRs. They also describe MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_TC_PRECISE_EVENT.

19.6.4.1 ESCR MSRs

Figure 19-51 shows the layout of an ESCR MSR in processors supporting Intel Hyper-Threading Technology.

The functions of the flags and fields are as follows:

- **T1_USR flag, bit 0** — When set, events are counted when thread 1 (logical processor 1) is executing at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.

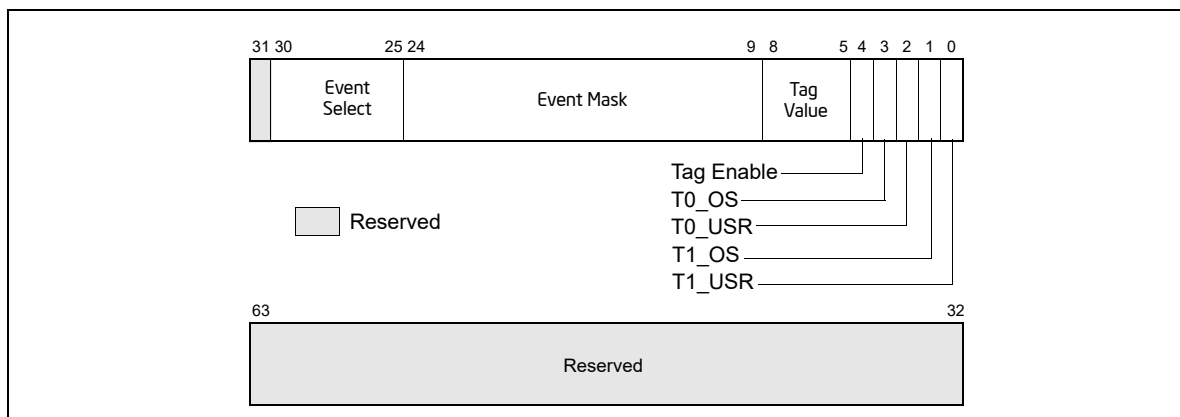


Figure 19-51. Event Selection Control Register (ESCR) for the Pentium 4 Processor, Intel® Xeon® Processor and Intel® Xeon® Processor MP Supporting Hyper-Threading Technology

- **T1_OS flag, bit 1** — When set, events are counted when thread 1 (logical processor 1) is executing at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the T1_OS and T1_USR flags are set, thread 1 events are counted at all privilege levels.)
- **T0_USR flag, bit 2** — When set, events are counted when thread 0 (logical processor 0) is executing at a CPL of 1, 2, or 3.
- **T0_OS flag, bit 3** — When set, events are counted when thread 0 (logical processor 0) is executing at CPL of 0. (When both the T0_OS and T0_USR flags are set, thread 0 events are counted at all privilege levels.)
- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 19.6.3.6, “At-Retirement Counting.”
- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

The T0_OS and T0_USR flags and the T1_OS and T1_USR flags allow event counting and sampling to be specified for a specific logical processor (0 or 1) within an Intel Xeon processor MP (See also: Section 8.4.5, “Identifying Logical Processors in an MP System,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*).

Not all performance monitoring events can be detected within an Intel Xeon processor MP on a per logical processor basis (see Section 19.6.4.4, “Performance Monitoring Events”). Some sub-events (specified by an event mask bits) are counted or sampled without regard to which logical processor is associated with the detected event.

19.6.4.2 CCCR MSRs

Figure 19-52 shows the layout of a CCCR MSR in processors supporting Intel Hyper-Threading Technology. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset
- **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
- **Active thread field, bits 16 and 17** — Enables counting depending on which logical processors are active (executing a thread). This field enables filtering of events based on the state (active or inactive) of the logical processors. The encodings of this field are as follows:
 - 00** — None. Count only when neither logical processor is active.

01 — Single. Count only when one logical processor is active (either 0 or 1).

10 — Both. Count only when both logical processors are active.

11 — Any. Count when either logical processor is active.

A halted logical processor or a logical processor in the “wait for SIPI” state is considered inactive.

- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.

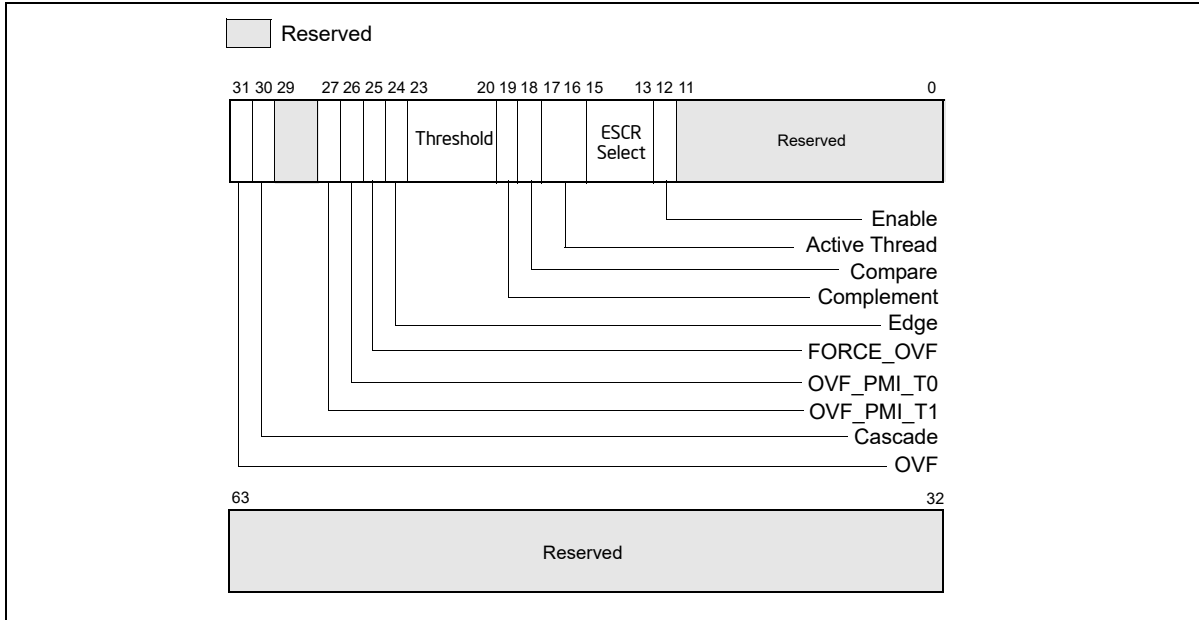


Figure 19-52. Counter Configuration Control Register (CCCR)

- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 19.6.3.5.2, “Filtering Events”). The compare flag is not active unless the compare flag is set.
- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 19.6.3.5.2, “Filtering Events”).
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.
- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI_T0 flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 0 when the counter overflows occurs; when clear, disables PMI generation for logical processor 0. Note that the PMI is generate on the next event count after the counter has overflowed.
- **OVF_PMI_T1 flag, bit 27** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 1 when the counter overflows occurs; when clear, disables PMI generation for logical processor 1. Note that the PMI is generate on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 19.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.

- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

19.6.4.3 IA32_PEBS_ENABLE MSR

In a processor supporting Intel Hyper-Threading Technology and based on the Intel NetBurst microarchitecture, PEBS is enabled and qualified with two bits in the MSR_PEBS_ENABLE MSR: bit 25 (ENABLE_PEBS_MY_THR) and 26 (ENABLE_PEBS_OTH_THR) respectively. These bits do not explicitly identify a specific logical processor by logic processor ID(T0 or T1); instead, they allow a software agent to enable PEBS for subsequent threads of execution on the same logical processor on which the agent is running ("my thread") or for the other logical processor in the physical package on which the agent is not running ("other thread").

PEBS is supported for only a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can be carried out only with two performance counters: MSR_IQ_CCCR4 (MSR address 370H) for logical processor 0 and MSR_IQ_CCCR5 (MSR address 371H) for logical processor 1.

Performance monitoring tools should use a processor affinity mask to bind the kernel mode components that need to modify the ENABLE_PEBS_MY_THR and ENABLE_PEBS_OTH_THR bits in the MSR_PEBS_ENABLE MSR to a specific logical processor. This is to prevent these kernel mode components from migrating between different logical processors due to OS scheduling.

19.6.4.4 Performance Monitoring Events

When Intel Hyper-Threading Technology is active, many performance monitoring events can be qualified by the logical processor ID, which corresponds to bit 0 of the initial APIC ID. This allows for counting an event in any or all of the logical processors. However, not all the events have this logic processor specificity, or thread specificity.

Here, each event falls into one of two categories:

- **Thread specific (TS)** — The event can be qualified as occurring on a specific logical processor.
- **Thread independent (TI)** — The event cannot be qualified as being associated with a specific logical processor.

If for example, a TS event occurred in logical processor T0, the counting of the event (as shown in Table 19-89) depends only on the setting of the T0_USR and T0_OS flags in the ESCR being used to set up the event counter. The T1_USR and T1_OS flags have no effect on the count.

Table 19-89. Effect of Logical Processor and CPL Qualification for Logical-Processor-Specific (TS) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while T1 in USR	Counts while T1 in OS or USR	Counts while T1 in OS
T0_OS/T0_USR = 01	Counts while T0 in USR	Counts while T0 in USR or T1 in USR	Counts while (a) T0 in USR or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 11	Counts while T0 in OS or USR	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) or T0 in USR or (c) T1 in OS
T0_OS/T0_USR = 10	Counts T0 in OS	Counts T0 in OS or T1 in USR	Counts while (a)T0 in Os or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS

When a bit in the event mask field is TI, the effect of specifying bit-0-3 of the associated ESCR are described in Table 15-6. For events that are marked as TI, the effect of selectively specifying T0_USR, T0_OS, T1_USR, T1_OS bits is shown in Table 19-90.

Table 19-90. Effect of Logical Processor and CPL Qualification for Non-logical-Processor-specific (TI) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 01	Counts while (a) T0 in USR or (b) T1 in USR	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 11	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 0	Counts while (a) T0 in OS or (b) T1 in OS	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS

19.6.4.5 Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

19.6.4.5.1 Non-Halted Clockticks

Use the following procedure to program ESCRs and CCCRs to obtain non-halted clockticks on processors based on Intel NetBurst microarchitecture:

1. Select an ESCR for the global_power_events and specify the RUNNING sub-event mask and the desired T0_OS/T0_USR/T1_OS/T1_USR bits for the targeted processor.
2. Select an appropriate counter.
3. Enable counting in the CCCR for that counter by setting the enable bit.

19.6.4.5.2 Non-Sleep Clockticks

Performance monitoring counters can be configured to count clockticks whenever the performance monitoring hardware is not powered-down. To count Non-sleep Clockticks with a performance-monitoring counter, do the following:

1. Select one of the 18 counters.
2. Select any of the ESCRs whose events the selected counter can count. Set its event select to anything other than "no_event"; the counter may be disabled if this is not done.
3. Turn threshold comparison on in the CCCR by setting the compare bit to "1".
4. Set the threshold to "15" and the complement to "1" in the CCCR. Since no event can exceed this threshold, the threshold condition is met every cycle and the counter counts every cycle. Note that this overrides any qualification (e.g., by CPL) specified in the ESCR.
5. Enable counting in the CCCR for the counter by setting the enable bit.

In most cases, the counts produced by the non-halted and non-sleep metrics are equivalent if the physical package supports one logical processor and is not placed in a power-saving state. Operating systems may execute an HLT instruction and place a physical processor in a power-saving state.

On processors that support Intel Hyper-Threading Technology (Intel HT Technology), each physical package can support two or more logical processors. Current implementation of Intel HT Technology provides two logical processors for each physical processor. While both logical processors can execute two threads simultaneously, one logical processor may halt to allow the other logical processor to execute without sharing execution resources between two logical processors.

Non-halted Clockticks can be set up to count the number of processor clock cycles for each logical processor whenever the logical processor is not halted (the count may include some portion of the clock cycles for that logical processor to complete a transition to a halted state). Physical processors that support Intel HT Technology enter into a power-saving state if all logical processors halt.

The Non-sleep Clockticks mechanism uses a filtering mechanism in CCCRs. The mechanism will continue to increment as long as one logical processor is not halted or in a power-saving state. Applications may cause a processor to enter into a power-saving state by using an OS service that transfers control to an OS's idle loop. The idle loop then may place the processor into a power-saving state after an implementation-dependent period if there is no work for the processor.

19.6.5 Performance Monitoring and Dual-Core Technology

The performance monitoring capability of dual-core processors duplicates the microarchitectural resources of a single-core processor implementation. Each processor core has dedicated performance monitoring resources.

In the case of Pentium D processor, each logical processor is associated with dedicated resources for performance monitoring. In the case of Pentium processor Extreme edition, each processor core has dedicated resources, but two logical processors in the same core share performance monitoring resources (see Section 19.6.4, "Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst[®] Microarchitecture").

19.6.6 Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache

The 64-bit Intel Xeon processor MP with up to 8-MByte L3 cache has a CPUID signature of family [0FH], model [03H or 04H]. Performance monitoring capabilities available to Pentium 4 and Intel Xeon processors with the same values (see Section 19.1 and Section 19.6.4) apply to the 64-bit Intel Xeon processor MP with an L3 cache.

The level 3 cache is connected between the system bus and IOQ through additional control logic. See Figure 19-53.

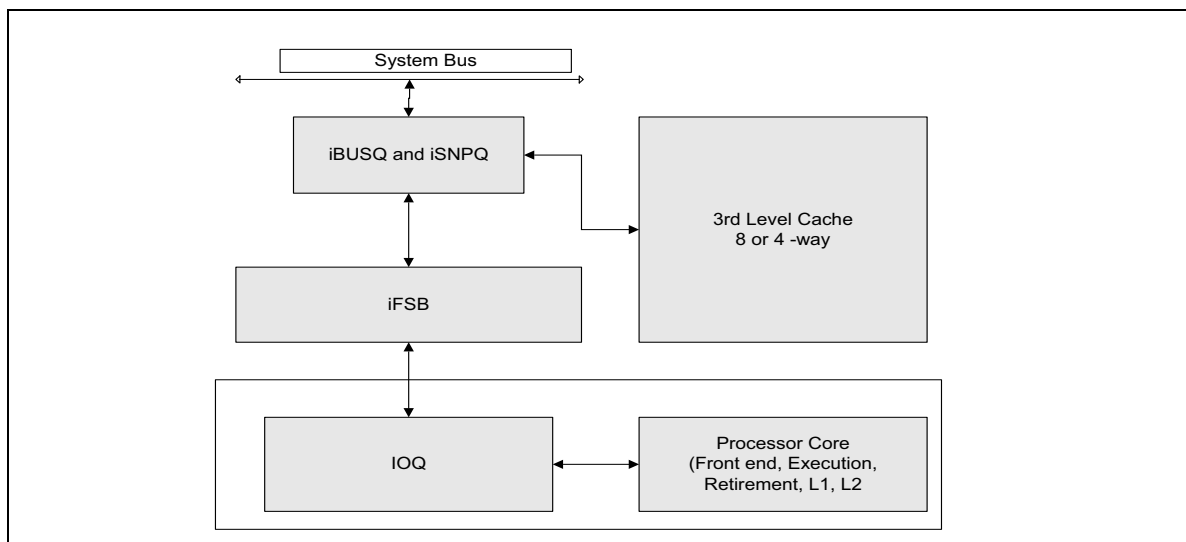


Figure 19-53. Block Diagram of 64-bit Intel[®] Xeon[®] Processor MP with 8-MByte L3

Additional performance monitoring capabilities and facilities unique to 64-bit Intel Xeon processor MP with an L3 cache are described in this section. The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs), each dedicated to a specific event. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values.

The lower 32-bits of the MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers. These performance counters can be accessed using RDPKC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

The performance monitoring capabilities consist of four events. These are:

- IBUSQ event** — This event detects the occurrence of micro-architectural conditions related to the iBUSQ unit. It provides two MSRs: MSR_IFSB_IBUSQ0 and MSR_IFSB_IBUSQ1. Configure sub-event qualification and enable/disable functions using the high 32 bits of these MSRs. The low 32 bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32 bits. See Figure 19-54.

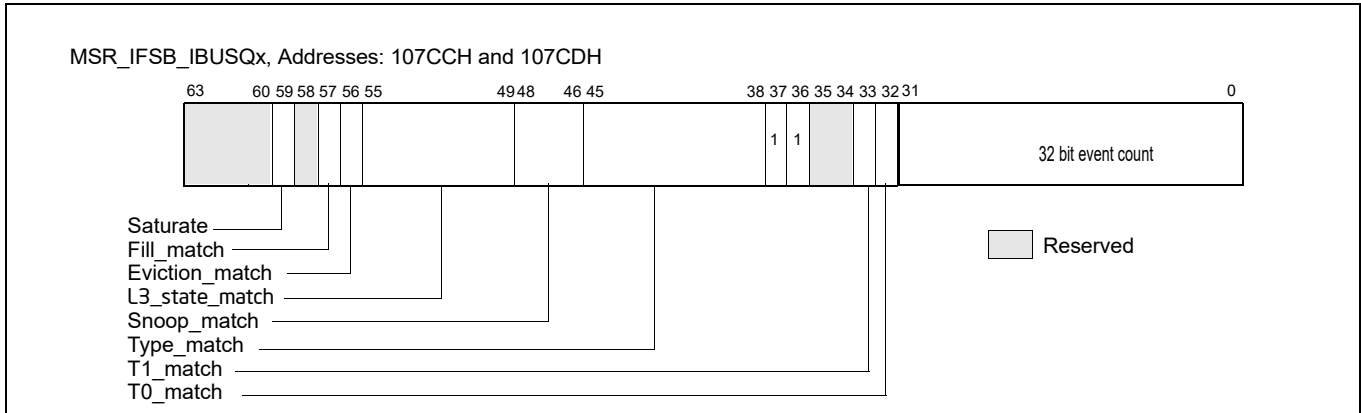


Figure 19-54. MSR_IFSB_IBUSQx, Addresses: 107CCH and 107CDH

- ISNPQ event** — This event detects the occurrence of microarchitectural conditions related to the iSNPQ unit. It provides two MSRs: MSR_IFSB_ISNPQ0 and MSR_IFSB_ISNPQ1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the MSRs. The low 32-bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32-bits. See Figure 19-55.

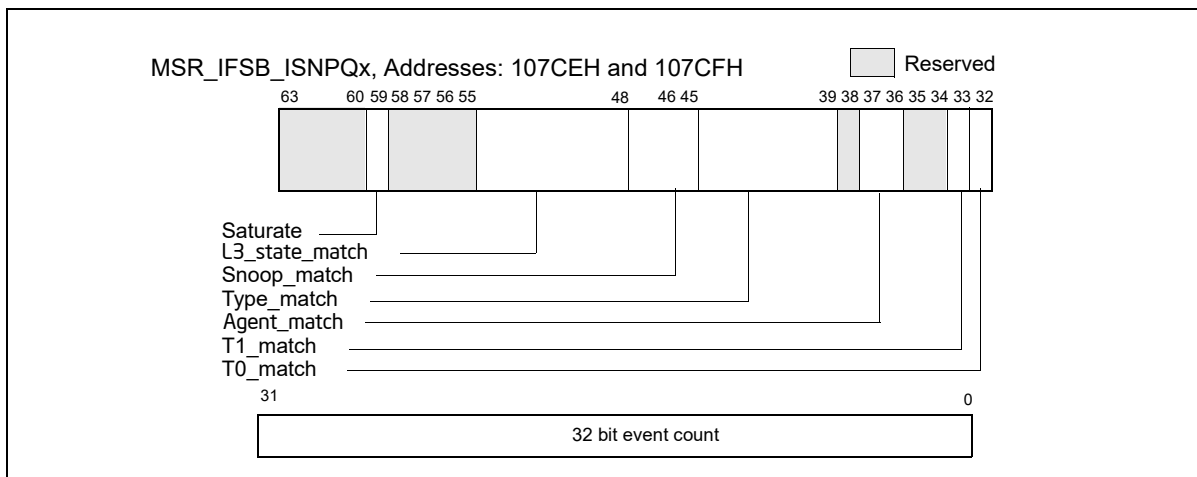


Figure 19-55. MSR_IFSB_ISNPQx, Addresses: 107CEH and 107CFH

- EFSB event** — This event can detect the occurrence of micro-architectural conditions related to the iFSB unit or system bus. It provides two MSRs: MSR_EFSB_DRDY0 and MSR_EFSB_DRDY1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the 64-bit MSR. The low 32-bit act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the qualification bits in the upper 32-bits of the MSR. See Figure 19-56.

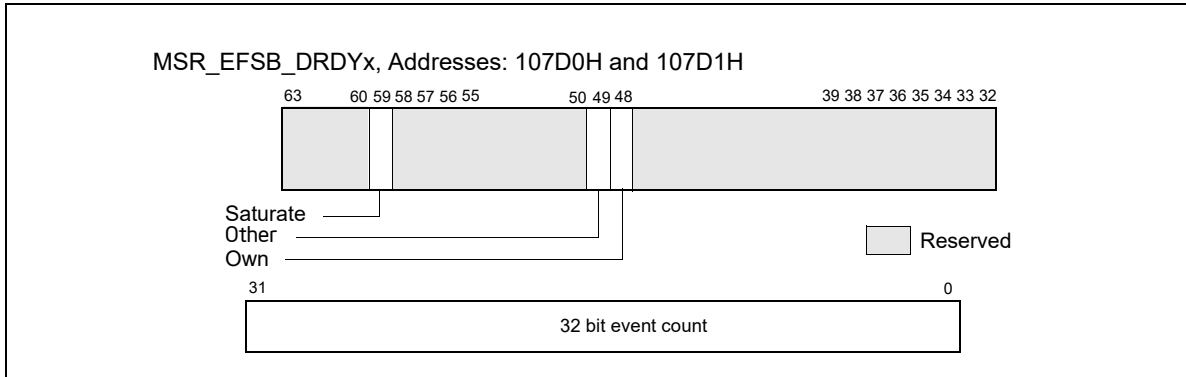


Figure 19-56. MSR_EFSB_DRDYx, Addresses: 107D0H and 107D1H

- IBUSQ Latency event** — This event accumulates weighted cycle counts for latency measurement of transactions in the iBUSQ unit. The count is enabled by setting MSR_IFSB_CTRL6[bit 26] to 1; the count freezes after software sets MSR_IFSB_CTRL6[bit 26] to 0. MSR_IFSB_CNTR7 acts as a 64-bit event counter for this event. See Figure 19-57.

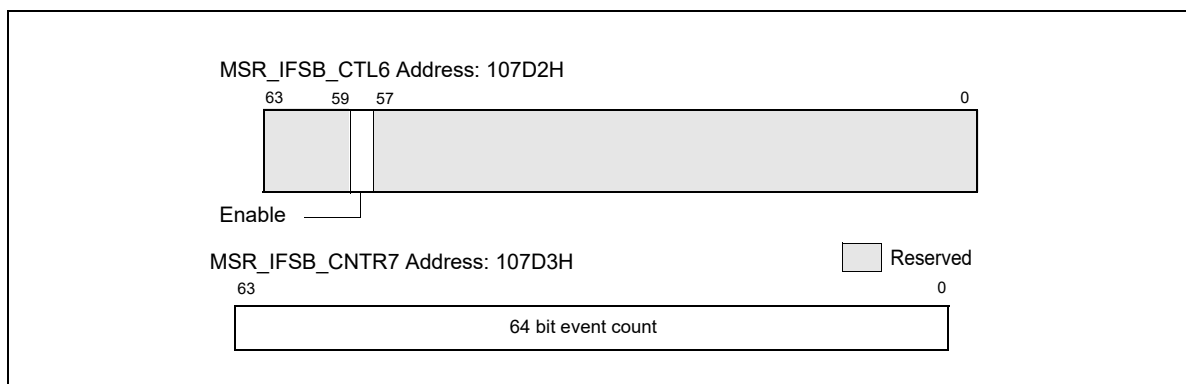


Figure 19-57. MSR_IFSB_CTL6, Address: 107D2H;
 MSR_IFSB_CNTR7, Address: 107D3H

19.6.7 Performance Monitoring on L3 and Caching Bus Controller Sub-Systems

The Intel Xeon processor 7400 series and Dual-Core Intel Xeon processor 7100 series employ a distinct L3/caching bus controller sub-system. These sub-system have a unique set of performance monitoring capability and programming interfaces that are largely common between these two processor families.

Intel Xeon processor 7400 series are based on 45 nm enhanced Intel Core microarchitecture. The CPUID signature is indicated by DisplayFamily_DisplayModel value of 06_1DH (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Intel Xeon processor 7400 series have six processor cores that share an L3 cache.

Dual-Core Intel Xeon processor 7100 series are based on Intel NetBurst microarchitecture, have a CPUID signature of family [0FH], model [06H] and a unified L3 cache shared between two cores. Each core in an Intel Xeon processor 7100 series supports Intel Hyper-Threading Technology, providing two logical processors per core.

Both Intel Xeon processor 7400 series and Intel Xeon processor 7100 series support multi-processor configurations using system bus interfaces. In Intel Xeon processor 7400 series, the L3/caching bus controller sub-system provides three Simple Direct Interface (SDI) to service transactions originated the XQ-replacement SDI logic in each dual-core modules. In Intel Xeon processor 7100 series, the IOQ logic in each processor core is replaced with a Simple Direct Interface (SDI) logic. The L3 cache is connected between the system bus and the SDI through

additional control logic. See Figure 19-58 for the block configuration of six processor cores and the L3/Caching bus controller sub-system in Intel Xeon processor 7400 series. Figure 19-58 shows the block configuration of two processor cores (four logical processors) and the L3/Caching bus controller sub-system in Intel Xeon processor 7100 series.

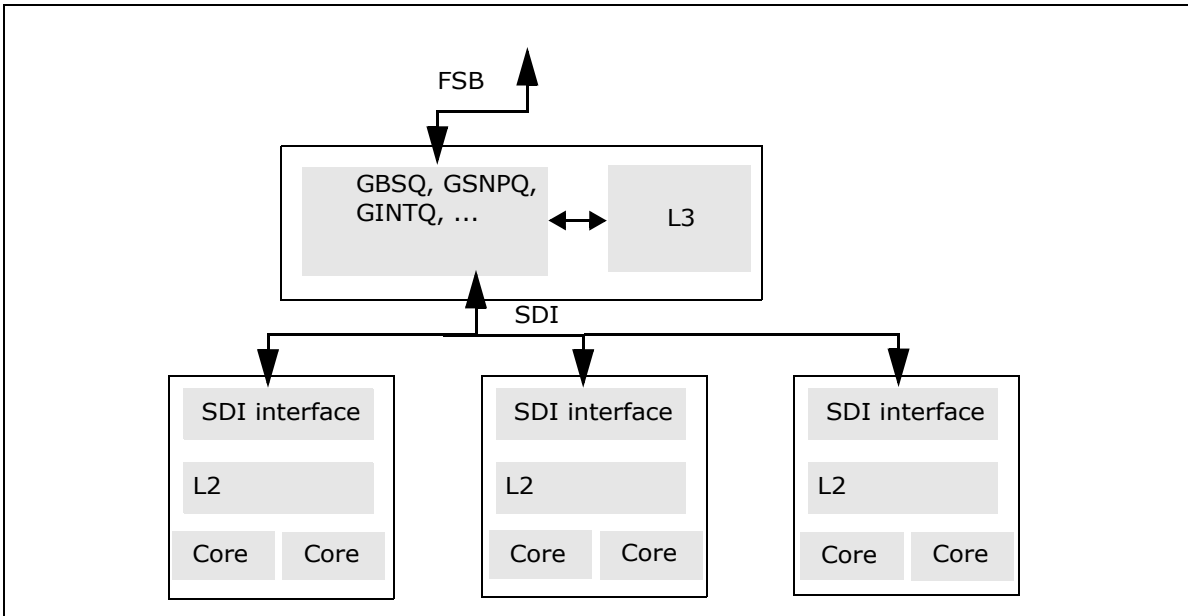


Figure 19-58. Block Diagram of the Intel® Xeon® Processor 7400 Series

Almost all of the performance monitoring capabilities available to processor cores with the same CPUID signatures (see Section 19.1 and Section 19.6.4) apply to Intel Xeon processor 7100 series. The MSR's used by performance monitoring interface are shared between two logical processors in the same processor core.

The performance monitoring capabilities available to processor with DisplayFamily_DisplayModel signature 06_17H also apply to Intel Xeon processor 7400 series. Each processor core provides its own set of MSR's for performance monitoring interface.

The IOQ_allocation and IOQ_active_entries events are not supported in Intel Xeon processor 7100 series and 7400 series. Additional performance monitoring capabilities applicable to the L3/caching bus controller sub-system are described in this section.

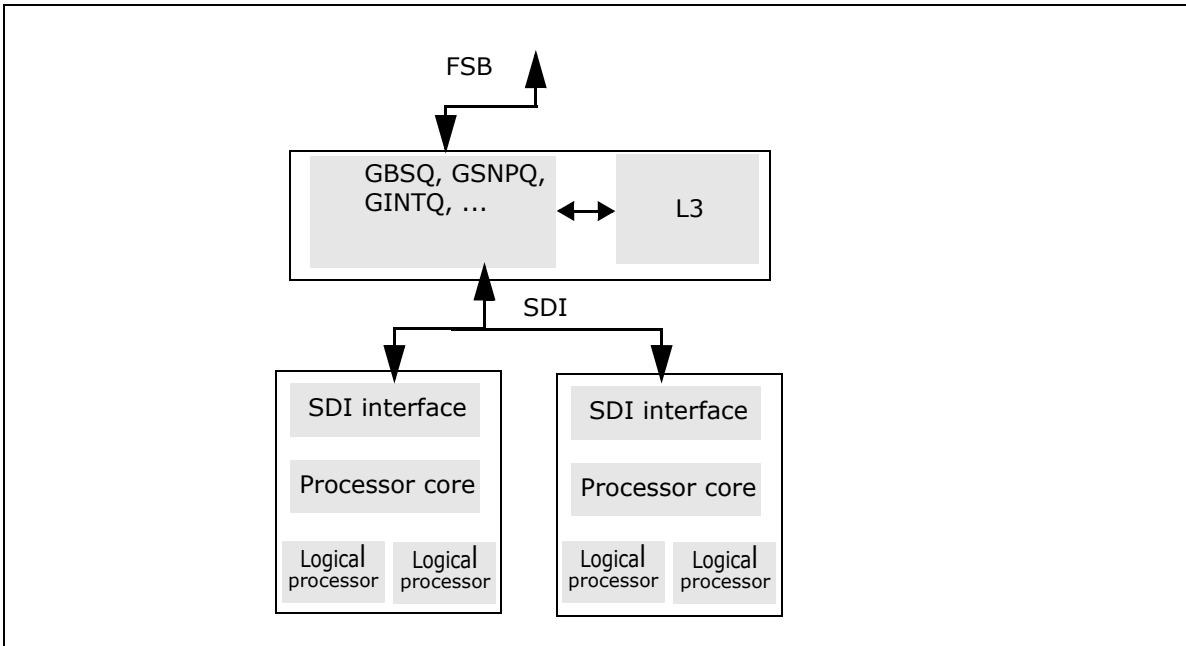


Figure 19-59. Block Diagram of the Intel® Xeon® Processor 7100 Series

19.6.7.1 Overview of Performance Monitoring with L3/Caching Bus Controller

The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs). There are eight event select/counting MSRs that are dedicated to counting events associated with specified microarchitectural conditions. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values. In addition, an MSR MSR_EMON_L3_GL_CTL provides simplified interface to control freezing, resetting, re-enabling operation of any combination of these event select/counting MSRs.

The eight MSRs dedicated to count occurrences of specific conditions are further divided to count three sub-classes of microarchitectural conditions:

- Two MSRs (MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1) are dedicated to counting GBSQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Two MSRs (MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3) are dedicated to counting GSNPQ events. Up to two GSNPQ events can be programmed and counted simultaneously.
- Four MSRs (MSR_EMON_L3_CTR_CTL4, MSR_EMON_L3_CTR_CTL5, MSR_EMON_L3_CTR_CTL6, and MSR_EMON_L3_CTR_CTL7) are dedicated to counting external bus operations.

The bit fields in each of eight MSRs share the following common characteristics:

- Bits 63:32 is the event control field that includes an event mask and other bit fields that control counter operation. The event mask field specifies details of the microarchitectural condition, and its definition differs across GBSQ, GSNPQ, FSB.
- Bits 31:0 is the event count field. If the specified condition is met during each relevant clock domain of the event logic, the matched condition signals the counter logic to increment the associated event count field. The lower 32-bits of these 8 MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers.

In Dual-Core Intel Xeon processor 7100 series, the uncore performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

In Intel Xeon processor 7400 series, RDPMC with ECX between 2 and 9 can be used to access the eight uncore performance counter/control registers.

19.6.7.2 GBSQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1 is given in Figure 19-60. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following eight attributes:

- Agent_Select (bits 35:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, each bit specifies a logical processor in the physical package. The lower two bits corresponds to two logical processors in the first processor core, the upper two bits corresponds to two logical processors in the second processor core. 0FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each bit of [34:32] specifies the SDI logic of a dual-core module as the originator of the transaction. A value of 0111B in bits [35:32] specifies transaction from any processor core.

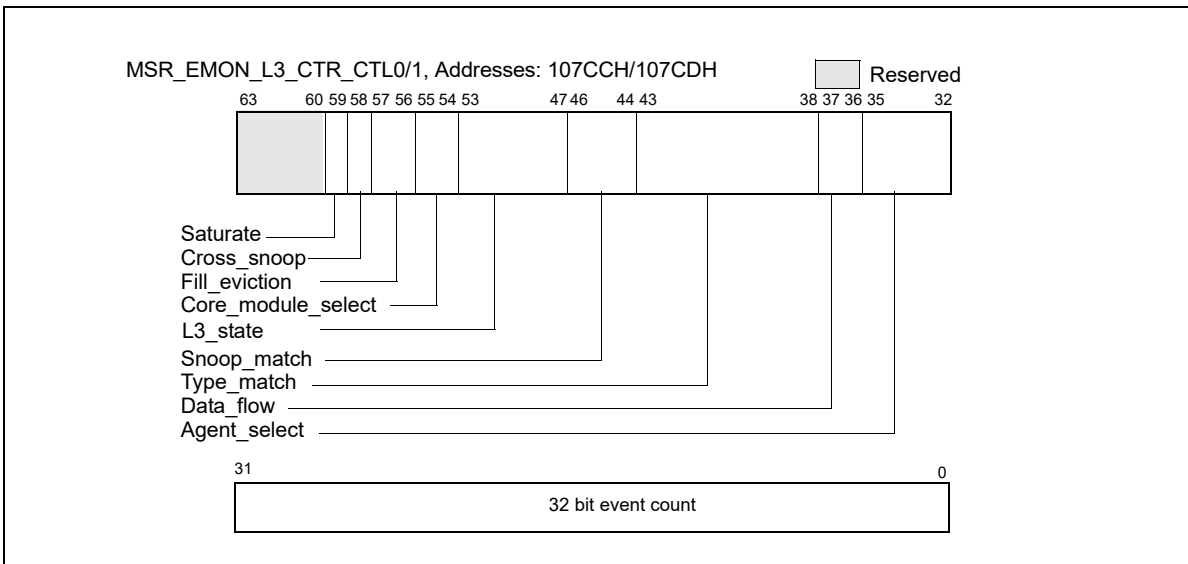


Figure 19-60. MSR_EMON_L3_CTR_CTL0/1, Addresses: 107CCH/107CDH

- Data_Flow (bits 37:36): Bit 36 specifies demand transactions, bit 37 specifies prefetch transactions.
- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include all transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L3_State (bits 53:47): Each bit specifies an L2 coherency state.
- Core_Module_Select (bits 55:54): The valid encodings for L3 lookup differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series,

- 00B: Match transactions from any core in the physical package
- 01B: Match transactions from this core only
- 10B: Match transactions from the other core in the physical package
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series,

- 00B: Match transactions from any dual-core module in the physical package
- 01B: Match transactions from this dual-core module only
- 10B: Match transactions from either one of the other two dual-core modules in the physical package

- 11B: Match transaction from more than one dual-core modules in the physical package
- Fill_Eviction (bits 57:56): The valid encodings are
 - 00B: Match any transactions
 - 01B: Match transactions that fill L3
 - 10B: Match transactions that fill L3 without an eviction
 - 11B: Match transaction fill L3 with an eviction
- Cross_Snoop (bit 58): The encodings are
 - 0B: Match any transactions
 - 1B: Match cross snoop transactions

For each counting clock domain, if all eight attributes match, event logic signals to increment the event count field.

19.6.7.3 GSNPQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3 is given in Figure 19-61. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following six attributes:

- Agent_Select (bits 37:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.
- For Intel Xeon processor 7100 series, each of the lowest 4 bits specifies a logical processor in the physical package. The lowest two bits corresponds to two logical processors in the first processor core, the next two bits corresponds to two logical processors in the second processor core. Bit 36 specifies other symmetric agent transactions. Bit 37 specifies central agent transactions. 3FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each of the lowest 3 bits specifies a dual-core module in the physical package. Bit 37 specifies central agent transactions.

- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include any transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L2_State (bits 53:47): Each bit specifies an L3 coherency state.
- Core_Module_Select (bits 56:54): Bit 56 enables Core_Module_Select matching. If bit 56 is clear, Core_Module_Select encoding is ignored. The valid encodings for the lower two bits (bit 55, 54) differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one core (irrespective which core) in the physical package
- 01B: Match transactions from this core and not the other core
- 10B: Match transactions from the other core in the physical package, but not this core
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one dual-core module (irrespective which module) in the physical package.
- 01B: Match transactions from one or more dual-core modules.
- 10B: Match transactions from two or more dual-core modules.
- 11B: Match transaction from all three dual-core modules in the physical package.

- Block_Snoop (bit 57): specifies blocked snoop.

For each counting clock domain, if all six attributes match, event logic signals to increment the event count field.

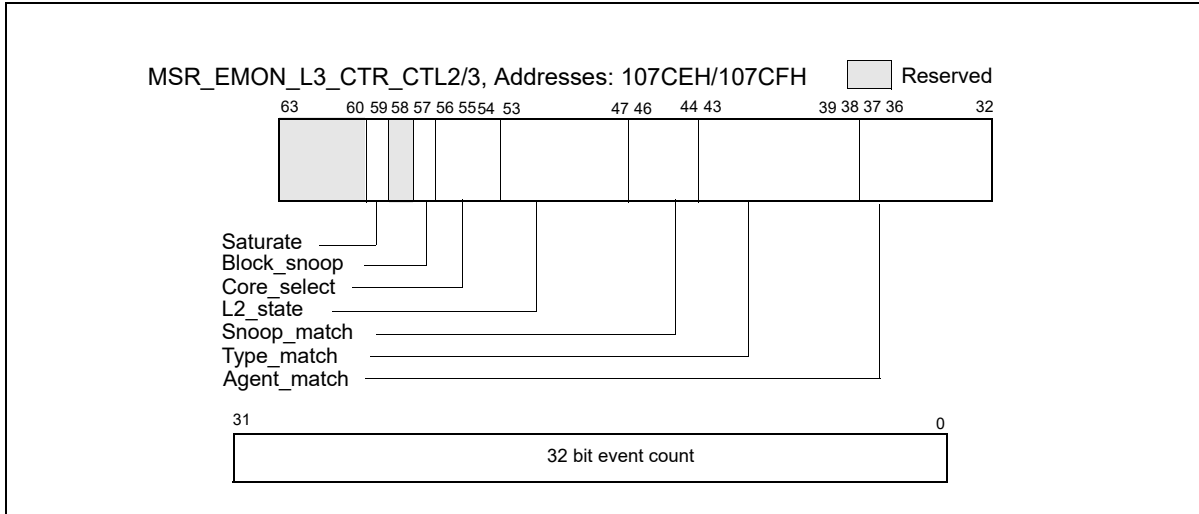


Figure 19-61. MSR_EMON_L3_CTR_CTL2/3, Addresses: 107CEH/107CFH

19.6.7.4 FSB Event Interface

The layout of MSR_EMON_L3_CTR_CTL4 through MSR_EMON_L3_CTR_CTL7 is given in Figure 19-62. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) is organized as follows:

- Bit 58: must set to 1.
- FSB_Submask (bits 57:32): Specifies FSB-specific sub-event mask.

The FSB sub-event mask defines a set of independent attributes. The event logic signals to increment the associated event count field if one of the attribute matches. Some of the sub-event mask bit counts durations. A duration event increments at most once per cycle.

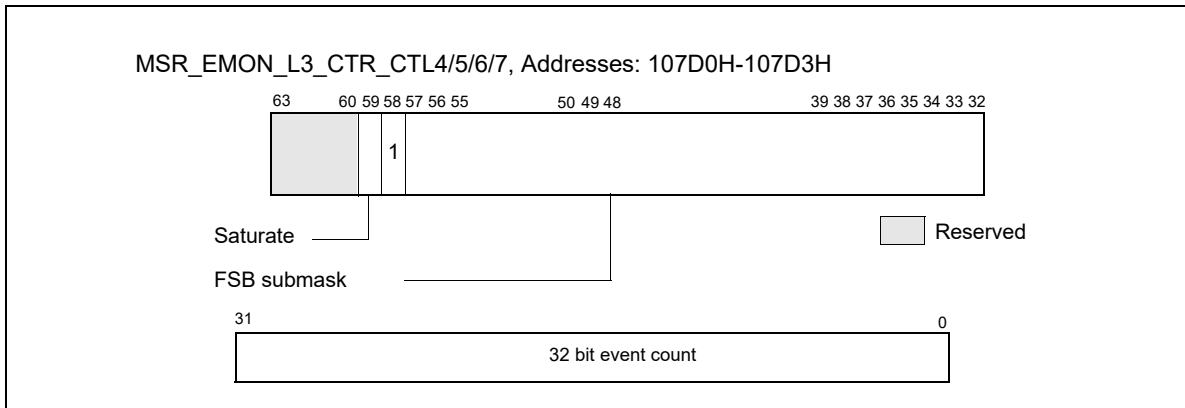


Figure 19-62. MSR_EMON_L3_CTR_CTL4/5/6/7, Addresses: 107D0H-107D3H

19.6.7.4.1 FSB Sub-Event Mask Interface

- FSB_type (bit 37:32): Specifies different FSB transaction types originated from this physical package.
- FSB_L_clear (bit 38): Count clean snoop results from any source for transaction originated from this physical package.
- FSB_L_hit (bit 39): Count HIT snoop results from any source for transaction originated from this physical package.

- FSB_L_hitm (bit 40): Count HITM snoop results from any source for transaction originated from this physical package.
- FSB_L_defer (bit 41): Count DEFER responses to this processor's transactions.
- FSB_L_retry (bit 42): Count RETRY responses to this processor's transactions.
- FSB_L_snoop_stall (bit 43): Count snoop stalls to this processor's transactions.
- FSB_DBSY (bit 44): Count DBSY assertions by this processor (without a concurrent DRDY).
- FSB_DRDY (bit 45): Count DRDY assertions by this processor.
- FSB_BNR (bit 46): Count BNR assertions by this processor.
- FSB_IOQ_empty (bit 47): Counts each bus clocks when the IOQ is empty.
- FSB_IOQ_full (bit 48): Counts each bus clocks when the IOQ is full.
- FSB_IOQ_active (bit 49): Counts each bus clocks when there is at least one entry in the IOQ.
- FSB_WW_data (bit 50): Counts back-to-back write transaction's data phase.
- FSB_WW_issue (bit 51): Counts back-to-back write transaction request pairs issued by this processor.
- FSB_WR_issue (bit 52): Counts back-to-back write-read transaction request pairs issued by this processor.
- FSB_RW_issue (bit 53): Counts back-to-back read-write transaction request pairs issued by this processor.
- FSB_other_DBSY (bit 54): Count DBSY assertions by another agent (without a concurrent DRDY).
- FSB_other_DRDY (bit 55): Count DRDY assertions by another agent.
- FSB_other_snoop_stall (bit 56): Count snoop stalls on the FSB due to another agent.
- FSB_other_BNR (bit 57): Count BNR assertions from another agent.

19.6.7.5 Common Event Control Interface

The MSR_EMON_L3_GL_CTL MSR provides simplified access to query overflow status of the GBSQ, GSNPQ, FSB event counters. It also provides control bit fields to freeze, unfreeze, or reset those counters. The following bit fields are supported:

- GL_freeze_cmd (bit 0): Freeze the event counters specified by the GL_event_select field.
- GL_unfreeze_cmd (bit 1): Unfreeze the event counters specified by the GL_event_select field.
- GL_reset_cmd (bit 2): Clear the event count field of the event counters specified by the GL_event_select field. The event select field is not affected.
- GL_event_select (bit 23:16): Selects one or more event counters to subject to specified command operations indicated by bits 2:0. Bit 16 corresponds to MSR_EMON_L3_CTR_CTL0, bit 23 corresponds to MSR_EMON_L3_CTR_CTL7.
- GL_event_status (bit 55:48): Indicates the overflow status of each event counters. Bit 48 corresponds to MSR_EMON_L3_CTR_CTL0, bit 55 corresponds to MSR_EMON_L3_CTR_CTL7.

In the event control field (bits 63:32) of each MSR, if the saturate control (bit 59, see Figure 19-60 for example) is set, the event logic forces the value FFFF_FFFFH into the event count field instead of incrementing it.

19.6.8 Performance Monitoring (P6 Family Processor)

The P6 family processors provide two 40-bit performance counters, allowing two types of events to be monitored simultaneously. These can either count events or measure duration. When counting events, a counter increments each time a specified event takes place or a specified number of events takes place. When measuring duration, it counts the number of processor clocks that occur while a specified condition is true. The counters can count events or measure durations that occur at any privilege level.

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The performance-monitoring counters are supported by four MSRs: the performance event select MSRs (PerfEvtSel0 and PerfEvtSel1) and the performance counter MSRs (PerfCtr0 and PerfCtr1). These registers can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0. The PerfCtr0 and PerfCtr1 MSRs can be read from any privilege level using the RDPMC (read performance-monitoring counters) instruction.

NOTE

The PerfEvtSel0, PerfEvtSel1, PerfCtr0, and PerfCtr1 MSRs and the events listed for P6 family processors are model-specific for P6 family processors. They are not guaranteed to be available in other IA-32 processors.

19.6.8.1 PerfEvtSel0 and PerfEvtSel1 MSRs

The PerfEvtSel0 and PerfEvtSel1 MSRs control the operation of the performance-monitoring counters, with one register used to set up each counter. They specify the events to be counted, how they should be counted, and the privilege levels at which counting should take place. Figure 19-63 shows the flags and fields in these MSRs.

The functions of the flags and fields in the PerfEvtSel0 and PerfEvtSel1 MSRs are as follows:

- **Event select field (bits 0 through 7)** — Selects the event logic unit to detect certain microarchitectural conditions.
- **Unit mask (UMASK) field (bits 8 through 15)** — Further qualifies the event logic unit selected in the event select field to detect a specific microarchitectural condition. For example, for some cache events, the mask is used as a MESI-protocol qualifier of cache states.

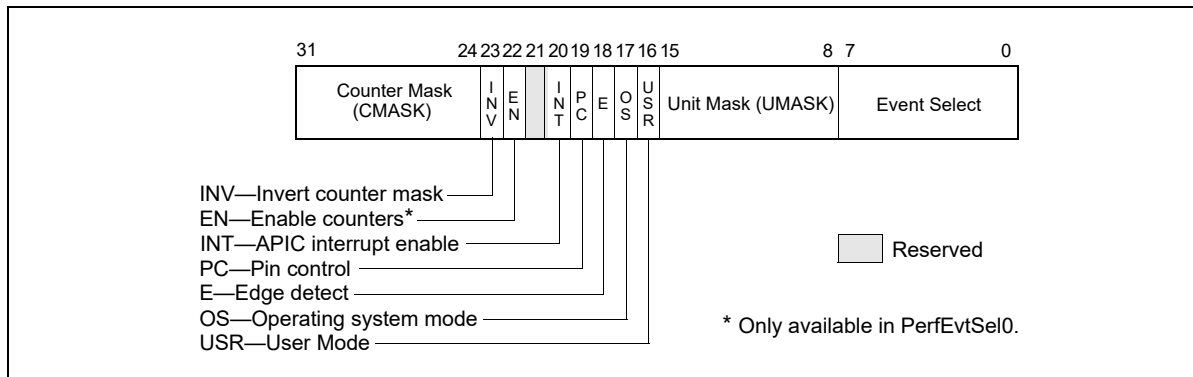


Figure 19-63. PerfEvtSel0 and PerfEvtSel1 MSRs

- **USR (user mode) flag (bit 16)** — Specifies that events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of events. The processor counts the number of deasserted to asserted transitions of any condition that can be expressed by the other fields. The mechanism is limited in that it does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

- **PC (pin control) flag (bit 19)** — When set, the processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.
- **INT (APIC interrupt enable) flag (bit 20)** — When set, the processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — This flag is only present in the PerfEvtSel0 MSR. When set, performance counting is enabled in both performance-monitoring counters; when clear, both counters are disabled.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When nonzero, the processor compares this mask to the number of events counted during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. This mask can be used to count events only if multiple occurrences happen per clock (for example, two or more instructions retired per clock). If the counter-mask field is 0, then the counter is incremented each cycle by the number of events that occurred that cycle.

19.6.8.2 PerfCtr0 and PerfCtr1 MSRs

The performance-counter MSRs (PerfCtr0 and PerfCtr1) contain the event or duration counts for the selected events being counted. The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

The WRMSR instruction cannot arbitrarily write to the performance-monitoring counter MSRs (PerfCtr0 and PerfCtr1). Instead, the lower-order 32 bits of each MSR may be written with any value, and the high-order 8 bits are sign-extended according to the value of bit 31. This operation allows writing both positive and negative values to the performance counters.

19.6.8.3 Starting and Stopping the Performance-Monitoring Counters

The performance-monitoring counters are started by writing valid setup information in the PerfEvtSel0 and/or PerfEvtSel1 MSRs and setting the enable counters flag in the PerfEvtSel0 MSR. If the setup is valid, the counters begin counting following the execution of a WRMSR instruction that sets the enable counter flag. The counters can be stopped by clearing the enable counters flag or by clearing all the bits in the PerfEvtSel0 and PerfEvtSel1 MSRs. Counter 1 alone can be stopped by clearing the PerfEvtSel1 MSR.

19.6.8.4 Event and Time-Stamp Monitoring Software

To use the performance-monitoring counters and time-stamp counter, the operating system needs to provide an event-monitoring device driver. This driver should include procedures for handling the following operations:

- Feature checking.
- Initialize and start counters.
- Stop counters.
- Read the event counters.
- Read the time-stamp counter.

The event monitor feature determination procedure must check whether the current processor supports the performance-monitoring counters and time-stamp counter. This procedure compares the family and model of the processor returned by the CPUID instruction with those of processors known to support performance monitoring. (The Pentium and P6 family processors support performance counters.) The procedure also checks the MSR and TSC flags returned to register EDX by the CPUID instruction to determine if the MSRs and the RDTSC instruction are supported.

The initialize and start counters procedure sets the PerfEvtSel0 and/or PerfEvtSel1 MSRs for the events to be counted and the method used to count them and initializes the counter MSRs (PerfCtr0 and PerfCtr1) to starting counts. The stop counters procedure stops the performance counters (see Section 19.6.8.3, “Starting and Stopping the Performance-Monitoring Counters”).

The read counters procedure reads the values in the PerfCtr0 and PerfCtr1 MSRs, and a read time-stamp counter procedure reads the time-stamp counter. These procedures would be provided in lieu of enabling the RDTSC and RDPMC instructions that allow application code to read the counters.

19.6.8.5 Monitoring Counter Overflow

The P6 family processors provide the option of generating a local APIC interrupt when a performance-monitoring counter overflows. This mechanism is enabled by setting the interrupt enable flag in either the PerfEvtSel0 or the PerfEvtSel1 MSR. The primary use of this option is for statistical performance sampling.

To use this option, the operating system should do the following things on the processor for which performance events are required to be monitored:

- Provide an interrupt vector for handling the counter-overflow interrupt.
- Initialize the APIC PERF local vector entry to enable handling of performance-monitor counter overflow events.
- Provide an entry in the IDT that points to a stub exception handler that returns without executing any instructions.
- Provide an event monitor driver that provides the actual interrupt handler and modifies the reserved IDT entry to point to its interrupt routine.

When interrupted by a counter overflow, the interrupt handler needs to perform the following actions:

- Save the instruction pointer (EIP register), code-segment selector, TSS segment selector, counter values and other relevant information at the time of the interrupt.
- Reset the counter to its initial setting and return from the interrupt.

An event monitor application utility or another application program can read the information collected for analysis of the performance of the profiled application.

19.6.9 Performance Monitoring (Pentium Processors)

The Pentium processor provides two 40-bit performance counters, which can be used to count events or measure duration. The counters are supported by three MSRs: the control and event select MSR (CESR) and the performance counter MSRs (CTR0 and CTR1). These can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0.

Each counter has an associated external pin (PM0/BP0 and PM1/BP1), which can be used to indicate the state of the counter to external hardware.

NOTES

The CESR, CTR0, and CTR1 MSRs and the events listed for Pentium processors are model-specific for the Pentium processor.

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

19.6.9.1 Control and Event Select Register (CESR)

The 32-bit control and event select MSR (CESR) controls the operation of performance-monitoring counters CTR0 and CTR1 and the associated pins (see Figure 19-64). To control each counter, the CESR register contains a 6-bit event select field (ES0 and ES1), a pin control flag (PC0 and PC1), and a 3-bit counter control field (CC0 and CC1). The functions of these fields are as follows:

- **ES0 and ES1 (event select) fields (bits 0-5, bits 16-21)** — Selects (by entering an event code in the field) up to two events to be monitored.

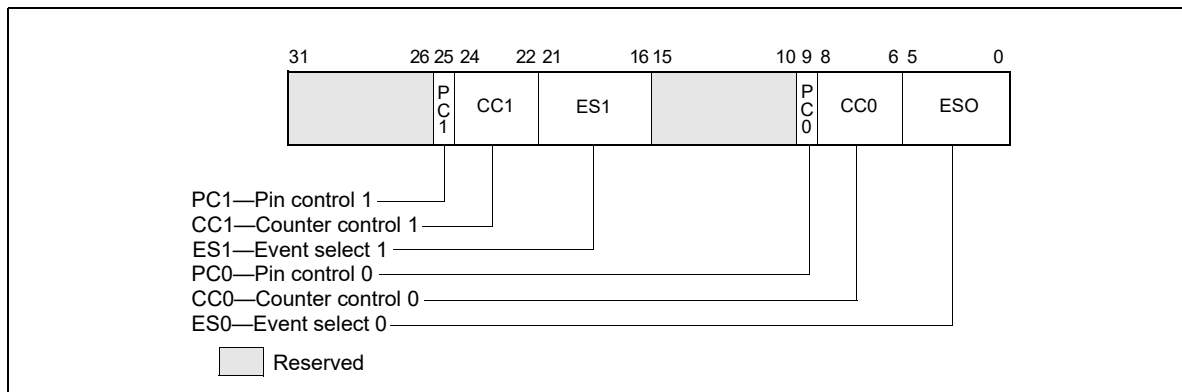


Figure 19-64. CESR MSR (Pentium Processor Only)

- **CC0 and CC1 (counter control) fields (bits 6-8, bits 22-24)** — Controls the operation of the counter. Control codes are as follows:

- 000 — Count nothing (counter disabled).
- 001 — Count the selected event while CPL is 0, 1, or 2.
- 010 — Count the selected event while CPL is 3.
- 011 — Count the selected event regardless of CPL.
- 100 — Count nothing (counter disabled).
- 101 — Count clocks (duration) while CPL is 0, 1, or 2.
- 110 — Count clocks (duration) while CPL is 3.
- 111 — Count clocks (duration) regardless of CPL.

The highest order bit selects between counting events and counting clocks (duration); the middle bit enables counting when the CPL is 3; and the low-order bit enables counting when the CPL is 0, 1, or 2.

- **PC0 and PC1 (pin control) flags (bits 9, 25)** — Selects the function of the external performance-monitoring counter pin (PM0/BP0 and PM1/BP1). Setting one of these flags to 1 causes the processor to assert its associated pin when the counter has overflowed; setting the flag to 0 causes the pin to be asserted when the counter has been incremented. These flags permit the pins to be individually programmed to indicate the overflow or incremented condition. The external signaling of the event on the pins will lag the internal event by a few clocks as the signals are latched and buffered.

While a counter need not be stopped to sample its contents, it must be stopped and cleared or preset before switching to a new event. It is not possible to set one counter separately. If only one event needs to be changed, the CESR register must be read, the appropriate bits modified, and all bits must then be written back to CESR. At reset, all bits in the CESR register are cleared.

19.6.9.2 Use of the Performance-Monitoring Pins

When performance-monitor pins PM0/BP0 and/or PM1/BP1 are configured to indicate when the performance-monitor counter has incremented and an "occurrence event" is being counted, the associated pin is asserted (high) each time the event occurs. When a "duration event" is being counted, the associated PM pin is asserted for the

entire duration of the event. When the performance-monitor pins are configured to indicate when the counter has overflowed, the associated PM pin is asserted when the counter has overflowed.

When the PM0/BP0 and/or PM1/BP1 pins are configured to signal that a counter has incremented, it should be noted that although the counters may increment by 1 or 2 in a single clock, the pins can only indicate that the event occurred. Moreover, since the internal clock frequency may be higher than the external clock frequency, a single external clock may correspond to multiple internal clocks.

A “count up to” function may be provided when the event pin is programmed to signal an overflow of the counter. Because the counters are 40 bits, a carry out of bit 39 indicates an overflow. A counter may be preset to a specific value less than $2^{40} - 1$. After the counter has been enabled and the prescribed number of events has transpired, the counter will overflow.

Approximately 5 clocks later, the overflow is indicated externally and appropriate action, such as signaling an interrupt, may then be taken.

The PM0/BP0 and PM1/BP1 pins also serve to indicate breakpoint matches during in-circuit emulation, during which time the counter increment or overflow function of these pins is not available. After RESET, the PM0/BP0 and PM1/BP1 pins are configured for performance monitoring, however a hardware debugger may reconfigure these pins to indicate breakpoint matches.

19.6.9.3 Events Counted

Events that performance-monitoring counters can be set to count and record (using CTR0 and CTR1) are divided in two categories: occurrence and duration:

- **Occurrence events** — Counts are incremented each time an event takes place. If PM0/BP0 or PM1/BP1 pins are used to indicate when a counter increments, the pins are asserted each clock counters increment. But if an event happens twice in one clock, the counter increments by 2 (the pins are asserted only once).
- **Duration events** — Counters increment the total number of clocks that the condition is true. When used to indicate when counters increment, PM0/BP0 and/or PM1/BP1 pins are asserted for the duration.

19.7 COUNTING CLOCKS

The count of cycles, also known as clockticks, forms the basis for measuring how long a program takes to execute. Clockticks are also used as part of efficiency ratios like cycles per instruction (CPI). Processor clocks may stop ticking under circumstances like the following:

- The processor is halted when there is nothing for the CPU to do. For example, the processor may halt to save power while the computer is servicing an I/O request. When Intel Hyper-Threading Technology is enabled, both logical processors must be halted for performance-monitoring counters to be powered down.
- The processor is asleep as a result of being halted or because of a power-management scheme. There are different levels of sleep. In the some deep sleep levels, the time-stamp counter stops counting.

In addition, processor core clocks may undergo transitions at different ratios relative to the processor’s bus clock frequency. Some of the situations that can cause processor core clock to undergo frequency transitions include:

- TM2 transitions.
- Enhanced Intel SpeedStep Technology transitions (P-state transitions).

For Intel processors that support TM2, the processor core clocks may operate at a frequency that differs from the Processor Base frequency (as indicated by processor frequency information reported by CPUID instruction). See Section 19.7.2 for more detail.

Due to the above considerations there are several important clocks referenced in this manual:

- **Base Clock** — The frequency of this clock is the frequency of the processor when the processor is not in turbo mode, and not being throttled via Intel SpeedStep.
- **Maximum Clock** — This is the maximum frequency of the processor when turbo mode is at the highest point.
- **Bus Clock** — These clockticks increment at a fixed frequency and help coordinate the bus on some systems.

- **Core Crystal Clock** — This is a clock that runs at fixed frequency; it coordinates the clocks on all packages across the system.
- **Non-halted Clockticks** — Measures clock cycles in which the specified logical processor is not halted and is not in any power-saving state. When Intel Hyper-Threading Technology is enabled, ticks can be measured on a per-logical-processor basis. There are also performance events on dual-core processors that measure clockticks per logical processor when the processor is not halted.
- **Non-sleep Clockticks** — Measures clock cycles in which the specified physical processor is not in a sleep mode or in a power-saving state. These ticks cannot be measured on a logical-processor basis.
- **Time-stamp Counter** — See Section 17.17, “Time-Stamp Counter”.
- **Reference Clockticks** — TM2 or Enhanced Intel SpeedStep technology are two examples of processor features that can cause processor core clockticks to represent non-uniform tick intervals due to change of bus ratios. Performance events that counts clockticks of a constant reference frequency was introduced Intel Core Duo and Intel Core Solo processors. The mechanism is further enhanced on processors based on Intel Core microarchitecture.

Some processor models permit clock cycles to be measured when the physical processor is not in deep sleep (by using the time-stamp counter and the RDTSC instruction). Note that such ticks cannot be measured on a per-logical-processor basis. See Section 17.17, “Time-Stamp Counter,” for detail on processor capabilities.

The first two methods use performance counters and can be set up to cause an interrupt upon overflow (for sampling). They may also be useful where it is easier for a tool to read a performance counter than to use a time stamp counter (the timestamp counter is accessed using the RDTSC instruction).

For applications with a significant amount of I/O, there are two ratios of interest:

- **Non-halted CPI** — Non-halted clockticks/instructions retired measures the CPI for phases where the CPU was being used. This ratio can be measured on a logical-processor basis when Intel Hyper-Threading Technology is enabled.
- **Nominal CPI** — Time-stamp counter ticks/instructions retired measures the CPI over the duration of a program, including those periods when the machine halts while waiting for I/O.

19.7.1 Non-Halted Reference Clockticks

Software can use UnHalted Reference Cycles on either a general purpose performance counter using event mask 0x3C and UMASK 0x01 or on fixed function performance counter 2 to count at a constant rate. These events count at a consistent rate irrespective of P-state, TM2, or frequency transitions that may occur to the processor. The UnHalted Reference Cycles event may count differently on the general purpose event and fixed counter.

19.7.2 Cycle Counting and Opportunistic Processor Operation

As a result of the state transitions due to opportunistic processor performance operation (see Chapter 14, “Power and Thermal Management”), a logical processor or a processor core can operate at frequency different from the Processor Base frequency.

The following items are expected to hold true irrespective of when opportunistic processor operation causes state transitions:

- The time stamp counter operates at a fixed-rate frequency of the processor.
- The IA32_MPERF counter increments at a fixed frequency irrespective of any transitions caused by opportunistic processor operation.
- The IA32_FIXED_CTR2 counter increments at the same TSC frequency irrespective of any transitions caused by opportunistic processor operation.
- The Local APIC timer operation is unaffected by opportunistic processor operation.
- The TSC, IA32_MPERF, and IA32_FIXED_CTR2 operate at close to the maximum non-turbo frequency, which is equal to the product of scalable bus frequency and maximum non-turbo ratio.

19.7.3 Determining the Processor Base Frequency

For Intel processors in which the nominal core crystal clock frequency is enumerated in CPUID.15H.ECX and the core crystal clock ratio is encoded in CPUID.15H (see Table 3-8 “Information Returned by CPUID Instruction”), the nominal TSC frequency can be determined by using the following equation:

$$\text{Nominal TSC frequency} = (\text{CPUID.15H.ECX}[31:0] * \text{CPUID.15H.EBX}[31:0]) \div \text{CPUID.15H.EAX}[31:0]$$

For Intel processors in which CPUID.15H.EBX[31:0] ÷ CPUID.0x15.EAX[31:0] is enumerated but CPUID.15H.ECX is not enumerated, Table 19-91 can be used to look up the nominal core crystal clock frequency.

Table 19-91. Nominal Core Crystal Clock Frequency

Processor Families/Processor Number Series ¹	Nominal Core Crystal Clock Frequency
Intel® Xeon® Processor Scalable Family with CPUID signature 06_55H.	25 MHz
6th and 7th generation Intel® Core™ processors and Intel® Xeon® W Processor Family.	24 MHz
Next Generation Intel Atom® processors based on Goldmont Microarchitecture with CPUID signature 06_5CH (does not include Intel Xeon processors).	19.2 MHz

NOTES:

1. For any processor in which CPUID.15H is enumerated and MSR_PLATFORM_INFO[15:8] (which gives the scalable bus frequency) is available, a more accurate frequency can be obtained by using CPUID.15H.

19.7.3.1 For Intel® Processors Based on Sandy Bridge, Ivy Bridge, Haswell and Broadwell Microarchitectures

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 100 MHz.

19.7.3.2 For Intel® Processors Based on Nehalem Microarchitecture

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 133.33 MHz.

19.7.3.3 For Intel Atom® Processors Based on Silvermont Microarchitecture (Including Intel Processors Based on Airmont Microarchitecture)

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by the scalable bus frequency. The scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] for Intel Atom processors based on the Silvermont microarchitecture, and in bit field MSR_FSB_FREQ[3:0] for processors based on the Airmont microarchitecture; see Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*.

19.7.3.4 For Intel® Core™ 2 Processor Family and for Intel® Xeon® Processors Based on Intel Core Microarchitecture

For processors based on Intel Core microarchitecture, the scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] at (0CDH), see Chapter 2, “Model-Specific Registers (MSRs)” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4*. The maximum resolved bus ratio can be read from the following bit field:

- If XE operation is disabled, the maximum resolved bus ratio can be read in MSR_PLATFORM_ID[12:8]. It corresponds to the Processor Base frequency.

- If XE operation is enabled, the maximum resolved bus ratio is given in MSR_PERF_STATUS[44:40], it corresponds to the maximum XE operation frequency configured by BIOS.

XE operation of an Intel 64 processor is implementation specific. XE operation can be enabled only by BIOS. If MSR_PERF_STATUS[31] is set, XE operation is enabled. The MSR_PERF_STATUS[31] field is read-only.

19.8 IA32_PERF_CAPABILITIES MSR ENUMERATION

The layout of IA32_PERF_CAPABILITIES MSR is shown in Figure 19-65; it provides enumeration of a variety of interfaces:

- IA32_PERF_CAPABILITIES.LBR_FMT[bits 5:0]: encodes the LBR format, details are described in Section 17.4.8.1.
- IA32_PERF_CAPABILITIES.PEBSTrap[6]: Trap/Fault-like indicator of PEBS recording assist; see Section 19.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBSArchRegs[7]: Indicator of PEBS assist save architectural registers; see Section 19.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBS_FMT[bits 11:8]: Specifies the encoding of the layout of PEBS records; see Section 19.6.2.4.2.
- IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[12]: Indicates IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if 1, see Section 19.8.1.
- IA32_PERF_CAPABILITIES.FULL_WRITE[13]: Indicates the processor supports IA32_A_PMCx interface for updating bits 32 and above of IA32_PMCx; see Section 19.2.6.
- IA32_PERF_CAPABILITIES.PEBS_BASELINE [bit 14]: If set, the following is true:
 - The IA32_PEBS_ENABLE MSR (address 3F1H) exists and all architecturally enumerated fixed and general-purpose counters have corresponding bits in IA32_PEBS_ENABLE that enable generation of PEBS records. The general-purpose counter bits start at bit IA32_PEBS_ENABLE[0], and the fixed counter bits start at bit IA32_PEBS_ENABLE[32].
 - The format of the PEBS record is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT; see Section 19.6.2.4.2.
 - Extended PEBS is supported. All counters support the PEBS facility, and all events (both precise and non-precise) can generate PEBS records when PEBS is enabled for that counter. Note that not all events may be available on all counters.
 - Adaptive PEBS is supported. The PEBS_DATA_CFG MSR (address 3F2H) and adaptive record enable bits (IA32_PERFEVTSELx.Adaptive_Record and IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record) are supported. The definition of the PEBS_DATA_CFG MSR, including which bits are supported and how they affect the record, is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT; see Section 19.9.2.3.
 - NOTE: Software is recommended to feature PEBS Baseline when the following is true:
IA32_PERF_CAPABILITIES.PEBS_BASELINE[14] && IA32_PERF_CAPABILITIES.PEBS_FMT[11:8] ≥ 4.
- IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE[15]: If set, indicates that the architecture provides built in support for TMA L1 metrics through the PERF_METRICS MSR, see Section 19.3.9.3.
- IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16]: If set on parts that enumerate support for Intel PT (CPUID.0x7.0.EBX[25]=1), setting IA32_PEBS_ENABLE.PEBS_OUTPUT to 01B will result in PEBS output being written into the Intel PT trace stream. See Section 19.5.5.2.

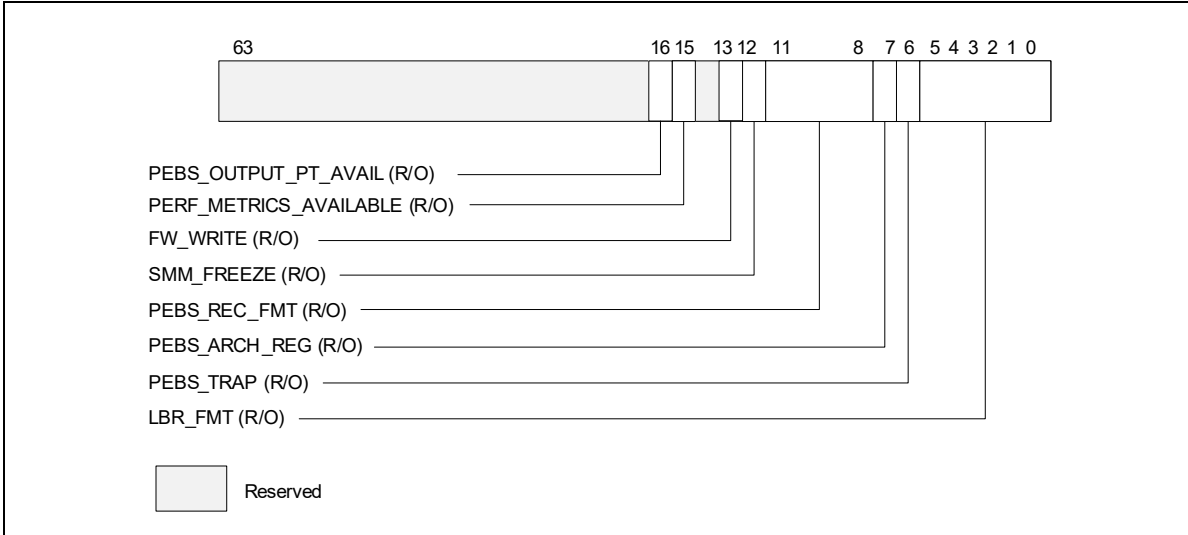


Figure 19-65. Layout of IA32_PERF_CAPABILITIES MSR

19.8.1 Filtering of SMM Handler Overhead

When performance monitoring facilities and/or branch profiling facilities (see Section 17.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)”) are enabled, these facilities capture event counts, branch records and branch trace messages occurring in a logical processor. The occurrence of interrupts, instruction streams due to various interrupt handlers all contribute to the results recorded by these facilities.

If CPUID.01H:ECX.PDCM[bit 15] is 1, the processor supports the IA32_PERF_CAPABILITIES MSR. If IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is 1, the processor supports the ability for system software using performance monitoring and/or branch profiling facilities to filter out the effects of servicing system management interrupts.

If the FREEZE_WHILE_SMM capability is enabled on a logical processor and after an SMI is delivered, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler.

The enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its servicing.

It is the responsibility of the SMM code to ensure the state of the performance monitoring and branch profiling facilities are preserved upon entry or until prior to exiting the SMM. If any of this state is modified due to actions by the SMM code, the SMM code is required to restore such state to the values present at entry to the SMM handler.

System software is allowed to set IA32_DEBUGCTL.FREEZE_WHILE_SMM[bit 14] to 1 only supported as indicated by IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] reporting 1.

19.9 PEBS FACILITY

19.9.1 Extended PEBS

- The Extended PEBS feature supports Processor Event Based Sampling (PEBS) on all counters, both fixed function and general purpose; and all performance monitoring events, both precise and non-precise. PEBS can be enabled for the general purpose counters using PEBS_EN_PMCi bits of IA32_PEBS_ENABLE (i = 0, 1,..m). PEBS can be enabled for 'i' fixed function counters using the PEBS_EN_FIXEDi bits of IA32_PEBS_ENABLE (i = 0, 1, ...n).

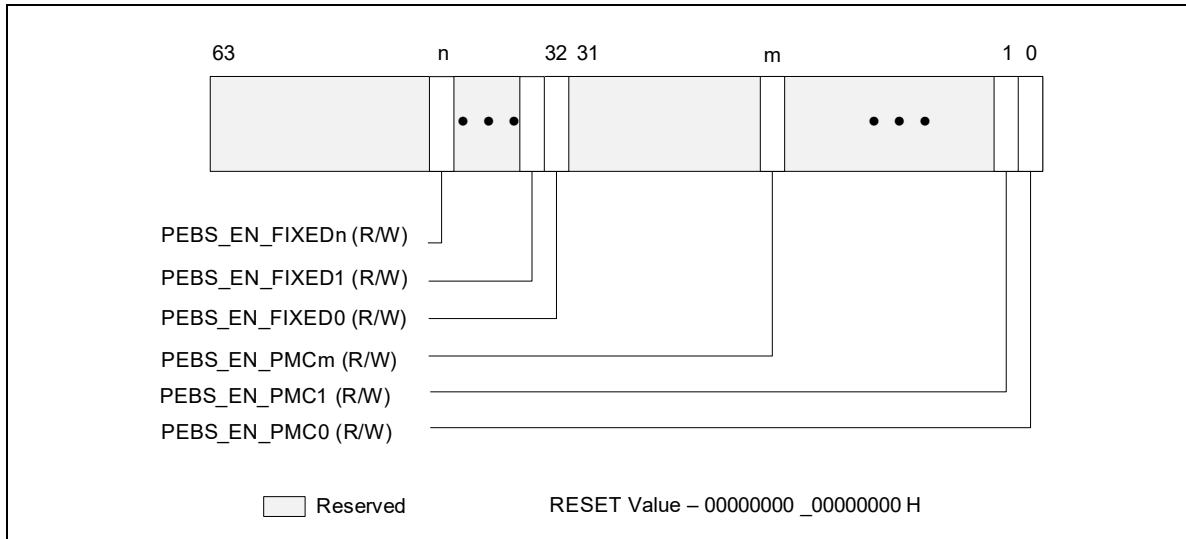


Figure 19-66. Layout of IA32_PEBS_ENABLE MSR

A PEBS record due to a precise event will be generated after an instruction that causes the event when the counter has already overflowed. A PEBS record due to a non-precise event will occur at the next opportunity after the counter has overflowed, including immediately after an overflow is set by an MSR write.

Currently, IA32_FIXED_CTR0 counts instructions retired and is a precise event. IA32_FIXED_CTR1, IA32_FIXED_CTR2 ... IA32_FIXED_CTR m count as non-precise events.

The Applicable Counter field in the Basic Info Group of the PEBS record indicates which counters caused the PEBS record to be generated. It is in the same format as the enable bits for each counter in IA32_PEBS_ENABLE. As an example, an Applicable Counter field with bits 2 and 32 set would indicate that both general purpose counter 2 and fixed function counter 0 generated the PEBS record.

- To properly use PEBS for the additional counters, software will need to set up the counter reset values in PEBS portion of the DS_BUFFER_MANAGEMENT_AREA data structure that is indicated by the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 19-67. When a counter generates a PEBS records, the appropriate counter reset values will be loaded into that counter. In the above example where general purpose counter 2 and fixed function counter 0 generated the PEBS record, general purpose counter 2 would be reloaded with the value contained in PEBS GP Counter 2 Reset (offset 50H) and fixed function counter 0 would be reloaded with the value contained in PEBS Fixed Counter 0 Reset (offset 80H).

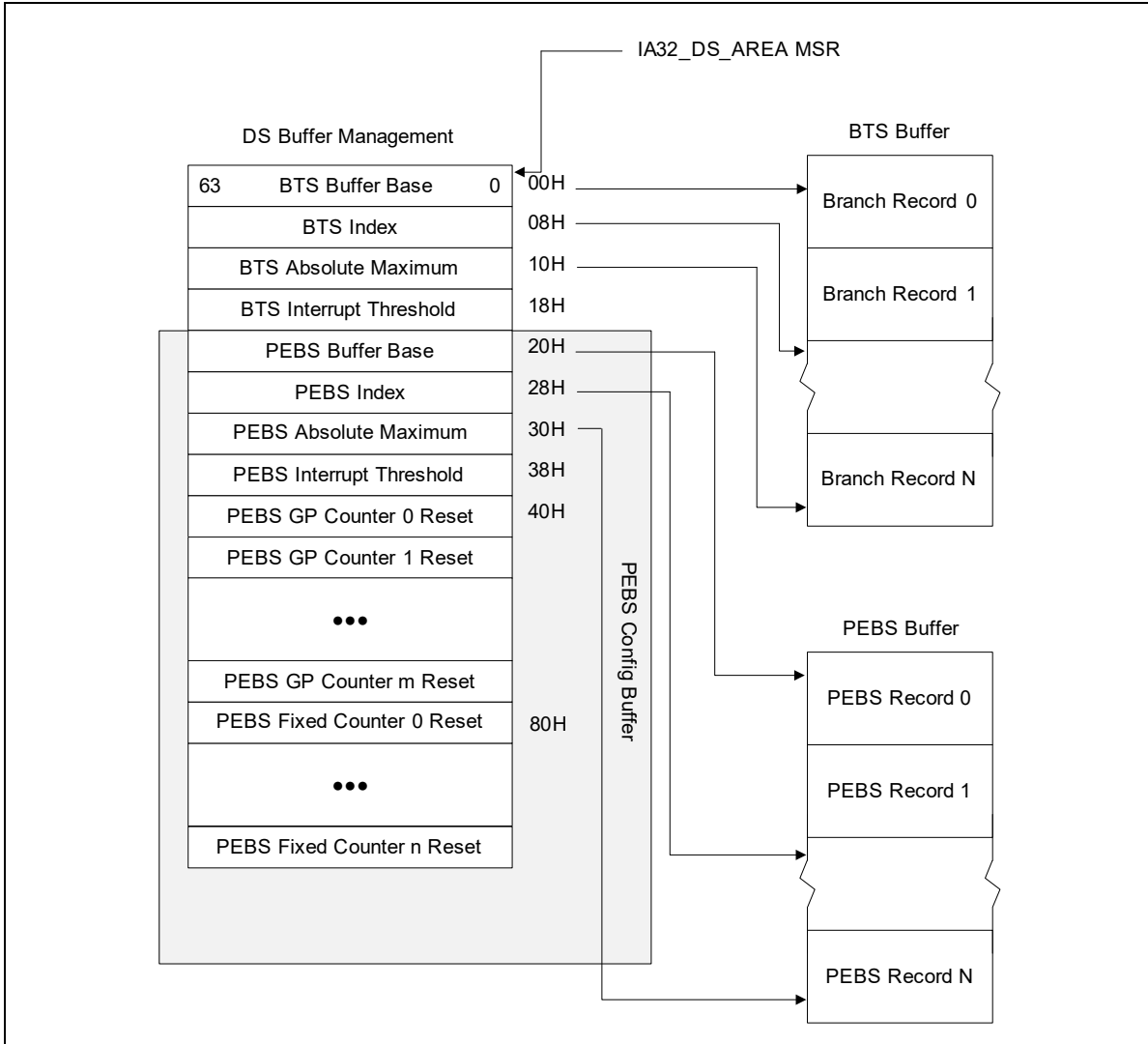


Figure 19-67. PEBS Programming Environment

Extended PEBS support debuts on Intel Atom[®] processors based on the Goldmont Plus microarchitecture and future Intel[®] Core[™] processors based on the Ice Lake microarchitecture.

19.9.2 Adaptive PEBS

The PEBS facility has been enhanced to collect the following CPU state in addition to GPRs, EventingIP, TSC and memory access related information collected by legacy PEBS:

- XMM registers
- LBR records (TO/FROM/INFO)

The PEBS record is restructured where fields are grouped into Basic group, Memory group, GPR group, XMM group and LBR group. A new register MSR_PEBS_DATA_CFG provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.

By default, the PEBS record will only contain the Basic group. Optionally, each counter can be configured to generate a PEBS records with the groups specified in MSR_PEBS_DATA_CFG.

Details and examples for the Adaptive PEBS capability follow below.

19.9.2.1 Adaptive_Record Counter Control

- IA32_PERFEVTSELx.Adaptive_Record[34]: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_PMCx is set for the corresponding GP counter, an overflow of PMCx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

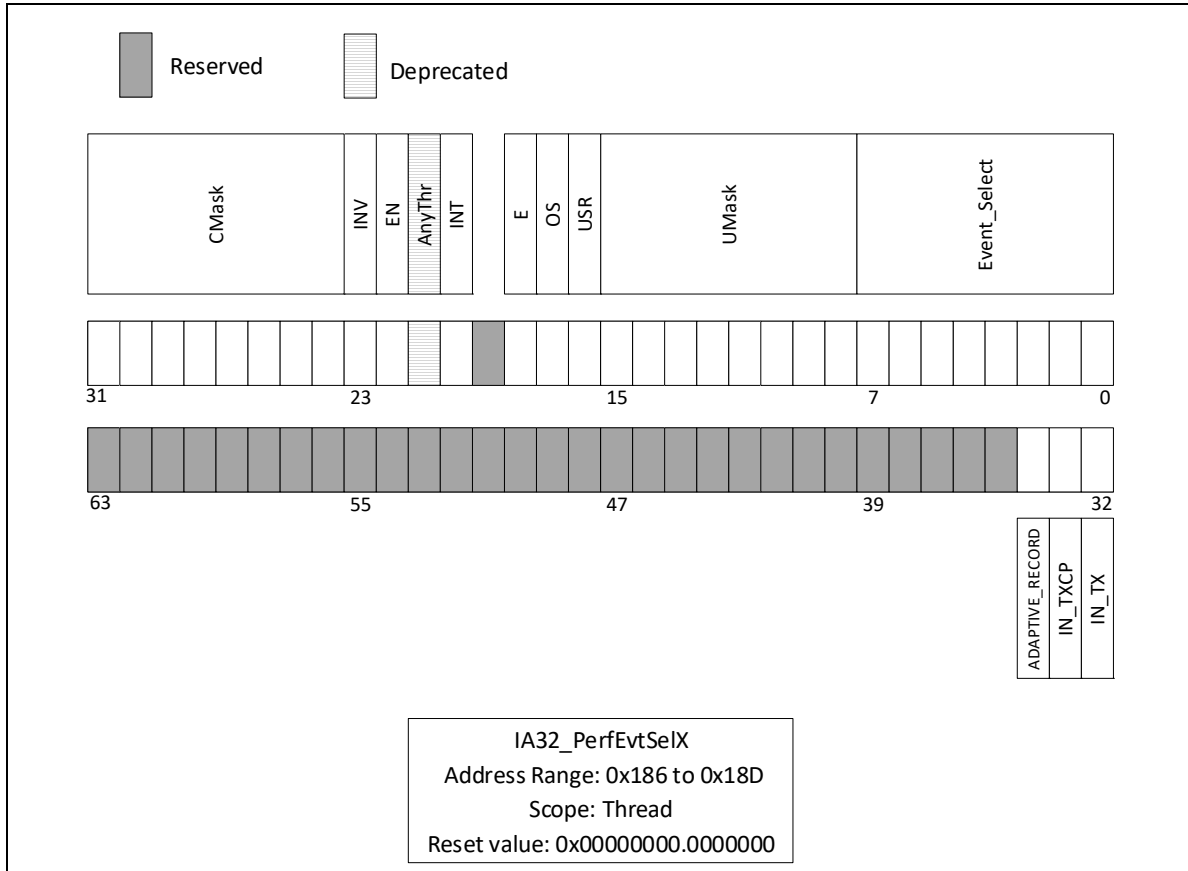


Figure 19-68. Layout of IA32_PerfEvtSelX MSR Supporting Adaptive PEBS

- IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_FIXEDx is set for the corresponding Fixed counter, an overflow of FixedCtrx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

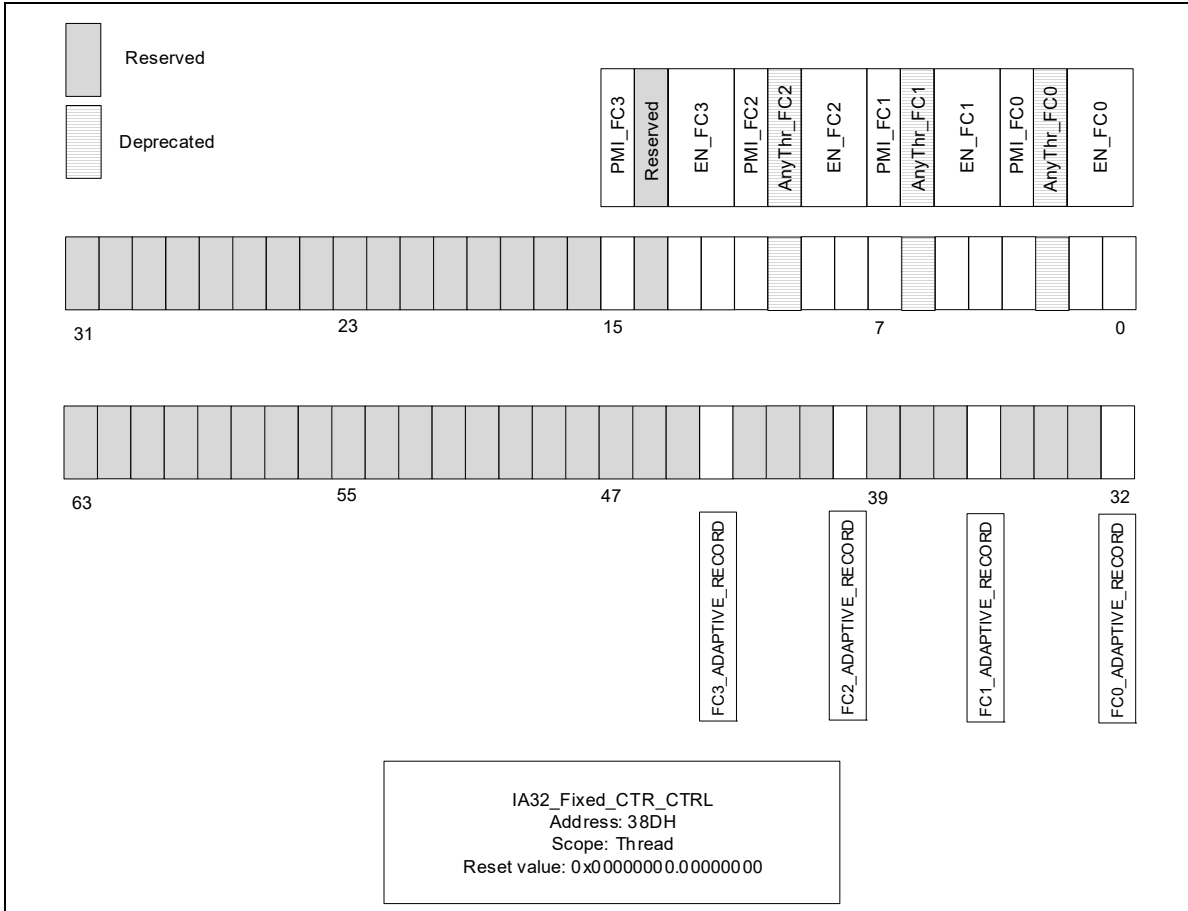


Figure 19-69. Layout of IA32_FIXED_CTR_CTRL MSR Supporting Adaptive PEBS

19.9.2.2 PEBS Record Format

The data fields in the PEBS record are aggregated into five groups which are described in the sub-sections below. Processors that support Adaptive PEBS implement a new MSR called MSR_PEBS_DATA_CFG which allows software to select the data groups to be captured. The data groups are not placed at fixed locations in the PEBS record, but are positioned immediately after one another, thus making the record format/size variable based on the groups selected.

19.9.2.2.1 Basic Info

The Basic group contains essential information for software to parse a record along with several critical fields. It is always collected.

Table 19-92. Basic Info Group

Field Name	Bit Width	Description
Record Format	[47:0]	This field indicates which data groups are included in the record. The field is zero if none of the counters that triggered the current PEBS record have their Adaptive_Record bit set. Otherwise it contains the value of MSR_PEBS_DATA_CFG.
	[63:48]	This field provides the size of the current record in bytes. Selected groups are packed back-to-back in the record without gaps or padding for unselected groups.

Table 19-92. Basic Info Group (Contd.)

Instruction Pointer	[63:0]	This field reports the Eventing Instruction Pointer (EventingIP) of the retired instruction that triggered the PEBS record generation. Note that this field is different than R/EIP which records the instruction pointer of the next instruction to be executed after record generation. The legacy R/EIP field has been removed.
Applicable Counters	[63:0]	The Applicable Counters field indicates which counters triggered the generation of the PEBS record, linking the record to specific events. This allows software to correlate the PEBS record entry properly with the instruction that caused the event, even when multiple counters are configured to generate PEBS records and multiple bits are set in the field.
TSC	[63:0]	This field provides the time stamp counter value when the PEBS record was generated.

19.9.2.2.2 Memory Access Info

This group contains the legacy PEBS memory-related fields; see Section 19.3.1.1.2.

Table 19-93. Memory Access Info Group

Field Name	Bit Width	Description
Memory Access Address	[63:0]	This field contains the linear address of the source of the load, or linear address of the destination (target) of the store. This value is written as a 64-bit address in canonical form.
Memory Auxiliary Info	[63:0]	When MEM_TRANS_RETIRED.* event is configured in a General Purpose counter, this field contains an encoded value indicating the memory hierarchy source which satisfied the load. These encodings are detailed in Table 19-4 and Table 19-13. If the PEBS assist was triggered for a store uop, this field will contain information indicating the status of the store, as detailed in Table 19-14.
Memory Access Latency	[63:0]	When MEM_TRANS_RETIRED.* event is configured in a General Purpose counter, this field contains the latency to service the load in core clock cycles.
TSX Auxiliary Info	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	[63:32]	This field contains the abort details. Refer to Section 19.3.6.5.1.

Beginning with 12th generation Intel Core processors, the memory access information group has been updated. New fields added are shaded gray in Table 19-94.

Table 19-94. Updated Memory Access Info Group

Field Name	Sub-field Name	Bits	Description
Access Address (offset 0H)	DLA	[63:0]	This field reports the data linear address (DLA) of the memory access in canonical form. A zero value indicates the processor could not retrieve the address of the particular access.
Access Info (offset 8H)	Data Src	[3:0]	An encoded value indicating the memory hierarchy source which satisfied the access. These encodings are detailed in Table 19-4. A zero value indicates the processor could not retrieve the data source of the particular access.
	STLB-miss	[4]	A value of 1 indicates the access has missed the Second-level TLB (STLB).

Table 19-94. Updated Memory Access Info Group (Contd.)

Field Name	Sub-field Name	Bits	Description
	Is-Lock	[5]	A value of 1 indicates the access was part of a locked (atomic) memory transaction.
	Data-Blk	[6]	A value of 1 indicates the load was blocked since its data could not be forwarded from a preceding store.
	Address-Blk	[7]	A value of 1 indicates the load was blocked due to potential address conflict with a preceding store.
Access Latency (offset 10H)	Instruction Latency	[15:0]	Measured instruction latency in core cycles. For loads, the latency starts by the dispatch of the load operation for execution and lasts until completion of the instruction it belongs to. This field includes the entire latency including time for data-dependency resolution or TLB lookups.
	Cache Latency	[47:32]	Measured cache access latency in core cycles. For loads, the latency starts by the actual cache access until the data is returned by the memory subsystem. For stores, the latency starts when the demand write accesses the L1 data-cache and lasts until the cacheline write is completed in the memory subsystem. This field does not include non-data-cache latency such as memory ordering checks or TLB lookups.
TSX (offset 18H)	Transaction Latency	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	Abort Info	[63:32]	This field contains the abort details. Refer to Section 19.3.6.5.1.

To determine which fields are supported for certain performance monitoring events, consult the Memory Info attribute in the event lists at <https://download.01.org/perfmon/>.

NOTE

There may be additional block reasons, even if Data-Blk and Address-Blk are both clear, e.g., non-optimal instruction latency.

On P-core, the new Data-Blk and Address-Blk bits require the event LD_BLOCKS.STORE_FORWARD (r8203) to be configured in a programmable counter.

19.9.2.2.3 GPRs

This group is captured when the GPR bit is enabled in MSR_PEBS_DATA_CFG. GPRs are always 64 bits wide. If they are selected for non 64-bit mode, the upper 32-bit of the legacy RAX - RDI and all contents of R8-15 GPRs will be filled with 0s. In 64bit mode, the full 64 bit value of each register is written.

The order differs from legacy. The table below shows the order of the GPRs in Ice Lake microarchitecture.

Table 19-95. GPRs in Ice Lake Microarchitecture

Field Name	Bit Width
RFLAGS	[63:0]
RIP	[63:0]
RAX	[63:0]
RCX*	[63:0]
RDX*	[63:0]

Table 19-95. GPRs in Ice Lake Microarchitecture (Contd.)

RBX*	[63:0]
RSP*	[63:0]
RBP*	[63:0]
RSI*	[63:0]
RDI*	[63:0]
R8	[63:0]
...	...
R15	[63:0]

The machine state reported in the PEBS record is the committed machine state immediately after the instruction that triggers PEBS completes.

For instance, consider the following instruction sequence:

MOV eax, [eax]; triggers PEBS record generation

NOP

If the mov instruction triggers PEBS record generation, the EventingIP field in the PEBS record will report the address of the mov, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation. And the value of RIP will contain the linear address of the nop.

19.9.2.2.4 XMMs

This group is captured when the XMM bit is enabled in MSR_PEBS_DATA_CFG and SSE is enabled. If SSE is not enabled, the fields will contain zeroes. XMM8-XMM15 will also contain zeroes if not in 64-bit mode.

Table 19-96. XMMs

Field Name	Bit Width
XMM0	[127:0]
...	...
XMM15	[127:0]

19.9.2.2.5 LBRs

To capture LBR data in the PEBS record, the LBR bit in MSR_PEBS_DATA_CFG must be enabled. The number of LBR entries included in the record can be configured in the LBR_entries field of MSR_PEBS_DATA_CFG.

Table 19-97. LBRs

Field Name	Bit Width	Description
LBR[.].FROM	[63:0]	Branch from address.
LBR[.].TO	[63:0]	Branch to address.
LBR[.].INFO	[63:0]	Other LBR information, like timing. This field is described in more detail in Section 17.12.1, "MSR_LBR_INFO_x MSR".

LBR entries are recorded into the record starting at LBR[TOS] and proceeding to LBR[TOS-1] and following. Note that LBR index is modulo the number of LBRs supporting on the processor.

19.9.2.3 MSR_PEBS_DATA_CFG

Bits in MSR_PEBS_DATA_CFG can be set to include data field blocks/groups into adaptive records. The Basic Info group is always included in the record. Additionally, the number of LBR entries included in the record is configurable.

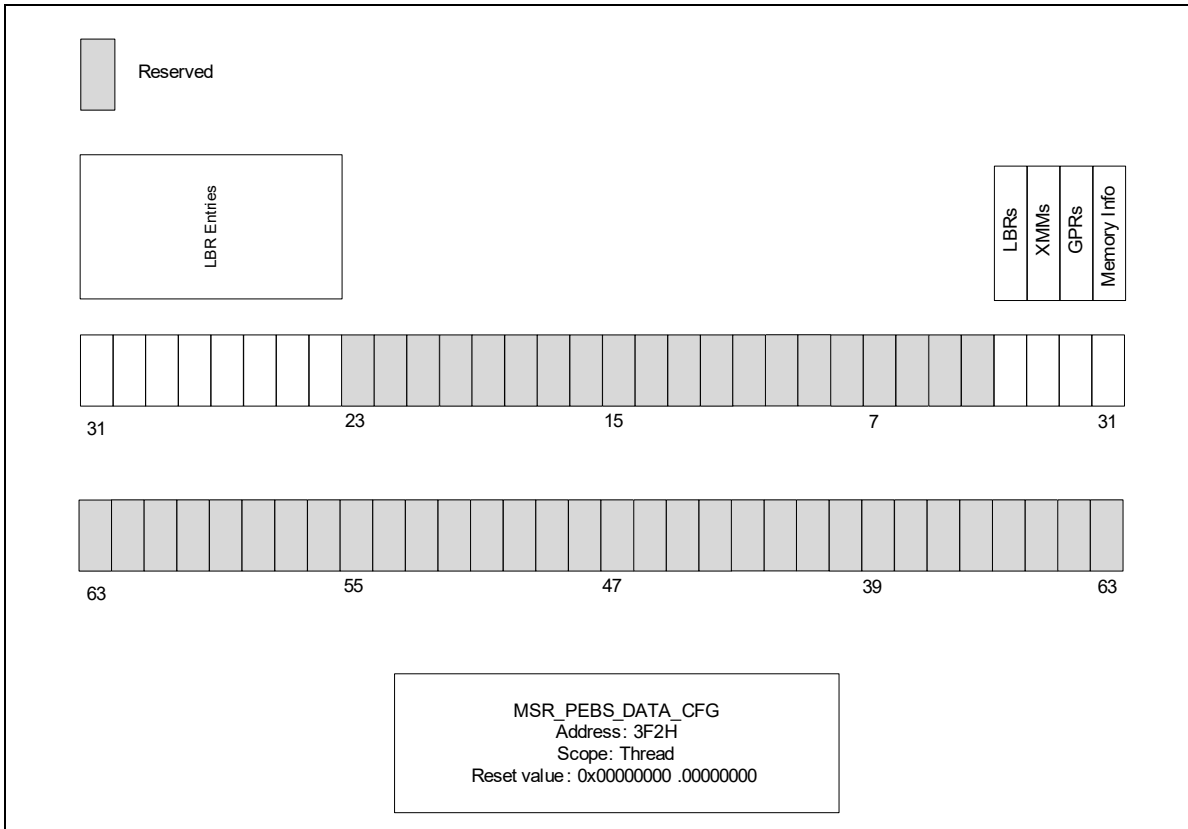


Figure 19-70. MSR_PEBS_DATA_CFG

Table 19-98. MSR_PEBS_CFG Programming¹

Bit	Bit Index	Access	Description
Memory Info	0	R/W	Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.
GPRs	1	R/W	Setting this bit will capture the contents of the General Purpose registers in the PEBS record.
XMMs	2	R/W	Setting this bit will capture the contents of the XMM registers in the PEBS record.
LBRs	3	R/W	Setting this bit will capture LBR TO, FROM and INFO in the PEBS record.
Reserved ²	23:4	NA	Reserved

Table 19-98. MSR_PEBS_CFG Programming¹

LBR Entries	31:24	R/W	Set the field to the desired number of entries minus 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, it is recommended to select an even number of LBR entries (programmed into LBR_entries as an odd number).
-------------	-------	-----	---

NOTES:

1. A write to the MSR will be ignored when IA32_MISC_ENABLE.PERFMON_AVAILABLE is zero (default).
2. Writing to the reserved bits will cause a GP fault.

19.9.2.4 PEBS Record Examples

The following example shows the layout of the PEBS record when all data groups are selected (all valid bits in MSR_PEBS_DATA_CFG are set) and maximum number of LBRs are selected. There are no gaps in the PEBS record when a subset of the groups are selected, thus keeping the layout compact. Implementations that do not support some features will have to pad zeroes in the corresponding fields.

Table 19-99. PEBS Record Example 1

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	
0x20	Memory Info	Memory Access Address	DLA
0x28		Memory Auxiliary Info	DATA_SRC
0x30		Memory Access Latency	Load Latency
0x38		TSX Auxiliary Info	HLE Information
0x40	GPRs	RFLAGS	
0x48		RIP	
0x50		RAX	
...		...	
0x88		RDI	
0x90		R8	
...		...	
0xC8		R15	
0xD0	XMMs	XMM0	New
...		...	
0x1C0		XMM15	

Table 19-99. PEBS Record Example 1

0x1D0	LBRs	LBR[TOS].FROM	New
0x1D8		LBR[TOS].TO	
0x1E0		LBR[TOS].INFO	
...		...	
0x4B8		LBR[TOS + 1].FROM	
0x4C0		LBR[TOS + 1].TO	
0x4C8		LBR[TOS + 1].INFO	

The following example shows the layout of the PEBS record when Basic, GPR, and LBR group with 3 LBR entries are selected.

Table 19-100. PEBS Record Example 2

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	
0x20	GPRs	RFLAGS	
0x28		RIP	
0x30		RAX	
...		...	
0x68		RDI	
0x70		R8	
...		...	
0xA8	R15		
0xB0	LBRs	LBR[TOS].FROM	New
0xB8		LBR[TOS].TO	
0xC0		LBR[TOS].INFO	
...		...	
0xE0		LBR[TOS + 1].FROM	
0xE8		LBR[TOS + 1].TO	
0xF0		LBR[TOS + 1].INFO	

19.9.3 Precise Distribution of Instructions Retired (PDIR) Facility

Precise Distribution of Instructions Retired Facility is available via PEBS on some microarchitectures. Refer to Section 19.3.4.4.4. Counters that support PDIR also vary. See the processor specific sections for availability.

19.9.4 Reduced Skid PEBS

For precise events, upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. The Reduced Skid mechanism mitigates the “skid” problem by providing an early indication of when the counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow thus greatly reducing skid.

This mechanism is a superset of the PDIR mechanism available in the Sandy Bridge microarchitecture. See Section 19.3.4.4.4

In the Goldmont microarchitecture, the mechanism applies to all precise events including, `INST_RETIRED`, except for `UOPS_RETIRED`. However, the Reduced Skid mechanism is disabled for any counter when the `INV`, `ANY`, `E`, or `CMASK` fields are set.

With Reduced Skid PEBS, the skid is precisely one event occurrence. Hence if counting `INST_RETIRED`, PEBS will indicate the instruction that follows that which caused the counter to overflow.

For the Reduced Skid mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via `IA32_PERF_GLOBAL_CTRL` MSR) before changes to the configuration (e.g., what event is specified in `IA32_PERFEVTSELx` or whether PEBS is enabled for that counter via `IA32_PEBS_ENABLE`) or counter value (MSR write to `IA32_PMCx` and `IA32_A_PMCx`).

19.9.5 EPT-Friendly PEBS

The 3rd generation Intel Xeon Scalable Family of processors based on Ice Lake microarchitecture (and later processors) and the 12th generation Intel Core processor (and later processors) support VMX guest use of PEBS when the DS Area (including the PEBS Buffer and DS Management Area) is allocated from a paged pool of EPT pages. In such a configuration PEBS DS Area accesses may result in VM exits (e.g., EPT violations due to “lazy” EPT page-table entry propagation), and in such cases the PEBS record will not be lost but instead will “skid” to after the subsequent VM Entry back to the guest. For precise events the guest will observe that the record skid by one event occurrence, while for non-precise events the record will skid by one instruction.

19.9.6 PDist: Precise Distribution

PDist eliminates any skid or shadowing effects from PEBS. With PDist, the PEBS record will be generated precisely upon completion of the instruction or operation that causes the counter to overflow (there is no “wait for next occurrence” by default).

PDist is supported by selected counters, and is only supported when those counters are programmed to count select precise events¹. The legacy PEBS behavior applies to counters that do not support PDist, unless specified otherwise. PDist requires that the `INV`, `ANY`, `E`, and `CMASK` fields are cleared. Which counters support PDist, and which events are supported for PDist, is model-specific. Further, the counter reload value must not be lesser than 127 for PDist to operate.

For the PDist mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via `IA32_PERF_GLOBAL_CTRL` MSR) before changes to the configuration (e.g., what event is specified in `IA32_PERFEVTSELx` or whether PEBS is enabled for that counter via `IA32_PEBS_ENABLE`) or counter value (MSR write to `IA32_PMCx` and `IA32_A_PMCx`).

1. To determine whether an event is precise or supports PDist, consult the relevant attribute in the event lists at <https://download.01.org/perfmon/>.

19.9.7 Load Latency Facility

The load latency facility provides software a means to characterize the latencies of memory load operations to different levels of cache/memory hierarchy. This facility requires a processor supporting the enhanced PEBS record format in the PEBS buffer.

Beginning with 12th generation Intel Core processors, the load latency facility supports all fields in Table 19-94, “Updated Memory Access Info Group”, in addition to the Memory Access Address field:

- The **Instruction Latency** field measures the load latency from the load's first dispatch until final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches and data dependencies).
- The **Cache Latency** field measures the subset of cache access latency in core cycles. It starts from the actual cache access until the data is returned by the memory subsystem. The latency is reported for retired demand load operations in core cycles (it does not account for memory ordering blocks).
- The **Data Source** field is an encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 19-101. In the descriptions, local memory refers to system memory physically attached to a processor package, and remote memory refers to system memory or cache physically attached to another processor package (in a server product).
- Through the **Access Info** field, load latency features binary indications on certain blocks that the load operation may have encountered. Refer to STLB-miss, Is-Lock, Data-Blk and Address-Blk fields in Table 19-94.

Table 19-101. Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)

Encoding	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request).
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. This request was merged into an outstanding cache miss to same cache-line address.
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. This request was satisfied by the L3 cache with no coherency actions performed (snooping).
05H	XCORE MISS. This request was satisfied by the L3 cache but involved a coherency check in some sibling core(s).
06H	XCORE HIT. This request was satisfied by the L3 cache but involved a coherency check that hit a non-modified copy in a sibling core.
07H	XCORE FWD. This request was satisfied by a sibling core where either a modified (cross-core HITM) or a non-modified (cross-core FWD) cache-line copy was found.
08H	Local Far Memory. This request has missed the L3 cache and was serviced by local far memory.
09H	Remote Far Memory. This request has missed the L3 cache and was serviced by remote far memory.
0AH	Local Near Memory. This request has missed the L3 cache and was serviced by local near memory.
0BH	Remote Near Memory. This request has missed the L3 cache and was serviced by remote near memory.
0CH	Remote FWD. This request has missed the L3 cache and a non-modified cache-line copy was forwarded from a remote cache.
0DH	Remote HITM. This request has missed the L3 cache and a modified cache-line was forwarded from a remote cache.
0EH	I/O. Request of input/output operation.
0FH	UC. The request was to uncacheable memory.

To use this feature, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- One of the relevant IA32_PERFEVTSELx MSRs is programmed to specify the event unit MEM_TRANS_RETIRED.LOAD_LATENCY (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter, IA32_PMCx, will accumulate event counts for architecturally visible loads which exceed the programmed

latency threshold specified separately in an MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.

- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with instruction latency greater than this value are eligible for counting and PEBS data reporting. The minimum value that may be programmed in this register is 1.
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register.

Refer to Section 19.3.4.4.2 for further implementation details of Load Latency.

19.9.8 Store Latency Facility

Store latency support is available on the 12th generation Intel Core processor. Store latency is a PEBS extension that provides a means to profile store memory accesses in the system. It complements the load latency facility.

Store latency leverages the PEBS facility where it can provide additional information about sampled stores. The additional information includes the data address, memory auxiliary information, and the cache latency of the store access. Normal stores (those preceded with a read-for-ownership) as well as streaming stores are supported by the store latency facility.

Memory store operations typically do not limit performance since they update the memory with no operation that directly depends on them. Thus, data out of this facility should be carefully used once stores are suspected as a performance limiter; for example, once the TMA node of Backend_Bound.Memory_Bound.Store_Bound is flagged¹.

To enable the store latency facility, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- Program the MEM_TRANS_RETIRED.STORE_SAMPLE event on general-purpose performance-monitoring counter 0 (IA32_PERFEVTSELO[15:0] = 2CDH).
- Setup the PEBS buffer to hold at least two records, setting both 'PEBS Absolute Maximum' and 'PEBS Interrupt Threshold', should any other counter be used by PEBS (that is whenever IA32_PEBS_ENABLE[x] ≠ 0 for x ≠ 0).
- Set IA32_PEBS_ENABLE[0].

The store latency information is written into a PEBS record as shown in Table 19-48.

The store latency relies on the PEBS facility, so the PEBS configuration must be completed first. Unlike load latency, there is no option to filter on a subset of stores that exceed a certain threshold.

1. For more details about the method, refer to Section B.1, "Top-Down Analysis Method" of the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

12. Updates to Chapter 24, Volume 3C

Change bars and green text show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Added primary VM exit control bit 30, "Save IA32_PERF_GLOBAL_CTRL" to Table 24-13, "Definitions of Primary VM-Exit Controls".
- Added secondary processor-based VM-execution control bit 27 information (enable PCONFIG).
- Added PCONFIG-Exiting Bitmap information.

24.1 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 24.4.2) 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 24-1 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 24.11.3.

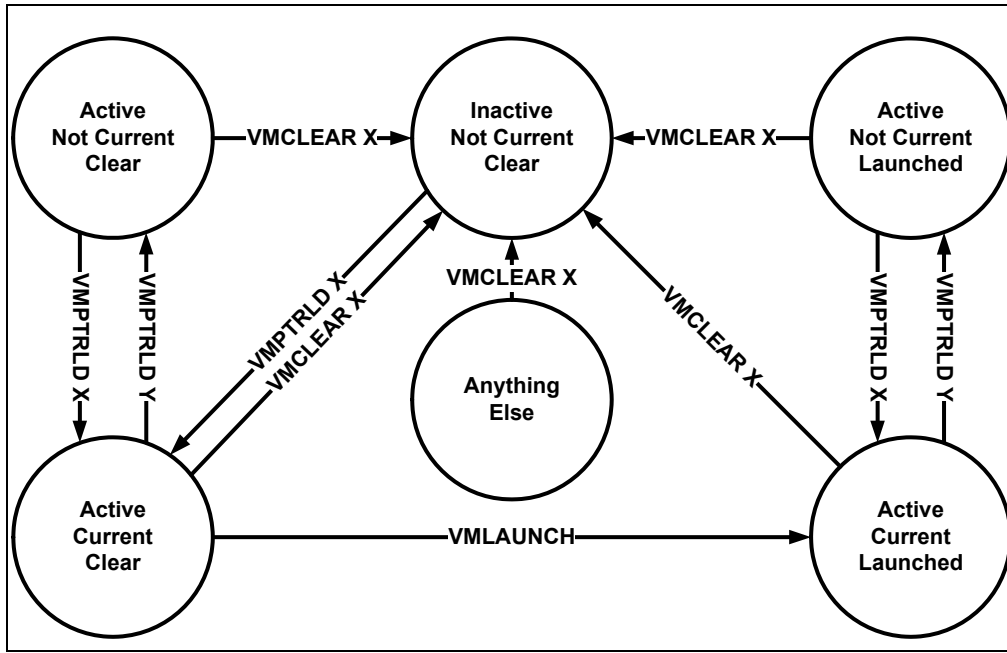


Figure 24-1. States of VMCS X

Because a shadow VMCS (see Section 24.10) cannot be used for VM entry, the launch state of a shadow VMCS is not meaningful. Figure 24-1 does not illustrate all the ways in which a shadow VMCS may be made active.

24.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.¹ The format of a VMCS region is given in Table 24-1.

Table 24-1. Format of the VMCS Region

Byte Offset	Contents
0	Bits 30:0: VMCS revision identifier Bit 31: shadow-VMCS indicator (see Section 24.10)
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 24.10).

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 24.6.2) 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 24.10). VMPTRLD fails if the shadow-VMCS indicator is set and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control. Software can discover support for this setting by reading the VMX capability MSR IA32_VMX_PROCBASED_CTL2 (see Appendix A.3.3).

The next 4 bytes of the VMCS region are used for the **VMX-abort indicator**. The contents of these bits do not control processor operation in any way. A logical processor writes a non-zero value into these bits if a VMX abort occurs (see Section 27.7). 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 24.3 through Section 24.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 24.11.4) in writeback cacheable memory. Future implementations may allow or require a different memory type³. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

24.3 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).

24.4 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 26.3.2 and Section 27.3 for details.

24.4.1 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 24-2 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 24-2. 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 10-1 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

Table 24-2. 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 26.3.1.2).

- 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.

24.4.2 Guest Non-Register State

In addition to the register state described in Section 24.4.1, 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 24-3.

Table 24-3. 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 <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B</i> . 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 <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A</i> . 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 31.2, “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 27.1.

2. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 24-3. 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 <i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A</i> and Section 31.8, "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 25.3 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 24.6.1) 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 26.3.1.5.

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 24-4.

Table 24-4. 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.
11:4	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5.
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7.
13	Reserved	VM entry fails if this bit is not 0. See Section 26.3.1.5.
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 26.3.1.5.

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 24-4. Format of Pending-Debug-Exceptions (Contd.)

Bit Position(s)	Bit Name	Notes
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 <i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1</i>). ¹
63:17	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5. 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 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 24.10). Otherwise, software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 26.3.1.5).
- **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 25.5.1 and Section 26.7.4.
- **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 29 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 28.3.6.

24.5 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 27.5).

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 27.5 for details of how state is loaded on VM exits.

24.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 24.6.1 through Section 24.6.8.

24.6.1 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 24-5 lists the controls. See Chapter 27 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 25.2).

Table 24-5. 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 25.3).
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 24-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 24.6.2).
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 25.5.1. A VM exit occurs when the timer counts down to zero; see Section 25.2.
7	Process posted interrupts	If this control is 1, the processor treats interrupts with the posted-interrupt notification vector (see Section 24.6.8) specially, updating the virtual-APIC page with posted-interrupt requests (see Section 29.6).

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 26.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR IA32_VMX_PINBASED_CTLs will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PINBASED_CTLs MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

24.6.2 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 24-6 lists the primary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-6. 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 24.4.2).
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 24.6.5 and Section 25.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	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 25.1.2), as do task switches (see Section 25.2).

Table 24-6. 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 24.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 25.1.3. 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 29.
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 24.4.2).
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 24.6.4 and Section 25.1.3). 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 25.5.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 24.6.9 and Section 25.1.3). 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 26.2.1.1).

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 24-7 lists the secondary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-7. 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 29.4.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 28.3.
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 29.5.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 28.1.
6	WBINVD exiting	This control determines whether executions of WBINVD and WBNOINVD cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpagged protected mode or in real-address mode.
8	APIC-register virtualization	If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 29.4 and Section 29.5.
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 24.6.13 and Section 25.1.3).
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 25.5.6.
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 24.10 and Section 30.3.
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 24.6.16 and Section 25.1.3.
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 28.3.6.
18	EPT-violation #VE	If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 25.5.7.
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 32).
20	Enable XSAVES/XRSTORS	If this control is 0, any execution of XSAVES or XRSTORS causes a #UD.
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 28.
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 28.3.4.

Table 24-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
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 25.5.4.
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 24.6.5 and Section 25.3).
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 24.6.17 and Section 25.1.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTL2 (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 26.2.1.1).

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 24-8 lists the tertiary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-8. 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 28.3.3.2.
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 28.3.3.2.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTL3 (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 26.2.1.1).

24.6.3 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 25.2 for details.

24.6.4 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 25.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

24.6.5 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 25 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation.

24.6.6 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 27 for details regarding how these fields affect VMX non-root operation.

24.6.7 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 26.2) 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.

24.6.8 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 10.4.4, “Local APIC Status and Location” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A* and *Intel® 64 Architecture Processor Topology Enumeration*).¹

- If the local APIC is in x2APIC mode, it can access the local APIC's registers using the RDMSR and WRMSR instructions (see *Intel® 64 Architecture Processor Topology Enumeration*).
- In 64-bit mode, it can access the local APIC's task-priority register (TPR) using the MOV CR8 instruction.

There are five processor-based VM-execution controls (see Section 24.6.2) that control such accesses. There are "use TPR shadow", "virtualize APIC accesses", "virtualize x2APIC mode", "virtual-interrupt delivery", and "APIC-register 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 29.4.

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 29.

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 29.3).
- Accesses to the APIC-access page if, in addition, the "virtualize APIC accesses" VM-execution control is 1 (see Section 29.4).
- 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 29.5).

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 29.1.1) 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 29.1.2.

The TPR threshold exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.

- **EOI-exit bitmap** (4 fields; 64 bits each). These fields are supported only on processors that support the 1-setting of the "virtual-interrupt delivery" VM-execution control. They are used to determine which virtualized writes to the APIC's EOI register cause VM exits:

- EOI_EXIT0 contains bits for vectors from 0 (bit 0) to 63 (bit 63).
- EOI_EXIT1 contains bits for vectors from 64 (bit 0) to 127 (bit 63).
- EOI_EXIT2 contains bits for vectors from 128 (bit 0) to 191 (bit 63).
- EOI_EXIT3 contains bits for vectors from 192 (bit 0) to 255 (bit 63).

See Section 29.1.4 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 29.6 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 29.6 for more information on the use of this field.

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

24.6.9 MSR-Bitmap Address

On processors that support the 1-setting of the “use MSR bitmaps” VM-execution control, the VM-execution control fields include the 64-bit physical address of four contiguous **MSR bitmaps**, which are each 1-KByte in size. This field does not exist on processors that do not support the 1-setting of that control. The four bitmaps are:

- **Read bitmap for low MSRs** (located at the MSR-bitmap address). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Read bitmap for high MSRs** (located at the MSR-bitmap address plus 1024). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Write bitmap for low MSRs** (located at the MSR-bitmap address plus 2048). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.
- **Write bitmap for high MSRs** (located at the MSR-bitmap address plus 3072). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

A logical processor uses these bitmaps if and only if the “use MSR bitmaps” control is 1. If the bitmaps are used, an execution of RDMSR or WRMSR causes a VM exit if the value of RCX is in neither of the ranges covered by the bitmaps or if the appropriate bit in the MSR bitmaps (corresponding to the instruction and the RCX value) is 1. See Section 25.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

24.6.10 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 31.15.2. VM entries that return from SMM use this field as described in Section 31.15.4.

24.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of the base of EPT PML4 table (see Section 28.3.2), as well as other EPT configuration information. The format of this field is shown in Table 24-9.

Table 24-9. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 28.3.7): 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 28.3.2)
6	Setting this control to 1 enables accessed and dirty flags for EPT (see Section 28.3.5) ²
7	Setting this control to 1 enables enforcement of access rights for supervisor shadow-stack pages (see Section 28.3.3.2) ³
11:8	Reserved
N-1:12	Bits N-1:12 of the physical address of the 4-KByte aligned EPT PML4 table ⁴
63:N	Reserved

NOTES:

1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT paging-structure memory types are supported.
2. Not all processors support accessed and dirty flags for EPT. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
3. Not all processors enforce access rights for shadow-stack pages. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
4. N is the physical-address width supported by the logical processor. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

The EPTP exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

24.6.12 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 28.1 for details regarding the use of this field.

24.6.13 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 25.1.3 for more details regarding PAUSE-loop exiting.

24.6.14 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 24-10 lists the VM-function controls. See Section 25.5.6 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-10. 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 25.5.6.3.

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 26.2.1.1).

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 25.5.6.3).

24.6.15 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 24.10 and Section 30.3).

24.6.16 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 25.1.3 for more information.

24.6.17 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 25.1.3 for more information.

24.6.18 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 25.1.3 for more information.

24.6.19 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 28.3.6.

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.

24.6.20 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 25.5.7.2.
- **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 25.5.6.3).

24.6.21 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 25.1.3 and Section 25.3).

24.6.22 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 28.3.4). 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 28.3.4.2). The format of this field is shown in Table 24-9.

Table 24-11. 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.

24.6.23 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 24-12.

Table 24-12. 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.

24.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 24.7.1 and Section 24.7.2.

24.7.1 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 24-13 lists the primary VM-exit controls. See Chapter 27 for complete details of how these controls affect VM exits.

Table 24-13. Definitions of Primary VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: <ul style="list-style-type: none"> ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt’s vector. The vector is stored in the VM-exit interruption-information field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the VM-exit interruption-information field is marked invalid.
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.
22	Save VMX-preemption timer value	This control determines whether the value of the VMX-preemption timer is saved on VM exit.
23	Clear IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is cleared on VM exit.
24	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM exit or a VMCS packet on an SMM VM exit (see Chapter 32).
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.
28	Load CET state	This control determines whether CET-related MSRs and SPP 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_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is saved on VM exit.

Table 24-13. Definitions of Primary VM-Exit Controls (Contd.)

Bit Position(s)	Name	Description
31	Activate secondary controls	This control determines whether the secondary VM-exit controls are used. If this control is 0, the logical processor operates as if all the secondary VM-exit controls were also 0.

NOTES:

1. Since the Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CRO.PG and IA32_EFER.LME, and since CRO.PG is always 1 in VMX root operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX root operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_EXIT_CTLS and IA32_VMX_TRUE_EXIT_CTLS (see Appendix A.4) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 26.2.1.2).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8, 10, 11, 13, 14, 16, and 17. The VMX capability MSR IA32_VMX_EXIT_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_EXIT_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-exit controls determines whether the secondary VM-exit controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary VM-exit controls were 0. Processors that support only the 0-setting of bit 31 of the primary VM-exit controls do not support the secondary VM-exit controls.

Currently, no secondary VM-exit controls are defined, and all bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_EXIT_CTLS2 (see Appendix A.4.2) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 26.2.1.2).

24.7.2 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 24-14. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 24-14. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 27.4 for how this area is used on VM exits.

The following VM-exit control fields determine how MSRs are loaded on VM exits:

1. Future implementations may allow more MSRs to be stored reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.¹
- **VM-exit MSR-load address** (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count (see Table 24-14). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 27.6 for how this area is used on VM exits.

24.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 24.8.1 through 24.8.3.

24.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 24-15 lists the controls supported. See Chapter 24 for how these controls affect VM entries.

Table 24-15. Definitions of VM-Entry Controls

Bit Position(s)	Name	Description
2	Load debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are loaded on VM entry. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	IA-32e mode guest	On processors that support Intel 64 architecture, this control determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA as part of VM entry. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
10	Entry to SMM	This control determines whether the logical processor is in system-management mode (SMM) after VM entry. This control must be 0 for any VM entry from outside SMM.
11	Deactivate dual-monitor treatment	If set to 1, the default treatment of SMIs and SMM is in effect after the VM entry (see Section 31.15.7). This control must be 0 for any VM entry from outside SMM.
13	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM entry.
14	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM entry.
15	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM entry.
16	Load IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is loaded on VM entry.
17	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM entry or a VMCS packet on a VM entry that returns from SMM (see Chapter 32).
18	Load IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is loaded on VM entry.
20	Load CET state	This control determines whether CET-related MSRs and SPP 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.

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).

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 27.2).

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 26.2.1.3).

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.

24.8.2 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 24-14. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 26.4 for details of how this area is used on VM entries.

24.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event through the IDT (after all guest state and MSRs have been loaded). This process is called **event injection** and is controlled by the following three VM-entry control fields:

- **VM-entry interruption-information field** (32 bits). This field provides details about the event to be injected. Table 24-16 describes the field.

Table 24-16. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g., #PF) 4: Software interrupt (INT <i>n</i>) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved

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 24-16. Format of the VM-Entry Interruption-Information Field (Contd.)

Bit Position(s)	Content
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 27.2).
- **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 26.6 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.

24.9 VM-EXIT INFORMATION FIELDS

The VMCS contains a section of fields that contain information about the most recent VM exit.

On some processors, attempts to write to these fields with VMWRITE fail (see “VMWRITE—Write Field to Virtual-Machine Control Structure” in Chapter 30).³

24.9.1 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 24-17.

Table 24-17. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason
16	Always cleared to 0
26:17	Not currently defined

1. The type hardware exception should be used for all other debug exceptions.
 2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with values 1 or 3 for *n*.
 3. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

Table 24-17. Format of Exit Reason (Contd.)

Bit Position(s)	Contents
27	A VM exit saves this bit as 1 to indicate that the VM exit was incident to enclave mode.
28	Pending MTF VM exit
29	VM exit from VMX root operation
30	Not currently defined
31	VM-entry failure (0 = true VM exit; 1 = VM-entry failure)

- Bits 15:0 provide basic information about the cause of the VM exit (if bit 31 is clear) or of the VM-entry failure (if bit 31 is set). Appendix C enumerates the basic exit reasons.
- Bit 16 is always cleared to 0.
- Bit 27 is set to 1 if the VM exit occurred while the logical processor was in enclave mode.
A VM exit also sets this bit if it is incident to delivery of an event injected by VM entry and the guest interruptibility-state field indicates an enclave interrupt (bit 4 of the field is 1). See Section 27.2.1 for details.
- Bit 28 is set only by an SMM VM exit (see Section 31.15.2) that took priority over an MTF VM exit (see Section 25.5.2) that would have occurred had the SMM VM exit not occurred. See Section 31.15.2.3.
- 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 31.15.2.
- Because some VM-entry failures load processor state from the host-state area (see Section 26.8), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose.
- **Exit qualification** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field contains additional information about the cause of VM exits due to the following: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); task switches; INVEPT; INVLPG; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; and MWAIT. The format of the field depends on the cause of the VM exit. See Section 27.2.1 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 27.2.1 and Section 31.15.2.3 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 27.2.1 for details of when and how this field is used.

24.9.2 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 24-18 describes this field.

Table 24-18. Format of the VM-Exit Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Not used 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	NMI unblocking due to IRET
30:13	Not currently defined
31	Valid

- **VM-exit interruption error code** (32 bits). For VM exits caused by hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

Section 27.2.2 provides details of how these fields are saved on VM exits.

24.9.3 Information for VM Exits That Occur During Event Delivery

Additional information is provided for VM exits that occur during event delivery in VMX non-root operation.¹ This information is provided in the following fields:

- **IDT-vectoring information** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 24-19 describes this field.

Table 24-19. Format of the IDT-Vectoring Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
30:12	Not currently defined
31	Valid

- **IDT-vectoring error code** (32 bits). For VM exits that occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

See Section 27.2.4 provides details of how these fields are saved on VM exits.

1. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 26.6.1.2.

24.9.4 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 27.2.5 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 27.2.5 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).

24.9.5 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.

24.10 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 24-1): 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 24.6.2).

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 26.1).
- 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 25.1.3).
 - 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 30.3).
 - 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 26.3.1.5).

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 24.11.1). Before modifying the shadow-VMCS indicator, software should execute `VMCLEAR` for the VMCS to ensure that it is not active.

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).

24.11 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.

24.11.1 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 24-1) 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 24.11.2). Software should never access or modify the VMCS data of an active VMCS using ordinary memory operations, in part because the format used to store the VMCS data is implementation-specific and not architecturally defined, and also because a logical processor may maintain some VMCS data of an active VMCS on the processor and not in the VMCS region. The following items detail some of the hazards of accessing VMCS data using ordinary memory operations:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS. Results may vary from time to time or from logical processor to logical processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. Doing so may cause the VMCS to become corrupted (see below).

(Software can avoid these hazards by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.)

If a logical processor leaves VMX operation, any VMCSs active on that logical processor may be corrupted (see below). To prevent such corruption of a VMCS that may be used either after a return to VMX operation or on another logical processor, software should execute VMCLEAR for that VMCS before executing the VMXOFF instruction or removing power from the processor (e.g., as part of a transition to the S3 and S4 power states).

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 25 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.

24.11.2 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 30 for a description of these instructions.

1. As noted in Section 24.1, execution of the VMPTRLD instruction makes a VMCS 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.

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 24-20.

Table 24-20. Structure of VMCS Component Encoding

Bit Position(s)	Contents
0	Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural-width fields
9:1	Index
11:10	Type: 0: control 1: VM-exit information 2: guest state 3: host state
12	Reserved (must be 0)
14:13	Width: 0: 16-bit 1: 64-bit 2: 32-bit 3: natural-width
31:15	Reserved (must be 0)

The following items detail the meaning of the bits in each encoding:

- **Field width.** Bits 14:13 encode the width of the field.
 - A value of 0 indicates a 16-bit field.
 - A value of 1 indicates a 64-bit field.
 - A value of 2 indicates a 32-bit field.
 - A value of 3 indicates a **natural-width** field. Such fields have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

Fields whose encodings use value 1 are specially treated to allow 32-bit software access to all 64 bits of the field. Such access is allowed by defining, for each such field, an encoding that allows direct access to the high 32 bits of the field. See below.
- **Field type.** Bits 11:10 encode the type of VMCS field: control, guest-state, host-state, or VM-exit information. (The last category also includes the VM-instruction error field.)
- **Index.** Bits 9:1 distinguish components with the same field width and type.
- **Access type.** Bit 0 must be 0 for all fields except for 64-bit fields (those with field-width 1; see above). A VMREAD or VMWRITE using an encoding with this bit cleared to 0 accesses the entire field. For a 64-bit field with field-width 1, a VMREAD or VMWRITE using an encoding with this bit set to 1 accesses only the high 32 bits of the field.

Appendix B gives the encodings of all fields in the VMCS.

The following describes the operation of VMREAD and VMWRITE based on processor mode, VMCS-field width, and access type:

- 16-bit fields:
 - A VMREAD returns the value of the field in bits 15:0 of the destination operand; other bits of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 15:0 of the source operand into the VMCS field; other bits of the source operand are not used.
- 32-bit fields:

- A VMREAD returns the value of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
- A VMWRITE writes the value of bits 31:0 of the source operand into the VMCS field; in 64-bit mode, bits 63:32 of the source operand are not used.
- 64-bit fields and natural-width fields using the full access type outside IA-32e mode.
 - A VMREAD returns the value of bits 31:0 of the field in its destination operand; bits 63:32 of the field are ignored.
 - A VMWRITE writes the value of its source operand to bits 31:0 of the field and clears bits 63:32 of the field.
- 64-bit fields and natural-width fields using the full access type in 64-bit mode (only on processors that support Intel 64 architecture).
 - A VMREAD returns the value of the field in bits 63:0 of the destination operand
 - A VMWRITE writes the value of bits 63:0 of the source operand into the VMCS field.
- 64-bit fields using the high access type.
 - A VMREAD returns the value of bits 63:32 of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 31:0 of the source operand to bits 63:32 of the field; in 64-bit mode, bits 63:32 of the source operand are not used.

Software seeking to read a 64-bit field outside IA-32e mode can use VMREAD with the full access type (reading bits 31:0 of the field) and VMREAD with the high access type (reading bits 63:32 of the field); the order of the two VMREAD executions is not important. Software seeking to modify a 64-bit field outside IA-32e mode should first use VMWRITE with the full access type (establishing bits 31:0 of the field while clearing bits 63:32) and then use VMWRITE with the high access type (establishing bits 63:32 of the field).

24.11.3 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 24.1). 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 24-1 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 24.11.1), 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.

24.11.4 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 24.11.1). 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 28.4.3); the virtual-APIC page (Section 29.1); the posted interrupt descriptor (Section 29.6); and the virtualization-exception information area (Section 25.5.7.2).

24.11.5 VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region)¹ that the logical processor uses to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. The VMXON pointer is subject to the limitations that apply to VMCS pointers:

- The VMXON pointer must be 4-KByte aligned (bits 11:0 must be zero).
- The VMXON pointer must not set any bits beyond the processor's physical-address width.^{2,3}

Before executing VMXON, software should write the VMCS revision identifier (see Section 24.2) 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 24.11.1).

1. The amount of memory required for the VMXON region is the same as that required for a VMCS region. This size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

3. If IA32_VMX_BASIC[48] is read as 1, the VMXON pointer must not set any bits in the range 63:32; see Appendix A.1.

13. Updates to Chapter 25, Volume 3C

Change bars and green text show changes to Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Added secondary processor-based VM-execution control bit 27 information (enable PCONFIG).
- Added an entry for PCONFIG in section 25.1.3, "Instructions That Cause VM Exits Conditionally".

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 25.1, “Instructions That Cause VM Exits”
- Section 25.2, “Other Causes of VM Exits”
- Section 25.3, “Changes to Instruction Behavior in VMX Non-Root Operation”
- Section 25.4, “Other Changes in VMX Non-Root Operation”
- Section 25.5, “Features Specific to VMX Non-Root Operation”
- Section 25.6, “Unrestricted Guests”

Chapter 26, “VM Entries,” describes the data control structures that govern VMX non-root operation. Chapter 26, “VM Entries,” describes the operation of VM entries by which the processor transitions from VMX root operation to VMX non-root operation. Chapter 25, “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 28, “VMX Support for Address Translation,” describes two features that support address translation in VMX non-root operation. Chapter 29, “APIC Virtualization and Virtual Interrupts,” describes features that support virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC) in VMX non-root operation.

25.1 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 27.1 details architectural state in the context of a VM exit.

Section 25.1.1 defines the prioritization between faults and VM exits for instructions subject to both. Section 25.1.2 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 25.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields (see Section 24.6).

25.1.1 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 25.1.3).
- 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 25.1.3.

- 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 25.1.2 or Section 25.1.3 (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.

25.1.2 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.

25.1.3 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 24.6.16).
 - 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 24.6.17).
 - The value of EAX is greater than or equal to 63 and bit 63 in the ENCLV-exiting bitmap is 1.
- **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.
 - 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 24.6.4). If an I/O

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 31.15.2.

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 24.6.2.

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 25.1.1 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 25.1.1 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 25.3).
- **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 24.6.18).
- 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 24.6.9 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.
- **UMWAIT.** The UMWAIT 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 31.15.3.

- **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 24.6.15 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 30, “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 24.6.15 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 30, “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 24.6.9 for details regarding how these bitmaps are identified.
- **XRSTORS.** The XRSTORS instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 24.6.21).
- **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 24.6.21).

25.2 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 24.6.3). 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 UD2.¹

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

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 .

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 25.6 for an exception.) Otherwise, the interrupt is delivered normally through the IDT. (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The interrupt is not delivered through the IDT 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 25.4.2.
- **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 31.15.2.¹
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 25.5.1 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.

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 24-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.7.5).
Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.
These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.
- If the “NMI-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no blocking of events by MOV SS and no blocking of events by STI

1. 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 31.14.1.

(see Table 24-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.7.6).

VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

25.3 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 23.8), 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.
- **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 24-3) 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 26.2.1.1.)
 - 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 25.1.3) 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 23.8) 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

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 24.6.2.

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 25.1.3), the value loaded from CR3 is a guest-physical address; see Section 28.3.1.
- **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 25.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 29.3.
- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 25.1.3) 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 23.8).
 - 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 25.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 28.3.1.
 - 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 25.1.3) 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 23.8).
- **MOV to CR8.** If the MOV to CR8 instruction does not cause a VM exit (see Section 25.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 29.3.
- **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-

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*.

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 25.1.3; if it does not cause such a VM exit, it operates normally.
- **RDMSR.** Section 25.1.3 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.
 - If the control is 1, RDMSR first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

The 1-setting of the “use TSC-offsetting” VM-execution control does not affect executions of RDMSR if ECX contains 6E0H (indicating the IA32_TSC_DEADLINE MSR). Such executions return the APIC-timer deadline relative to the actual timestamp counter without regard to the TSC offset.

 - If ECX is in the range 800H–8FFH (indicating an APIC MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 29.5.
- **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.
- **RDTSR.** Behavior of the RDTSR instruction is determined by the settings of the “RDTSR exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTSR operates normally.
 - If the “RDTSR 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, RDTSR 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, RDTSR 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 “RDTSR exiting” VM-execution control is 1, RDTSR 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,FFFFFFFFCH).
The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:
 - If either control is 0, the physical delay is the virtual delay.
 - If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
 - If the “RDTSC exiting” VM-execution control is 1, TPAUSE causes a VM exit.
- **UMONITOR.** Behavior of the UMONITOR instruction is determined by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, UMONITOR causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.

- If the “enable user wait and pause” VM-execution control is 1, UMONITOR operates normally.
- **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,FFFFFFFFCH).
 The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:
 - If either control is 0, the physical delay is the virtual delay.
 - If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
 - If the “RDTSC exiting” VM-execution control is 1, UMWAIT causes a VM exit.
- **WRMSR.** Section 25.1.3 identifies when executions of the WRMSR instruction cause VM exits. If such an execution neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
 - If ECX contains 79H (indicating IA32_BIOS_UPDT_TRIG MSR), no microcode update is loaded, and control passes to the next instruction. This implies that microcode updates cannot be loaded in VMX non-root operation.
 - On processors that support Intel PT but which do not allow it to be used in VMX operation, if ECX contains 570H (indicating the IA32_RTIT_CTL MSR), the instruction causes a general-protection exception.¹
 - If ECX contains 808H (indicating the TPR MSR), 80BH (the EOI MSR), or 83FH (self-IPI MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 29.5.
- **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 24.6.21):
 - XRSTORS causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XRSTORS operates normally.
- **XSAVES.** Behavior of the XSAVES instruction is determined first by the setting of the “enable XSAVES/XRSTORS” VM-execution control:
 - If the “enable XSAVES/XRSTORS” VM-execution control is 0, XSAVES causes an invalid-opcode exception (#UD).
 - If the “enable XSAVES/XRSTORS” VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 24.6.21):

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).

- 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.

25.4 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking and of task switches differ in VMX non-root operation as described in the following sections.

25.4.1 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).

25.4.2 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 27.2. 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 27.2.

25.5 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 25.5.1) and the monitor trap flag (Section 25.5.2), translation of guest-physical addresses (Section 25.5.3 and Section 25.5.4), APIC virtualization (Section 25.5.5), VM functions (Section 25.5.6), and virtualization exceptions (Section 25.5.7).

25.5.1 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 26.7.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 25.2).

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 31.14) 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 25.2).
- If the dual-monitor treatment of SMIs and SMM (see Section 31.15) 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 31.15.2 and Section 31.15.4 detail some differences.

25.5.2 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 26.6.1), 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 26.6.2), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0.
- If the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and a pending event (e.g., debug exception or interrupt) is delivered before an instruction can execute, an MTF VM exit is pending on the instruction boundary following delivery of the event (or any nested exception).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is a REP-prefixed string instruction:
 - If the first iteration of the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).
 - If the first iteration of the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary after that iteration.
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is the XBEGIN instruction. In this case, an MTF VM exit is pending at the fallback instruction address of the XBEGIN instruction. This behavior applies regardless of whether advanced debugging of RTM transactional regions has been enabled (see Section 16.3.7, “RTM-Enabled Debugger Support,” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is neither a REP-prefixed string instruction or the XBEGIN instruction:
 - If the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).¹
 - If the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary following execution of that instruction. If the instruction is INT1, INT3, or INTO, this boundary follows delivery of any software exception. If the instruction is INT *n*, this boundary follows delivery of a software interrupt. If the instruction is HLT, the MTF VM exit will be from the HLT activity state.

No MTF VM exit occurs if another VM exit occurs before reaching the instruction boundary on which an MTF VM exit would be pending (e.g., due to an exception or triple fault).

An MTF VM exit occurs on the instruction boundary on which it is pending unless a higher priority event takes precedence or the MTF VM exit is blocked due to the activity state:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over MTF VM exits. MTF VM exits take priority over debug-trap exceptions and lower priority events.
- No MTF VM exit occurs if the processor is in either the shutdown activity state or wait-for-SIPI activity state. If a non-maskable interrupt subsequently takes the logical processor out of the shutdown activity state without causing a VM exit, an MTF VM exit is pending after delivery of that interrupt.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 39.2 for details.

25.5.3 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 28.3.

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.

25.5.4 Translation of Guest-Physical Addresses Used by Intel Processor Trace

As described in Chapter 32, 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 25.5.4.1. Section 25.5.4.2 describes a mechanism that ensures that these VM exits do not cause loss of trace data.

25.5.4.1 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 28.3.3), resulting in VM exits. EPT violations resulting for the trace-output process always cause VM exits and are never converted to virtualization exceptions (Section 25.5.7.1).

If no EPT violation or EPT misconfiguration occurs and if page-modification logging (Section 28.3.6) 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 29.4.6.1.

25.5.4.2 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 25.5.4.1 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 26.5 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.

25.5.5 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 29.

25.5.6 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 25.5.6.1 explains how VM functions are enabled. Section 25.5.6.2 specifies the behavior of the VMFUNC instruction. Section 25.5.6.3 describes a specific VM function called **EPTP switching**.

25.5.6.1 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 24.6.14). 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).

25.5.6.2 General Operation of the VMFUNC Instruction

The VMFUNC instruction causes an invalid-opcode exception (#UD) if the “enable VM functions” VM-execution controls is 0¹ or the value of EAX is greater than 63 (only VM functions 0–63 can be enable). Otherwise, the instruction causes a VM exit if the bit at position EAX is 0 in the VM-function controls (the selected VM function is not enabled). If such a VM exit occurs, the basic exit reason used is 59 (3BH), indicating “VMFUNC”, and the length of the VMFUNC instruction is saved into the VM-exit instruction-length field. If the instruction causes neither an invalid-opcode exception nor a VM exit due to a disabled VM function, it performs the functionality of the VM function specified by the value in EAX.

Individual VM functions may perform additional fault checking (e.g., one might cause a general-protection exception if CPL > 0). In addition, specific VM functions may include checks that might result in a VM exit. If such a VM exit occurs, VM-exit information is saved as described in the previous paragraph. The specification of a VM function may indicate that additional VM-exit information is provided.

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 24.6.2.

The specific behavior of the EPTP-switching VM function (including checks that result in VM exits) is given in Section 25.5.6.3.

25.5.6.3 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 28.3 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 24.6.14; because this structure contains 512 8-Byte entries, VMFUNC causes a VM exit if $ECX \geq 512$). If the selected entry is a valid EPTP value (it would not cause VM entry to fail; see Section 26.2.1.1), it is stored in the EPTP field of the current VMCS and is used for subsequent accesses using guest-physical addresses. The following pseudocode provides details:

```

IF ECX ≥ 512
  THEN VM exit;
ELSE
  tent_EPTP := 8 bytes from EPTP-list address + 8 * ECX;
  IF tent_EPTP is not a valid EPTP value (would cause VM entry to fail if in EPTP)
    THEN VM exit;
  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 25.5.7.2);
    FI;
  FI;
FI;

```

Execution of the EPTP-switching VM function does not modify the state of any registers; no flags are modified.

If the "Intel PT uses guest physical addresses" VM-execution control is 1 and IA32_RTIT_CTL.TraceEn = 1, any execution of the EPTP-switching VM function causes a VM exit.¹

As noted in Section 25.5.6.2, an execution of the EPTP-switching VM function that causes a VM exit (as specified above), uses the basic exit reason 59, indicating "VMFUNC". The length of the VMFUNC instruction is saved into the VM-exit instruction-length field. No additional VM-exit information is provided.

An execution of VMFUNC loads EPTP from the EPTP list (and thus does not cause a fault or VM exit) is called an **EPTP-switching VMFUNC**. After an EPTP-switching VMFUNC, control passes to the next instruction. The logical processor starts creating and using guest-physical and combined mappings associated with the new value of bits 51:12 of EPTP; the combined mappings created and used are associated with the current VPID and PCID (these are not changed by VMFUNC).² If the "enable VPID" VM-execution control is 0, an EPTP-switching VMFUNC invalidates combined mappings associated with VPID 0000H (for all PCIDs and for all EP4TA values, where EP4TA is the value of bits 51:12 of EPTP).

Because an EPTP-switching VMFUNC may change the translation of guest-physical addresses, it may affect use of the guest-physical address in CR3. The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration due to the translation of that guest-physical address through the new EPT paging structures. The following items provide details that apply if CR0.PG = 1:

- If 32-bit paging or 4-level paging³ is in use (either CR4.PAE = 0 or IA32_EFER.LMA = 1), the next memory access with a linear address uses the translation of the guest-physical address in CR3 through the new EPT

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.

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 25.5.7.2);

25.5.7 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 25.5.7.1 provides the details of how the processor determines whether an EPT violation causes a virtualization exception or a VM exit.

When the processor encounters a virtualization exception, it saves information about the exception to the virtualization-exception information area; see Section 25.5.7.2.

After saving virtualization-exception information, the processor delivers a virtualization exception as it would any other exception; see Section 25.5.7.3 for details.

25.5.7.1 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.

3. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

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.

- If the value of the EPT paging-structure entry is supported, the following cases apply:
 - If bit 7 of the entry is 1, or if the entry is an EPT PTE, the entry maps a page. If the processor uses such an entry to translate a guest-physical address, and if an access to that address causes an EPT violation, the EPT violation is convertible if and only if bit 63 of the entry is 0.
 - If bit 7 of the entry is 0 and the entry is not an EPT PTE, the entry references another EPT paging structure. The processor does not use the value of bit 63 of the entry to determine whether any subsequent EPT violation is convertible.

If an access to a guest-physical address causes an EPT violation, bit 63 of exactly one of the EPT paging-structure entries used to translate that address is used to determine whether the EPT violation is convertible: either a entry that is not present (if the guest-physical address does not translate to a physical address) or an entry that maps a page (if it does).

A convertible EPT violation instead causes a virtualization exception if the following all hold:

- CR0.PE = 1;
- the logical processor is not in the process of delivering an event through the IDT;
- the EPT violation does not result from the output process of Intel Processor Trace (Section 25.5.4); 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 25.5.7.2). Thus, once a virtualization exception occurs, another can occur only if software clears this field.

25.5.7.2 Virtualization-Exception Information

Virtualization exceptions save data into the virtualization-exception information area (see Section 24.6.20). Table 25-1 enumerates the data saved and the format of the area.

Table 25-1. 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 24.6.20 and Section 25.5.6.3)

A VMM may allow guest software to access the virtualization-exception information area. If it does, the guest software may modify that memory (e.g., to clear the 32-bit value at offset 4; see Section 25.5.7.1). (This is an exception to the general requirement given in Section 24.11.4.)

25.5.7.3 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 25.5.7.1).

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 27.2.2). 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 27.2.4). Specifically, the event is reported as a hardware exception with vector 20 and no error code.

25.6 UNRESTRICTED GUESTS

The first processors to support VMX operation require CR0.PE and CR0.PG to be 1 in VMX operation (see Section 23.8). This restriction implies that guest software cannot be run in unpagged protected mode or in real-address mode. Later processors support a VM-execution control called “unrestricted guest”.¹ If this control is 1, CR0.PE and CR0.PG may be 0 in VMX non-root operation. Such processors allow guest software to run in unpagged protected mode or in real-address mode. The following items describe the behavior of such software:

- The MOV CR0 instructions does not cause a general-protection exception simply because it would set either CR0.PE and CR0.PG to 0. See Section 25.3 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.
 - 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.² The guest memory type passed on to the EPT mechanism is WB (writeback).

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 24.6.2.

2. As noted in Section 26.2.1.1, the “enable EPT” VM-execution control must be 1 if the “unrestricted guest” VM-execution control is 1.

14. Updates to Chapter 27, Volume 3C

Change bars and green text show changes to Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Updated section 27.2.5, "Information for VM Exits Due to Instruction Execution", to add PCONFIG to the list of exiting instructions that report instruction length reliably.
- Updated section 27.3.1, "Saving Control Registers, Debug Registers, and MSRs" with the "save IA32_PERF_GLOBAL_CTRL" information.

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 25.1 through Section 25.2. 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 27.2.
2. Processor state is saved in the guest-state area (Section 27.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 27.4). 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 27.5):
 - 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 31.15.6 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 27.6). 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 27.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 27.2 through Section 27.6.

Section 31.15 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 31.15.2.

27.1 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 29.4), EPT violation, EPT misconfiguration, page-modification log-full event (see Section 28.3.6), or SPP-related event (see Section 28.3.4) that causes a VM exit.
- An event **results** in a VM exit if it causes a VM exit (directly or indirectly).

The following bullets detail when architectural state is and is not updated in response to VM exits:

- If an event causes a VM exit directly, it does not update architectural state as it would have if it had it not caused the VM exit:
 - A debug exception does not update DR6, DR7, or IA32_DEBUGCTL. (Information about the nature of the debug exception is saved in the exit qualification field.)
 - A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)

- An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
 - An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the “acknowledge interrupt on exit” VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
 - The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
 - If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT¹; or the TR register.
 - No last-exception record is made if the event that would do so directly causes a VM exit.
 - If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine-check event itself and not the resulting machine-check exception.
 - If the logical processor is in an inactive state (see Section 24.4.2) 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 25.5.2 and Section 26.7.8) 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 27.2.3.
- 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 27.2.3.
- 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 29.4), 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 27.3.4).
- 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 29.3). (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 29.5.
 - VM exits caused by APIC-write emulation (see Section 29.4.3.2) 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 24-3). 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 36, “Enclave Exiting Events”). An AEX modifies architectural state (Section 36.3). 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).

27.2 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 27.2.1 to Section 27.2.5 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

27.2.1 Basic VM-Exit Information

Section 24.9.1 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 27 of this field is set to 1 if the VM exit occurred while the logical processor was in enclave mode. Such VM exits include those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode. A VM exit also sets this bit if it is incident to delivery of an event injected by VM entry and the guest interruptibility-state field indicates an enclave interruption (bit 4 of the field is 1).
 - The remainder of the field (bits 31:28 and bits 26:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 31.15.2.3).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 29.4); EPT violations (see Section 28.3.3.2); EOI virtualization (see Section 29.1.4); APIC-write emulation (see Section 29.4.3.3); page-modification log full (see Section 28.3.6); and SPP-related events (see Section 28.3.4). For all other VM exits, this field is cleared. The following items provide details:
 - For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 27-1.

Table 27-1. 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.
12:4	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).

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 26.8.

Table 27-1. Exit Qualification for Debug Exceptions (Contd.)

Bit Position(s)	Contents
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 <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1</i>). ¹
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 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 27-2.
- 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 27-2. Exit Qualification for Task Switches

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Not currently defined
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
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 24.9.4 and Section 27.2.5) 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 27-3.
- For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 27-4.
- For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 27-5.
- 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 29.4), the exit qualification contains information about the access and has the format given in Table 27-6.¹

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.”

Table 27-3. 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

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 27-3. Exit Qualification for Control-Register Accesses (Contd.)

Bit Positions	Contents
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 27.2.4) 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 29.4.4 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 29.4.6), 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 27-7.

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

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).

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 27-4. 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 27-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
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.

Bit 12 reports "NMI unblocking due to IRET"; see Section 27.2.3.

Table 27-6. Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	<ul style="list-style-type: none"> ▪ If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. ▪ Undefined if the APIC-access VM exit is due a guest-physical access
15:12	<p>Access type:</p> <ul style="list-style-type: none"> 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 4 = linear access for monitoring 10 = guest-physical access during event delivery 11 = guest-physical access for monitoring or trace 15 = guest-physical access for an instruction fetch or during instruction execution <p>Other values not used</p>
16	This bit is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These includes guest-physical accesses related to trace output by Intel PT (see Section 25.5.4) and accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 19.9.5).
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

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 25.5.4) and accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 19.9.5).

- For VM exits caused as part of EOI virtualization (Section 29.1.4), 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 29.4.3.3), 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 28.3.6), bit 12 of the exit qualification reports “NMI unblocking due to IRET.” Bit 16 is set if the VM exit occurs during TAPT or EPT-friendly PEBS. All other bits of the exit qualification are undefined.
- For a VM exit due to an SPP-related event (Section 28.3.4), bit 11 of the exit qualification indicates the type of event: 0 indicates an SPP misconfiguration and 1 indicates an SPP miss. Bit 12 of the exit qualification reports “NMI unblocking due to IRET.” Bit 16 is set if the VM exit occurs during TAPT or EPT-friendly PEBS. 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.
 - VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 27-7; 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-

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 3FOH.

Table 27-7. Exit Qualification for EPT Violations

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read. ¹
1	Set if the access causing the EPT violation was a data write. ¹
2	Set if the access causing the EPT violation was an instruction fetch.
3	The logical-AND of bit 0 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was readable). ²
4	The logical-AND of bit 1 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was writeable).
5	The logical-AND of bit 2 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. If the “mode-based execute control for EPT” VM-execution control is 0, this indicates whether the guest-physical address was executable. If that control is 1, this indicates whether the guest-physical address was executable for supervisor-mode linear addresses.
6	If the “mode-based execute control” VM-execution control is 0, the value of this bit is undefined. If that control is 1, this bit is the logical-AND of bit 10 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. In this case, it indicates whether the guest-physical address was executable for user-mode linear addresses.
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTes as part of the execution of the MOV CR instruction and those due to trace-address pre-translation (TAPT; Section 25.5.4).
8	If bit 7 is 1: <ul style="list-style-type: none"> ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).
9	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ³ this bit is 0 if the linear address is a supervisor-mode linear address and 1 if it is a user-mode linear address. (If CRO.PG = 0, the translation of every linear address is a user-mode linear address and thus this bit will be 1.) Otherwise, this bit is undefined.
10	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ³ this bit is 0 if paging translates the linear address to a read-only page and 1 if it translates to a read/write page. (If CRO.PG = 0, every linear address is read/write and thus this bit will be 1.) Otherwise, this bit is undefined.
11	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ³ this bit is 0 if paging translates the linear address to an executable page and 1 if it translates to an execute-disable page. (If CRO.PG = 0, CR4.PAE = 0, or IA32_EFER.NXE = 0, every linear address is executable and thus this bit will be 0.) Otherwise, this bit is undefined.
12	NMI unblocking due to IRET (see Section 27.2.3).
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.

Table 27-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
15	This bit is set if the EPT violation was caused as a result of guest-paging verification. See Section 28.3.3.2.
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 25.5.4.) 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 28.3.3.2). If such an access causes an EPT violation, the processor sets both bit 0 and bit 1 of the exit qualification.
2. Bits 5:3 are cleared to 0 if any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present (see Section 28.3.2).
3. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the EPT violation occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

- VM exits due to SPP-related events.
- For all other VM exits, the field is undefined.
- **Guest-physical address.** For a VM exit due to an EPT violation, an EPT misconfiguration, or an SPP-related event, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

If the EPT violation or EPT misconfiguration occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

27.2.2 Information for VM Exits Due to Vectored Events

Section 24.9.2 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 24-18). The following items detail how this field is established for VM exits due to these events:
 - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 5 (privileged software exception), or 6 (software exception). Hardware exceptions comprise all exceptions except the following:
 - Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)

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*.

- 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).¹ 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 27.2.3. The value of this bit is undefined if the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- VM-exit interruption error code.
 - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally through the IDT. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the IDT-vectoring information field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).
 - For other VM exits, the value of this field is undefined.

27.2.3 Information About NMI Unblocking Due to IRET

A VM exit may occur during execution of the IRET instruction for reasons including the following: faults, EPT violations, page-modification log-full events, or SPP-related events.

An execution of IRET that commences while non-maskable interrupts (NMIs) are blocked will unblock NMIs even if a fault or VM exit occurs; the state saved by such a VM exit will indicate that NMIs were not blocked.

VM exits for the reasons enumerated above provide more information to software by saving a bit called “NMI unblocking due to IRET.” This bit is defined if (1) either the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1; (2) the VM exit does not set the valid bit in the IDT-vectoring information field (see Section 27.2.4); 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 24-3) 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 27.2.2). For VM exits due to EPT violations, page-modification log-full events, and SPP-related events, NMI unblocking due to IRET is saved in bit 12 of the exit qualification (Section 27.2.1).

1. 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.

(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.)

27.2.4 Information for VM Exits During Event Delivery

Section 24.9.3 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 25.4.2).
- Event delivery causes an APIC-access VM exit (see Section 29.4).
- An EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that occurs during event delivery.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 26.6.1.2).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the “NMI exiting” VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.

The following items detail the use of these fields:

- IDT-vectoring information (format given in Table 24-19). The following items detail how this field is established for VM exits that occur during event delivery:
 - If the VM exit occurred during delivery of an exception, bits 7:0 receive the exception vector (at most 31). If the VM exit occurred during delivery of an NMI, bits 7:0 are set to 2. If the VM exit occurred during delivery of an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to indicate the type of event that was being delivered when the VM exit occurred: 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 4 (software interrupt), 5 (privileged software interrupt), or 6 (software exception).

Hardware exceptions comprise all exceptions except the following:²

- 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).³ If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).

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).

2. 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*.

- Bit 12 is undefined.
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- IDT-vectoring error code.
 - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
 - For other VM exits, the value of this field is undefined.

27.2.5 Information for VM Exits Due to Instruction Execution

Section 24.9.4 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 25.1.2) or based on the settings of VM-execution controls (see Section 25.1.3): CLTS, CPUID, ENCLS, GETSEC, HLT, IN, INS, INVVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LOADIWKEY, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, PCONFIG, RDMSR, RDPIC, 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 to executions of VMFUNC that fail because one of the following is true:
 - EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 25.5.6.2).

3. 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.

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 29.4.6) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.

- EAX = 0 and either ECX \geq 512 or the value of ECX selects an invalid tentative EPTP value (see Section 25.5.6.3).

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 26.6.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

If the VM exit occurred in enclave mode, this field is cleared (none of the previous items apply).

Table 27-8. 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, 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 27-8.¹
 - For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 27-9.
 - For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 27-10.
 - For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 27-11.
 - For VM exits due to attempts to execute RDRAND, RDSEED, TPAUSE, or UMWAIT, the field has the format is given in Table 27-12.
 - For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 27-13.

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 27-8 is used by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

- For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 27-14.
- For VM exits due to attempts to execute LOADIWKEY, the field has the format is given in Table 27-15.

For all other VM exits, the field is undefined, unless the VM exit occurred in enclave mode, in which case the field is cleared.

- **I/O RCX, I/O RSI, I/O RDI, I/O RIP.** These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 31.15.2.3. Note that, if the VM exit occurred in enclave mode, these fields are all cleared.

Table 27-9. 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 27-10. 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 27-10. 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 27-11. 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 27-11. 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 27-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND, RDSEED, TPAUSE, and UMWAIT

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (destination for RDRAND and RDSEED; source for TPAUSE and UMWAIT): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture)
10:7	Undefined.
12:11	Operand size: 0: 16-bit 1: 32-bit 2: 64-bit The value 3 is not used.
31:13	Undefined.

Table 27-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.

Table 27-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES (Contd.)

Bit Position(s)	Content
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Undefined.

Table 27-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.

Table 27-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE (Contd.)

Bit Position(s)	Content
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above)

Table 27-15. Format of the VM-Exit Instruction-Information Field as Used for LOADIWKEY

Bit Position(s)	Content
2:0	Undefined.
6:3	Reg1: identifies the first XMM register operand (XMM0-XMM15; values 8-15 are used only on processors that support Intel 64 architecture).
30:7	Undefined.
31:28	Reg2: identifies the second XMM register operand (see above).

27.3 SAVING GUEST STATE

VM exits save certain components of processor state into corresponding fields in the guest-state area of the VMCS (see Section 24.4). On processors that support Intel 64 architecture, the full value of each natural-width field (see Section 24.11.2) is saved regardless of the mode of the logical processor before and after the VM exit.

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 27.1 for a discussion of which architectural updates occur at that time.

Section 27.3.1 through Section 27.3.4 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.

27.3.1 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 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 31.15.2.

27.3.2 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 24.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
 - CS.
 - The base-address and segment-limit fields are saved.

- The L, D, and G bits are saved in the access-rights field.
- SS.
 - DPL is saved in the access-rights field.
 - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
- DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
- FS and GS. The base-address field is saved.
- LDTR. The value saved for the base address is always canonical.
- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.
- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

27.3.3 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 27.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,¹ or into the old task-state segment had the event been delivered through a task gate).
 - If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.
 - If the VM exit is due to a software exception (due to an execution of INT3 or INTO) or a privileged software exception (due to an execution of INT1), the value saved references the INT3, INTO, or INT1 instruction that caused that exception.
 - Suppose that the VM exit is due to a task switch that was caused by execution of CALL, IRET, or JMP or by execution of a software interrupt (INT *n*), software exception (due to execution of INT3 or INTO), or privileged software exception (due to execution of INT1) that encountered a task gate in the IDT. The value saved references the instruction that caused the task switch (CALL, IRET, JMP, INT *n*, INT3, INTO, INT1).
 - Suppose that the VM exit is due to a task switch that was caused by a task gate in the IDT that was encountered for any reason except the direct access by a software interrupt or software exception. The

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.

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 29.1.1) below that of TPR threshold VM-execution control field (see Section 29.1.2), 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 29.4.3.2) 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 gate¹ or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.
 - If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
 - If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
 - If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.²
 - For APIC-access VM exits and for VM exits caused by EPT violations, EPT misconfigurations, page-modification log-full events, or SPP-related events, the value saved depends on whether the VM exit occurred during delivery of an event through the IDT:
 - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 27.2.4), 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.

27.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

- The activity-state field is saved with the logical processor’s activity state before the VM exit.³ See Section 27.1 for details of how events leading to a VM exit may affect the activity state.

1. The reference here is to the full value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.

2. This is true even if RFLAGS.RF was 1 before the instruction was executed. If, in response to such a VM exit, a VM monitor re-enters the guest to re-execute the instruction that caused the VM exit (for example, after clearing the VM-execution control that caused the VM exit), the instruction may encounter a code breakpoint that has already been processed. A VM monitor can avoid this by setting the guest value of RFLAGS.RF to 1 before resuming guest software.

3. If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

- The interruptibility-state field is saved to reflect the logical processor's interruptibility before the VM exit.
 - See Section 27.1 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",¹ "virtualized EOI", "APIC write", or "monitor trap flag."
 - A VM exit due to trace-address pre-translation (TAPT; see Section 25.5.4) or due to accesses related to PEBS on processors with the "EPT-friendly" enhancement (see Section 19.9.5). 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 26.7.3). 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 26.7.3) 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.")

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.

1. This item includes VM exits that occur as a result of certain VM entries (Section 26.7.7).

- 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 26.7.3). 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 26.7.3) 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 25.5.1). 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.

27.4 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 24.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMBASE is an MSR that can be read only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if

1. A logical processor uses PAE paging if CRO.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 24.6.2.

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 27.7.

27.5 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 27.5.1 to Section 27.5.5. 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 27.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 27.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 27.6). This loading occurs only after the state loading described in this section.

27.5.1 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 23.8).¹
 - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor’s physical-address width (they are cleared to 0).² (This item applies only to processors that support Intel 64 architecture.)
 - For CR4, any bits that are fixed in VMX operation (see Section 23.8).
 - 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.
- The following MSRs are established as follows:

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.

- 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 27.5.2).
 - 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 27.6.

27.5.2 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 Section 26.3.1.2 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.

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.

- 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 FFFFFFFH and a G-bit setting of 1).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.
 - TR. Set to 00000067H.
- The type field and S bit are set as follows:
 - CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
 - TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:
 - CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
 - CS, TR. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32_EFER.LMA (see Section 27.5.1), no VM exit is ever to compatibility mode (which requires IA32_EFER.LMA = 1 and CS.L = 0).
- D/B.
 - CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
 - SS. Set to 1.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.
- G.
 - CS. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.

The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined (although the base address is always canonical).

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N of each base address is set to the value of bit $N-1$ of that base address. The GDTR and IDTR limits are each set to FFFFH.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

27.5.3 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.

27.5.4 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 27.7). 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.

27.5.5 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 24-3). Other VM exits do not affect blocking by NMI. (See Section 27.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 28.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM exits invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).
- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

27.5.6 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 8.10.4 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.

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.

27.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 24.7.2). 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 C000100H (the IA32_FS_BASE MSR) or C000101H (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 27.7.

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.

27.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shutdown 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 24.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 27.4).
2. Host checking of the page-directory-pointer-table entries (PDPTes) failed (see Section 27.5.4).
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 27.6).
5. There was a machine-check event during VM exit (see Section 27.8).
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 27.5).

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 27.5) 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

1. Note the following about processors that support Intel 64 architecture. If CRO.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CRO.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:¹

- If the logical processor is in SMX operation, an Intel[®] TXT shutdown condition occurs. The error code used is 000DH, indicating “VMX abort.” See *Intel[®] Trusted Execution Technology Measured Launched Environment Programming Guide*.
- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the **VMX-abort shutdown state**. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine-check events; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

27.8 MACHINE-CHECK EVENTS DURING VM EXIT

If a machine-check event occurs during VM exit, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM exit:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:²
 - If the logical processor is in SMX operation, an Intel[®] TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- The machine-check event is handled after VM exit completes:
 - If the VM exit ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel[®] TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM exit ends with CR4.MCE = 1, a machine-check exception (#MC) is delivered through the host IDT.
- A VMX abort is generated (see Section 27.7). 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.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel[®] 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

2. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel[®] 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

15. Updates to Chapter 34, Volume 3D

Changes to Chapter 34 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter: Update to Table 34-2, "Layout of SGX Enclave Control Structure (SECS)", to correct typos in offset and size of the first reserved field. Update to Table 34-23, "Layout of REPORT", to correct typos in offset and size of the first reserved field.

CHAPTER 34

ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

34.1 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 33.3). 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 34.4 and Section 34.5 on such accesses.

34.2 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 34-1 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.

34.3 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 34.4 and Section 34.5 on such accesses.
- EPC pages of page types PT_REG, PT_TCS and PT_TRIM must be mapped to ELRANGE at the linear address specified when the EPC page was allocated to the enclave using ENCLS[EADD] or ENCLS[EAUG] leaf functions. Enclave accesses through other linear address result in a #PF with the PFEC.SGX bit set.

- Direct EAs to any EPC pages must conform to the currently defined security attributes for that EPC page in the EPCM. These attributes may be defined at enclave creation time (EADD) or when the enclave sets them using SGX2 instructions. The failure of these checks results in a #PF with the PFEC.SGX bit set.
 - Target page must belong to the currently executing enclave.
 - Data may be written to an EPC page if the EPCM allow write access.
 - Data may be read from an EPC page if the EPCM allow read access.
 - Instruction fetches from an EPC page are allowed if the EPCM allows execute access.
 - Shadow-stack-load from an EPC page and shadow-stack-store to an EPC page are allowed only if the page type is PT_SS_FIRST or PT_SS_REST.
 - Data writes that are not shadow-stack-store are not allowed if the EPCM page type is PT_SS_FIRST or PT_SS_REST.
 - Target page must not have a restricted page type¹ (PT_SECS, PT_TCS, PT_VA, or PT_TRIM).
 - The EPC page must not be BLOCKED.
 - The EPC page must not be PENDING.
 - The EPC page must not be MODIFIED.

34.4 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.

34.5 PAGE-BASED ACCESS CONTROL

34.5.1 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 34.3. 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.

34.5.2 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.

34.5.3 Implicit vs. Explicit Accesses

Memory accesses originating from Intel SGX instruction leaf functions are categorized as either explicit accesses or implicit accesses. Table 34-1 lists the implicit and explicit memory accesses made by Intel SGX leaf functions.

34.5.3.1 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 39.3.4.

34.5.3.2 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 34-1 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 34-1 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 36.

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 34-1. 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 34-1. 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 36.4

34.6 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 34.7 through Section 34.20.

34.7 SGX ENCLAVE CONTROL STRUCTURE (SECS)

The SECS data structure requires 4K-Bytes alignment.

Table 34-2. 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 34.7.2) 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 34-5. 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 34-3.
MRENCLAVE	64	32	Measurement Register of enclave build process. See SIGSTRUCT for format.
RESERVED	96	32	
MRSIGNER	128	32	Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for format.
RESERVED	160	32	
CONFIGID	192	64	Post EINIT configuration identity.
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Post EINIT configuration security version number (SVN).
RESERVED	262	3834	<p>The RESERVED field consists of the following:</p> <ul style="list-style-type: none"> ▪ EID: An 8 byte Enclave Identifier. Its location is implementation specific. ▪ PAD: A 352 bytes padding pattern from the Signature (used for key derivation strings). It's location is implementation specific. ▪ VIRTCHILDCNT: An 8 byte Count of virtual children that have been paged out by a VMM. Its location is implementation specific. ▪ ENCLAVECONTEXT: An 8 byte Enclave context pointer. Its location is implementation specific. ▪ ISVFAMILYID: A 16 byte value assigned to identify the family of products the enclave belongs to. ▪ ISVEXTPRODID: A 16 byte value assigned to identify the product identity of the enclave. ▪ The remaining 3226 bytes are reserved area. <p>The entire 3834 byte field must be cleared prior to executing ECREATE.</p>

34.7.1 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 34-3. Layout of ATTRIBUTES Structure

Field	Bit Position	Description
INIT	0	This bit indicates if the enclave has been initialized by EINIT. It must be cleared when loaded as part of ECREATE. For EREPORT instruction, TARGET_INFO.ATTRIBUTES[ENIT] must always be 1 to match the state after EINIT has initialized the enclave.
DEBUG	1	If 1, the enclave permit debugger to read and write enclave data using EDBGD and EDBGWR.
MODE64BIT	2	Enclave runs in 64-bit mode.
RESERVED	3	Must be Zero.
PROVISIONKEY	4	Provisioning Key is available from EGETKEY.
EINITTOKEN_KEY	5	EINIT token key is available from EGETKEY.
CET	6	Enable CET attributes. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0 this bit is reserved and must be 0.
KSS	7	Key Separation and Sharing Enabled.
RESERVED	63:8	Must be zero.
XFRM	127:64	XSAVE Feature Request Mask. See Section 38.7.

34.7.2 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 34-4.

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 34.9.2.

Table 34-4. 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).

34.7.3 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 34-5. 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).

34.8 THREAD CONTROL STRUCTURE (TCS)

Each executing thread in the enclave is associated with a Thread Control Structure. It requires 4K-Bytes alignment.

Table 34-6. 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 processor is currently executing an enclave in the context of this TCS.
FLAGS	8	8	The thread's execution flags (see Section 34.8.1).
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.

34.8.1 TCS.FLAGS

Table 34-7. 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.
RESERVED	63:1	

34.8.2 State Save Area Offset (OSSA)

The OSSA points to a stack of State Save Area (SSA) frames (see Section 34.9) used to save the processor state when an interrupt or exception occurs while executing in the enclave.

34.8.3 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.

34.8.4 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.

34.9 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 34-8). 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 34.9.2.

Table 34-8. 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.

Table 34-8. Top-to-Bottom Layout of an SSA Frame

Region	Offset (Byte)	Size (Bytes)	Description
MISC	base of GPRSGX - sizeof(MISC)	Calculate from high- est set bit of SECS.MISCSELECT	See Section 34.9.2.
GPRSGX	SSAFRAMESIZE - 176	176	See Table 34-9 for layout of the GPRSGX region.

34.9.1 GPRSGX Region

The layout of the GPRSGX region is shown in Table 34-9.

Table 34-9. 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 34.9.1.1).
RESERVED	164	4	
FSBASE	168	8	FS BASE.
GSBASE	176	8	GS BASE.

34.9.1.1 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 34-10. The VALID bit is set only for the exceptions conditions which are reported inside an enclave. See Table 34-11 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 34-10. Layout of EXITINFO Field

Field	Bit Position	Description
VECTOR	7:0	Exception number of exceptions reported inside enclave.
EXIT_TYPE	10:8	011b: Hardware exceptions. 110b: Software exceptions. Other values: Reserved.
RESERVED	30:11	Reserved as zero.
VALID	31	0: unsupported exceptions. 1: Supported exceptions. Includes two categories: <ul style="list-style-type: none"> • Unconditionally supported exceptions: #DE, #DB, #BP, #BR, #UD, #MF, #AC, #XM. • Conditionally supported exception: <ul style="list-style-type: none"> – #PF, #GP if SECS.MISCSELECT.EXINFO = 1. – #CP if SECS.MISCSELECT.CPINFO=1.

34.9.1.2 VECTOR Field Definition

Table 34-11 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 34-11. 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.
#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.

34.9.2 MISC Region

The layout of the MISC region is shown in Table 34-12. 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 34-12. 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 34-12. 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 34-12. 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).

34.9.2.1 EXINFO Structure

Table 34-13 contains the layout of the EXINFO structure that provides additional information.

Table 34-13. 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	

34.9.2.2 Page Fault Error Code

Table 34-14 contains page fault error code that may be reported in EXINFO.ERRCD.

Table 34-14. Page Fault Error Code

Name	Bit Position	Description
P	0	Same as non-SGX page fault exception P flag.
W/R	1	Same as non-SGX page fault exception W/R flag.
U/S ¹	2	Always set to 1 (user mode reference).
RSVD	3	Same as non-SGX page fault exception RSVD flag.
I/D	4	Same as non-SGX page fault exception I/D flag.
PK	5	Protection Key induced fault.
RSVD	14:6	Reserved.
SGX	15	EPCM induced fault.
RSVD	31:5	Reserved.

NOTES:

1. Page faults incident to enclave mode that report U/S=0 are not reported in EXINFO.

34.10 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 34-15. Layout of CET State Save Area Frame

Field	Offset (Bytes)	Size (Bytes)	Description
SSP	0	8	Shadow Stack Pointer. This field is reserved when CPUID.(EAX=7, ECX=0):ECX[CET_SS] is 0.
IB_TRACK_STATE	8	8	Indirect branch tracker state: Bit 0: SUPPRESS - suppressed(1), tracking(0) Bit 1: TRACKER - IDLE (0), WAIT_FOR_ENDBRANCH (1) Bits 63:2 - Reserved This field is reserved when CPUID.(EAX=7, ECX=0):EDX[CET_IBT] is 0.

34.11 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 34-16. Layout of PAGEINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
LINADDR	0	8	Enclave linear address.
SRCPGE	8	8	Effective address of the page where contents are located.
SECINFO/PCMD	16	8	Effective address of the SECINFO or PCMD (for ELDU, ELDB, EWB) structure for the page.
SECS	24	8	Effective address of EPC slot that currently contains the SECS.

34.12 SECURITY INFORMATION (SECINFO)

The SECINFO data structure holds meta-data about an enclave page.

Table 34-17. Layout of SECINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
FLAGS	0	8	Flags describing the state of the enclave page.
RESERVED	8	56	Must be zero.

34.12.1 SECINFO.FLAGS

The SECINFO.FLAGS are a set of fields describing the properties of an enclave page.

Table 34-18. Layout of SECINFO.FLAGS Field

Field	Bit Position	Description
R	0	If 1 indicates that the page can be read from inside the enclave; otherwise the page cannot be read from inside the enclave.
W	1	If 1 indicates that the page can be written from inside the enclave; otherwise the page cannot be written from inside the enclave.
X	2	If 1 indicates that the page can be executed from inside the enclave; otherwise the page cannot be executed from inside the enclave.
PENDING	3	If 1 indicates that the page is in the PENDING state; otherwise the page is not in the PENDING state.
MODIFIED	4	If 1 indicates that the page is in the MODIFIED state; otherwise the page is not in the MODIFIED state.
PR	5	If 1 indicates that a permission restriction operation on the page is in progress, otherwise a permission restriction operation is not in progress.
RESERVED	7:6	Must be zero.
PAGE_TYPE	15:8	The type of page that the SECINFO is associated with.
RESERVED	63:16	Must be zero.

34.12.2 PAGE_TYPE Field Definition

The SECINFO flags and EPC flags contain bits indicating the type of page.

Table 34-19. Supported PAGE_TYPE

TYPE	Value	Description
PT_SECS	0	Page is an SECS.
PT_TCS	1	Page is a TCS.
PT_REG	2	Page is a regular page.
PT_VA	3	Page is a Version Array.
PT_TRIM	4	Page is in trimmed state.
PT_SS_FIRST	5	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, Page is first page of a shadow stack. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this value is reserved.
PT_SS_REST	6	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, Page is not first page of a shadow stack. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this value is reserved.
	All others	Reserved.

34.13 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 34-20. Layout of PCMD Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
SECINFO	0	64	Flags describing the state of the enclave page; R/W by software.
ENCLAVEID	64	8	Enclave Identifier used to establish a cryptographic binding between paged-out page and the enclave.
RESERVED	72	40	Must be zero.
MAC	112	16	Message Authentication Code for the page, page meta-data and reserved field.

34.14 ENCLAVE SIGNATURE STRUCTURE (SIGSTRUCT)

SIGSTRUCT is a structure created and signed by the enclave developer that contains information about the enclave. SIGSTRUCT is processed by the EINIT leaf function to verify that the enclave was properly built.

SIGSTRUCT includes ENCLAVEHASH as SHA256 digest, as defined in FIPS PUB 180-4. The digests are byte strings of length 32. Each of the 8 HASH dwords is stored in little-endian order.

SIGSTRUCT includes four 3072-bit integers (MODULUS, SIGNATURE, Q1, Q2). Each such integer is represented as a byte strings of length 384, with the most significant byte at the position "offset + 383", and the least significant byte at position "offset".

The (3072-bit integer) SIGNATURE should be an RSA signature, where: a) the RSA modulus (MODULUS) is a 3072-bit integer; b) the public exponent is set to 3; c) the signing procedure uses the EMSA-PKCS1-v1.5 format with DER encoding of the "DigestInfo" value as specified in of PKCS#1 v2.1/RFC 3447.

The 3072-bit integers Q1 and Q2 are defined by:

$$q1 = \text{floor}(\text{Signature}^2 / \text{Modulus});$$

$$q2 = \text{floor}((\text{Signature}^3 - q1 * \text{Signature} * \text{Modulus}) / \text{Modulus});$$

SIGSTRUCT must be page aligned

In column 5 of Table 34-21, 'Y' indicates that this field should be included in the signature generated by the developer.

Table 34-21. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
HEADER	0	16	Must be byte stream 06000000E100000000001000000000H	Y
VENDOR	16	4	Intel Enclave: 00008086H Non-Intel Enclave: 00000000H	Y
DATE	20	4	Build date is yyyyymmdd in hex: yyyy=4 digit year, mm=1-12, dd=1-31	Y
HEADER2	24	16	Must be byte stream 010100006000000006000000001000000H	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*	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 34-21. Layout of Enclave Signature Structure (SIGSTRUCT)

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
<p>* If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISCSELECT must be 0. If CPUID.(EAX=12H, ECX=0):EBX[31:0] !=0, enclave writers must specify MISCSELECT such that each cleared bit in MISCMASK must also specify the corresponding bit as 0 in MISCSELECT.</p>				

34.15 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 34-22. Layout of EINIT Token (EINITTOKEN)

Field	OFFSET (Bytes)	Size (Bytes)	MACed	Description
Valid	0	4	Y	Bit 0: 1: Valid; 0: Invalid. All other bits reserved.
RESERVED	4	44	Y	Must be zero.
ATTRIBUTES	48	16	Y	ATTRIBUTES of the Enclave.
MRENCLAVE	64	32	Y	MRENCLAVE of the Enclave.
RESERVED	96	32	Y	Reserved.
MRSIGNER	128	32	Y	MRSIGNER of the Enclave.
RESERVED	160	32	Y	Reserved.
CPUSVNLE	192	16	N	Launch Enclave's CPUSVN.
ISVPRODIDLE	208	02	N	Launch Enclave's ISVPRODID.
ISVSVNLE	210	02	N	Launch Enclave's ISVSVN.
CET_MASKED_ATTRIBUTES_LE	212	1	N	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Launch enclaves masked CET attributes. This should be set to LE's CET_ATTRIBUTES masked with CET_ATTRIBUTES_MASK of the LE's KEYREQUEST. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved.
RESERVED	213	23	N	Reserved.
MASKEDMISCSELECTLE	236	4		Launch Enclave's MASKEDMISCSELECT: set by the LE to the resolved MISCSELECT value, used by EGETKEY (after applying KEYREQUEST's masking).
MASKEDATTRIBUTESLE	240	16	N	Launch Enclave's MASKEDATTRIBUTES: This should be set to the LE's ATTRIBUTES masked with ATTRIBUTEMASK of the LE's KEYREQUEST.
KEYID	256	32	N	Value for key wear-out protection.
MAC	288	16	N	Message Authentication Code on EINITTOKEN using EINITTOKEN_KEY.

34.16 REPORT (REPORT)

The REPORT structure is the output of the EREPORT instruction, and must be 512-Byte aligned.

Table 34-23. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CPUSVN	0	16	The security version number of the processor.
MISCSELECT	16	4	Bit vector specifying which extended features are saved to the MISC region of the SSA frame when an AEX occurs.
CET_ATTRIBUTES	20	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field reports the CET_ATTRIBUTES of the Enclave. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.
RESERVED	21	11	Zero.
ISVEXTNPRODID	32	16	The value of SECS.ISVEXTPRODID.
ATTRIBUTES	48	16	ATTRIBUTES of the Enclave. See Section 34.7.1.
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 34-23. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CONFIGID	192	64	Value provided by SW to identify enclave's post EINIT configuration.
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Value provided by SW to indicate expected SVN of enclave's post EINIT configuration.
RESERVED	262	42	Zero.
ISVFAMILYID	304	16	The value of SECS.ISVFAMILYID.
REPORTDATA	320	64	Data provided by the user and protected by the REPORT's MAC, see Section 34.16.1.
KEYID	384	32	Value for key wear-out protection.
MAC	416	16	Message Authentication Code on the report using report key.

34.16.1 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.

34.17 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 34-24. Layout of TARGETINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MEASUREMENT	0	32	The MRENCLAVE of the target enclave.
ATTRIBUTES	32	16	The ATTRIBUTES field of the target enclave.
CET_ATTRIBUTES	48	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the CET_ATTRIBUTES field of the target enclave. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved.
RESERVED	49	1	Must be zero.
CONFIGSVN	50	2	CONFIGSVN of the target enclave.
MISCSELECT	52	4	The MISCSELECT of the target enclave.
RESERVED	56	8	Must be zero.
CONFIGID	64	64	CONFIGID of target enclave.
RESERVED	128	384	Must be zero.

34.18 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 34-25. Layout of KEYREQUEST Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	2	Identifies the Key Required.
KEYPOLICY	2	2	Identifies which inputs are required to be used in the key derivation.
ISVSVN	4	2	The ISV security version number that will be used in the key derivation.
CET_ATTRIBUTES_MASK	6	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides a mask that defines which CET_ATTRIBUTES bits will be included in key derivation. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, then this field is reserved and must be 0.
RESERVED	7	1	Must be zero.
CPUSVN	8	16	The security version number of the processor used in the key derivation.
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in key derivation.
KEYID	40	32	Value for key wear-out protection.
MISCMASK	72	4	A mask defining which MISCSELECT bits will be included in key derivation.
CONFIGSVN	76	2	Identifies which enclave Configuration's Security Version should be used in key derivation.
RESERVED	78	434	

34.18.1 KEY REQUEST KeyNames

Table 34-26. Supported KEYName Values

Key Name	Value	Description
EINIT_TOKEN_KEY	0	EINIT_TOKEN key
PROVISION_KEY	1	Provisioning Key
PROVISION_SEAL_KEY	2	Provisioning Seal Key
REPORT_KEY	3	Report Key
SEAL_KEY	4	Seal Key
	All others	Reserved

34.18.2 Key Request Policy Structure

Table 34-27. Layout of KEYPOLICY Field

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register.
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register.
NOISVPRODID	2	If 1, derive key WITHOUT using the enclave' ISVPRODID value.
CONFIGID	3	If 1, derive key using the enclave's CONFIGID value.
ISVFAMILYID	4	If 1, derive key using the enclave ISVFAMILYID value.
ISVEXTPRODID	5	If 1, derive key using enclave's ISVEXTPRODID value.
RESERVED	15:6	Must be zero.

34.19 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 34-28. Layout of Version Array Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
Slot 0	0	8	Version Slot 0
Slot 1	8	8	Version Slot 1
...			
Slot 511	4088	8	Version Slot 511

34.20 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 34-29. Content of an Enclave Page Cache Map Entry

Field	Description
VALID	Indicates whether the EPCM entry is valid.
R	Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.
W	Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.
X	Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.
PT	EPCM page type (PT_SECS, PT_TCS, PT_REG, PT_VA, PT_TRIM, PT_SS_FIRST, PT_SS_REST).
ENCLAVESECS	SECS identifier of the enclave to which the EPC page belongs.
ENCLAVEADDRESS	Linear enclave address of the EPC page.
BLOCKED	Indicates whether the EPC page is in the blocked state.
PENDING	Indicates whether the EPC page is in the pending state.
MODIFIED	Indicates whether the EPC page is in the modified state.
PR	Indicates whether the EPC page is in a permission restriction state.

34.21 READ INFO (RDINFO)

The RDINFO structure contains status information about an EPC page. It must be aligned to 32-Bytes.

Table 34-30. Layout of RDINFO Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
STATUS	0	8	Page status information.
FLAGS	8	8	EPCM state of the page.
ENCLAVECONTEXT	16	8	Context pointer describing the page's parent location.

34.21.1 RDINFO Status Structure

Table 34-31. Layout of RDINFO STATUS Structure

Field	Bit Position	Description
CHILDPRESENT	0	Indicates that the page has one or more child pages present (always zero for non-SECS pages). In VMX non-root operation includes the presence of virtual children.
VIRTCHLDPRESENT	1	Indicates that the page has one or more virtual child pages present (always zero for non-SECS pages). In VMX non-root operation this value is always zero.
RESERVED	63:2	

34.21.2 RDINFO Flags Structure

Table 34-32. Layout of RDINFO FLAGS Structure

Field	Bit Position	Description
R	0	Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.
W	1	Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.
X	2	Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.
PENDING	3	Indicates whether the EPC page is in the pending state.
MODIFIED	4	Indicates whether the EPC page is in the modified state.
PR	5	Indicates whether the EPC page is in a permission restriction state.
RESERVED	7:6	
PAGE_TYPE	15:8	Indicates the page type of the EPC page.
RESERVED	62:16	
BLOCKED	63	Indicates whether the EPC page is in the blocked state.

16. Updates to Appendix B, Volume 3D

Change bars and green text show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter: Updated Table B-4, "Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)" with PCONFIG-exiting bitmap information.

APPENDIX B

FIELD ENCODING IN VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 24.11.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.)

B.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 24.11.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.1.1 16-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-1 enumerates the 16-bit control fields.

Table B-1. Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID) ¹	00000000B	00000000H
Posted-interrupt notification vector ²	00000001B	00000002H
EPTP index ³	00000010B	00000004H
HLAT prefix size ⁴	00000011B	00000006H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable VPID” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES selector	00000000B	0000800H
Guest CS selector	00000001B	0000802H
Guest SS selector	00000010B	0000804H
Guest DS selector	00000011B	0000806H
Guest FS selector	00000100B	0000808H
Guest GS selector	00000101B	000080AH
Guest LDTR selector	00000110B	000080CH

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest TR selector	000000111B	0000080EH
Guest interrupt status ¹	000001000B	00000810H
PML index ²	000001001B	00000812H

NOTES:

1. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.

B.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-3 enumerates the 16-bit host-state fields.

Table B-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host ES selector	000000000B	00000C00H
Host CS selector	000000001B	00000C02H
Host SS selector	000000010B	00000C04H
Host DS selector	000000011B	00000C06H
Host FS selector	000000100B	00000C08H
Host GS selector	000000101B	00000C0AH
Host TR selector	000000110B	00000C0CH

B.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 24.11.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)		00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)		00002003H
Address of MSR bitmaps (full) ¹	000000010B	00002004H
Address of MSR bitmaps (high) ¹		00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)		00002007H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)		00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)		0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)		0000200DH
PML address (full) ²	000000111B	0000200EH
PML address (high) ²		0000200FH
TSC offset (full)	000001000B	00002010H
TSC offset (high)		00002011H
Virtual-APIC address (full) ³	000001001B	00002012H
Virtual-APIC address (high) ³		00002013H
APIC-access address (full) ⁴	000001010B	00002014H
APIC-access address (high) ⁴		00002015H
Posted-interrupt descriptor address (full) ⁵	000001011B	00002016H
Posted-interrupt descriptor address (high) ⁵		00002017H
VM-function controls (full) ⁶	000001100B	00002018H
VM-function controls (high) ⁶		00002019H
EPT pointer (EPTP; full) ⁷	000001101B	0000201AH
EPT pointer (EPTP; high) ⁷		0000201BH
EOI-exit bitmap 0 (EOI_EXIT0; full) ⁸	000001110B	0000201CH
EOI-exit bitmap 0 (EOI_EXIT0; high) ⁸		0000201DH
EOI-exit bitmap 1 (EOI_EXIT1; full) ⁸	000001111B	0000201EH
EOI-exit bitmap 1 (EOI_EXIT1; high) ⁸		0000201FH
EOI-exit bitmap 2 (EOI_EXIT2; full) ⁸	000010000B	00002020H
EOI-exit bitmap 2 (EOI_EXIT2; high) ⁸		00002021H
EOI-exit bitmap 3 (EOI_EXIT3; full) ⁸	000010001B	00002022H
EOI-exit bitmap 3 (EOI_EXIT3; high) ⁸		00002023H
EPTP-list address (full) ⁹	000010010B	00002024H
EPTP-list address (high) ⁹		00002025H
VMREAD-bitmap address (full) ¹⁰	000010011B	00002026H
VMREAD-bitmap address (high) ¹⁰		00002027H
VMWRITE-bitmap address (full) ¹⁰	000010100B	00002028H
VMWRITE-bitmap address (high) ¹⁰		00002029H
Virtualization-exception information address (full) ¹¹	000010101B	0000202AH
Virtualization-exception information address (high) ¹¹		0000202BH
XSS-exiting bitmap (full) ¹²	000010110B	0000202CH
XSS-exiting bitmap (high) ¹²		0000202DH

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
ENCLS-exiting bitmap (full) ¹³	000010111B	0000202EH
ENCLS-exiting bitmap (high) ¹³		0000202FH
Sub-page-permission-table pointer (full) ¹⁴	000011000B	00002030H
Sub-page-permission-table pointer (high) ¹⁴		00002031H
TSC multiplier (full) ¹⁵	000011001B	00002032H
TSC multiplier (high) ¹⁵		00002033H
Tertiary processor-based VM-execution controls (full) ¹⁶	000011010B	00002034H
Tertiary processor-based VM-execution controls (high) ¹⁶		00002035H
ENCLV-exiting bitmap (full) ¹⁷	000011011B	00002036H
ENCLV-exiting bitmap (high) ¹⁷		00002037H
PCONFIG-exiting bitmap (full) ¹⁸	000011111B	0000203EH
PCONFIG-exiting bitmap (high) ¹⁸		0000203FH
Hypervisor-managed linear-address translation pointer (HLATP; full) ¹⁹	000100000B	00002040H
HLATP (high) ¹⁹		00002041H
Secondary VM-exit controls (full) ²⁰	000100010B	00002044H
Secondary VM-exit controls (high) ²⁰		00002045H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use MSR bitmaps” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
6. This field exists only on processors that support the 1-setting of the “enable VM functions” VM-execution control.
7. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
8. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
9. This field exists only on processors that support the 1-setting of the “EPTP switching” VM-function control.
10. This field exists only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control.
11. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
12. This field exists only on processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control.
13. This field exists only on processors that support the 1-setting of the “enable ENCLS exiting” VM-execution control.
14. This field exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.
15. This field exists only on processors that support the 1-setting of the “use TSC scaling” VM-execution control.
16. This field exists only on processors that support the 1-setting of the “activate tertiary controls” VM-execution control.
17. This field exists only on processors that support the 1-setting of the “enable ENCLV exiting” VM-execution control.
18. This field exists only on processors that support the 1-setting of the “enable PCONFIG” VM-execution control.
19. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.
20. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-exit control.

B.2.2 64-Bit Read-Only Data Field

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. There is only one such 64-bit field as given in Table B-5. (As with other 64-bit fields, this one

has two encodings.)

Table B-5. Encodings for 64-Bit Read-Only Data Field (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full) ¹	000000000B	00002400H
Guest-physical address (high) ¹		00002401H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)		00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H
Guest IA32_DEBUGCTL (high)		00002803H
Guest IA32_PAT (full) ¹	000000010B	00002804H
Guest IA32_PAT (high) ¹		00002805H
Guest IA32_EFER (full) ²	000000011B	00002806H
Guest IA32_EFER (high) ²		00002807H
Guest IA32_PERF_GLOBAL_CTRL (full) ³	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high) ³		00002809H
Guest PDPTE0 (full) ⁴	000000101B	0000280AH
Guest PDPTE0 (high) ⁴		0000280BH
Guest PDPTE1 (full) ⁴	000000110B	0000280CH
Guest PDPTE1 (high) ⁴		0000280DH
Guest PDPTE2 (full) ⁴	000000111B	0000280EH
Guest PDPTE2 (high) ⁴		0000280FH
Guest PDPTE3 (full) ⁴	000001000B	00002810H
Guest PDPTE3 (high) ⁴		00002811H
Guest IA32_BNDCFGS (full) ⁵	000001001B	00002812H
Guest IA32_BNDCFGS (high) ⁵		00002813H
Guest IA32_RTIT_CTL (full) ⁶	000001010B	00002814H
Guest IA32_RTIT_CTL (high) ⁶		00002815H
Guest IA32_LBR_CTL (full) ⁷	000001011B	00002816H
Guest IA32_LBR_CTL (high) ⁷		00002817H
Guest IA32_PKRS (full) ⁸	000001100B	00002818H
Guest IA32_PKRS (high) ⁸		00002819H

NOTES:

1. This field exists only on processors that support either the 1-setting of the “load IA32_PAT” VM-entry control or that of the “save IA32_PAT” VM-exit control.
2. This field exists only on processors that support either the 1-setting of the “load IA32_EFER” VM-entry control or that of the “save IA32_EFER” VM-exit control.
3. This field exists only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.
4. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
5. This field exists only on processors that support either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control.
6. This field exists only on processors that support either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control.
7. This field exists only on processors that support either the 1-setting of the “load IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control.
8. This field exists only on processors that support the 1-setting of the “load PKRS” VM-entry control.

B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full) ¹	000000000B	00002C00H
Host IA32_PAT (high) ¹		00002C01H
Host IA32_EFER (full) ²	000000001B	00002C02H
Host IA32_EFER (high) ²		00002C03H
Host IA32_PERF_GLOBAL_CTRL (full) ³	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high) ³		00002C05H
Host IA32_PKRS (full) ⁴	000000011B	00002C06H
Host IA32_PKRS (high) ⁴		00002C07H

NOTES:

1. This field exists only on processors that support the 1-setting of the “load IA32_PAT” VM-exit control.
2. This field exists only on processors that support the 1-setting of the “load IA32_EFER” VM-exit control.
3. This field exists only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
4. This field exists only on processors that support the 1-setting of the “load PKRS” VM-exit control.

B.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 24.11.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-8 enumerates the 32-bit control fields.

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Pin-based VM-execution controls	00000000B	00004000H
Primary processor-based VM-execution controls	00000001B	00004002H
Exception bitmap	00000010B	00004004H
Page-fault error-code mask	00000011B	00004006H
Page-fault error-code match	00000100B	00004008H
CR3-target count	00000101B	0000400AH
Primary VM-exit controls	00000110B	0000400CH
VM-exit MSR-store count	00000111B	0000400EH
VM-exit MSR-load count	00001000B	00004010H
VM-entry controls	00001001B	00004012H
VM-entry MSR-load count	00001010B	00004014H
VM-entry interruption-information field	00001011B	00004016H
VM-entry exception error code	00001100B	00004018H
VM-entry instruction length	00001101B	0000401AH
TPR threshold ¹	00001110B	0000401CH
Secondary processor-based VM-execution controls ²	00001111b	0000401EH
PLE_Gap ³	000010000b	00004020H
PLE_Window ³	000010001b	00004022H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control.

B.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-9 enumerates the 32-bit read-only data fields.

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
VM-instruction error	00000000B	00004400H
Exit reason	00000001B	00004402H
VM-exit interruption information	00000010B	00004404H
VM-exit interruption error code	00000011B	00004406H
IDT-vectoring information field	00000100B	00004408H
IDT-vectoring error code	00000101B	0000440AH
VM-exit instruction length	00000110B	0000440CH

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
VM-exit instruction information	000000111B	0000440EH

B.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-10 enumerates the 32-bit guest-state fields.

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES limit	000000000B	00004800H
Guest CS limit	000000001B	00004802H
Guest SS limit	000000010B	00004804H
Guest DS limit	000000011B	00004806H
Guest FS limit	000000100B	00004808H
Guest GS limit	000000101B	0000480AH
Guest LDTR limit	000000110B	0000480CH
Guest TR limit	000000111B	0000480EH
Guest GDTR limit	000001000B	00004810H
Guest IDTR limit	000001001B	00004812H
Guest ES access rights	000001010B	00004814H
Guest CS access rights	000001011B	00004816H
Guest SS access rights	000001100B	00004818H
Guest DS access rights	000001101B	0000481AH
Guest FS access rights	000001110B	0000481CH
Guest GS access rights	000001111B	0000481EH
Guest LDTR access rights	000010000B	00004820H
Guest TR access rights	000010001B	00004822H
Guest interruptibility state	000010010B	00004824H
Guest activity state	000010011B	00004826H
Guest SMBASE	000010100B	00004828H
Guest IA32_SYSENTER_CS	000010101B	0000482AH
VMX-preemption timer value ¹	000010111B	0000482EH

NOTES:

1. This field exists only on processors that support the 1-setting of the "activate VMX-preemption timer" VM-execution control.

The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

B.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table B-11.

Table B-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host IA32_SYSENTER_CS	000000000B	00004C00H

B.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 24.11.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-12 enumerates the natural-width control fields.

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR0 guest/host mask	000000000B	00006000H
CR4 guest/host mask	000000001B	00006002H
CR0 read shadow	000000010B	00006004H
CR4 read shadow	000000011B	00006006H
CR3-target value 0	000000100B	00006008H
CR3-target value 1	000000101B	0000600AH
CR3-target value 2	000000110B	0000600CH
CR3-target value 3 ¹	000000111B	0000600EH

NOTES:

1. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

B.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-13 enumerates the natural-width read-only data fields.

Table B-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
Exit qualification	000000000B	00006400H
I/O RCX	000000001B	00006402H
I/O RSI	000000010B	00006404H
I/O RDI	000000011B	00006406H
I/O RIP	000000100B	00006408H
Guest-linear address	000000101B	0000640AH

B.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-14 enumerates the natural-width guest-state fields.

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CR0	000000000B	00006800H
Guest CR3	000000001B	00006802H
Guest CR4	000000010B	00006804H
Guest ES base	000000011B	00006806H
Guest CS base	000000100B	00006808H
Guest SS base	000000101B	0000680AH
Guest DS base	000000110B	0000680CH
Guest FS base	000000111B	0000680EH
Guest GS base	000001000B	00006810H
Guest LDTR base	000001001B	00006812H
Guest TR base	000001010B	00006814H
Guest GDTR base	000001011B	00006816H
Guest IDTR base	000001100B	00006818H
Guest DR7	000001101B	0000681AH
Guest RSP	000001110B	0000681CH
Guest RIP	000001111B	0000681EH
Guest RFLAGS	000010000B	00006820H
Guest pending debug exceptions	000010001B	00006822H
Guest IA32_SYSENTER_ESP	000010010B	00006824H
Guest IA32_SYSENTER_EIP	000010011B	00006826H
Guest IA32_S_CET ¹	000010100B	00006828H
Guest SSP ¹	000010101B	0000682AH
Guest IA32_INTERRUPT_SSP_TABLE_ADDR ¹	000010110B	0000682CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

B.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-15 enumerates the natural-width host-state fields.

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host CR0	000000000B	00006C00H

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Host CR3	000000001B	00006C02H
Host CR4	000000010B	00006C04H
Host FS base	000000011B	00006C06H
Host GS base	000000100B	00006C08H
Host TR base	000000101B	00006C0AH
Host GDTR base	000000110B	00006C0CH
Host IDTR base	000000111B	00006C0EH
Host IA32_SYSENTER_ESP	000001000B	00006C10H
Host IA32_SYSENTER_EIP	000001001B	00006C12H
Host RSP	000001010B	00006C14H
Host RIP	000001011B	00006C16H
Host IA32_S_CET ¹	000001100B	00006C18H
Host SSP ¹	000001101B	00006C1AH
Host IA32_INTERRUPT_SSP_TABLE_ADDR ¹	000001110B	00006C1CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

17. Updates to Appendix C, Volume 3D

Change bars and green text show changes to Appendix C of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter: Added VMX basic exit reason 65 to Table C-1, "Basic Exit Reasons".

APPENDIX C VMX BASIC EXIT REASONS

Every VM exit writes a 32-bit exit reason to the VMCS (see Section 24.9.1). Certain VM-entry failures also do this (see Section 26.8). The low 16 bits of the exit-reason field form the basic exit reason which provides basic information about the cause of the VM exit or VM-entry failure.

Table C-1 lists values for basic exit reasons and explains their meaning. Entries apply to VM exits, unless otherwise noted.

Table C-1. Basic Exit Reasons

Basic Exit Reason	Description
0	Exception or non-maskable interrupt (NMI). Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. This case includes executions of BOUND that cause #BR, executions of INT1 (they cause #DB), executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UDO, UD1, and UD2 (they cause #UD). 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1.
1	External interrupt. An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	Triple fault. The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	INIT signal. An INIT signal arrived
4	Start-up IPI (SIPI). A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	I/O system-management interrupt (SMI). An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 31.15.2).
6	Other SMI. An SMI arrived and caused an SMM VM exit (see Section 31.15.2) but not immediately after retirement of an I/O instruction.
7	Interrupt window. At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	NMI window. At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	Task switch. Guest software attempted a task switch.
10	CPUID. Guest software attempted to execute CPUID.
11	GETSEC. Guest software attempted to execute GETSEC.
12	HLT. Guest software attempted to execute HLT and the "HLT exiting" VM-execution control was 1.
13	INVD. Guest software attempted to execute INVD.
14	INVLPG. Guest software attempted to execute INVLPG and the "INVLPG exiting" VM-execution control was 1.
15	RDPMC. Guest software attempted to execute RDPMC and the "RDPMC exiting" VM-execution control was 1.
16	RDTSC. Guest software attempted to execute RDTSC and the "RDTSC exiting" VM-execution control was 1.
17	RSM. Guest software attempted to execute RSM in SMM.
18	VMCALL. VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 31.15.2).
19	VMCLEAR. Guest software attempted to execute VMCLEAR.
20	VMLAUNCH. Guest software attempted to execute VMLAUNCH.
21	VMPTRLD. Guest software attempted to execute VMPTRLD.
22	VMPTRST. Guest software attempted to execute VMPTRST.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
23	VMREAD. Guest software attempted to execute VMREAD.
24	VMRESUME. Guest software attempted to execute VMRESUME.
25	VMWRITE. Guest software attempted to execute VMWRITE.
26	VMXOFF. Guest software attempted to execute VMXOFF.
27	VMXON. Guest software attempted to execute VMXON.
28	Control-register accesses. Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 25.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1. Such VM exits instead use basic exit reason 43.
29	MOV DR. Guest software attempted a MOV to or from a debug register and the “MOV-DR exiting” VM-execution control was 1.
30	I/O instruction. Guest software attempted to execute an I/O instruction and either: 1: The “use I/O bitmaps” VM-execution control was 0 and the “unconditional I/O exiting” VM-execution control was 1. 2: The “use I/O bitmaps” VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1.
31	RDMSR. Guest software attempted to execute RDMSR and either: 1: The “use MSR bitmaps” VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H - 00001FFFH nor in the range C0000000H - C0001FFFH. 3: The value of RCX was in the range 00000000H - 00001FFFH and the n^{th} bit in read bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H - C0001FFFH and the n^{th} bit in read bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
32	WRMSR. Guest software attempted to execute WRMSR and either: 1: The “use MSR bitmaps” VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H - 00001FFFH nor in the range C0000000H - C0001FFFH. 3: The value of RCX was in the range 00000000H - 00001FFFH and the n^{th} bit in write bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H - C0001FFFH and the n^{th} bit in write bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
33	VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 26.3.1.
34	VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs. See Section 26.4.
36	MWAIT. Guest software attempted to execute MWAIT and the “MWAIT exiting” VM-execution control was 1.
37	Monitor trap flag. A VM exit occurred due to the 1-setting of the “monitor trap flag” VM-execution control (see Section 25.5.2) or VM entry injected a pending MTF VM exit as part of VM entry (see Section 26.6.2).
39	MONITOR. Guest software attempted to execute MONITOR and the “MONITOR exiting” VM-execution control was 1.
40	PAUSE. Either guest software attempted to execute PAUSE and the “PAUSE exiting” VM-execution control was 1 or the “PAUSE-loop exiting” VM-execution control was 1 and guest software executed a PAUSE loop with execution time exceeding PLE_Window (see Section 25.1.3).
41	VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 26.9).
43	TPR below threshold. The logical processor determined that the value of bits 7:4 of the byte at offset 080H on the virtual-APIC page was below that of the TPR threshold VM-execution control field while the “use TPR shadow” VM-execution control was 1 either as part of TPR virtualization (Section 29.1.2) or VM entry (Section 26.7.7).
44	APIC access. Guest software attempted to access memory at a physical address on the APIC-access page and the “virtualize APIC accesses” VM-execution control was 1 (see Section 29.4).
45	Virtualized EOI. EOI virtualization was performed for a virtual interrupt whose vector indexed a bit set in the EOI-exit bitmap.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
46	Access to GDTR or IDTR. Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the “descriptor-table exiting” VM-execution control was 1.
47	Access to LDTR or TR. Guest software attempted to execute LLDT, LTR, SLDT, or STR and the “descriptor-table exiting” VM-execution control was 1.
48	EPT violation. An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.
49	EPT misconfiguration. An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.
50	INVEPT. Guest software attempted to execute INVEPT.
51	RDTSCP. Guest software attempted to execute RDTSCP and the “enable RDTSCP” and “RDTSC exiting” VM-execution controls were both 1.
52	VMX-preemption timer expired. The preemption timer counted down to zero.
53	INNVPID. Guest software attempted to execute INNVPID.
54	WBINVD or WBNOINVD. Guest software attempted to execute WBINVD or WBNOINVD and the “WBINVD exiting” VM-execution control was 1.
55	XSETBV. Guest software attempted to execute XSETBV.
56	APIC write. Guest software completed a write to the virtual-APIC page that must be virtualized by VMM software (see Section 29.4.3.3).
57	RDRAND. Guest software attempted to execute RDRAND and the “RDRAND exiting” VM-execution control was 1.
58	INVPCID. Guest software attempted to execute INVPCID and the “enable INVPCID” and “INVLPG exiting” VM-execution controls were both 1.
59	VMFUNC. Guest software invoked a VM function with the VMFUNC instruction and the VM function either was not enabled or generated a function-specific condition causing a VM exit.
60	ENCLS. Guest software attempted to execute ENCLS, “enable ENCLS exiting” VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the ENCLS-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the ENCLS-exiting bitmap is 1.
61	RDSEED. Guest software attempted to execute RDSEED and the “RDSEED exiting” VM-execution control was 1.
62	Page-modification log full. The processor attempted to create a page-modification log entry and the value of the PML index was not in the range 0-511.
63	XSAVES. Guest software attempted to execute XSAVES, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
64	XRSTORS. Guest software attempted to execute XRSTORS, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
65	PCONFIG. Guest software attempted to execute PCONFIG, “enable PCONFIG” VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the PCONFIG-exiting bitmap is 1.
66	SPP-related event. The processor attempted to determine an access’s sub-page write permission and encountered an SPP miss or an SPP misconfiguration. See Section 28.3.4.2.
67	UMWAIT. Guest software attempted to execute UMWAIT and the “enable user wait and pause” and “RDTSC exiting” VM-execution controls were both 1.
68	TPAUSE. Guest software attempted to execute TPAUSE and the “enable user wait and pause” and “RDTSC exiting” VM-execution controls were both 1.
69	LOADIWKEY. Guest software attempted to execute LOADIWKEY and the “LOADIWKEY exiting” VM-execution control was 1.

18. Updates to Chapter 2, Volume 4

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers (MSRs)*.

Changes to this chapter:

- Updated reserved range of MSRs in section 2.1.
- Removed incorrect CPUID signature value for 12th generation Intel® Core™ processors.
- MSR 8BH is R/W instead of RO; updated all occurrences of this where incorrect.
- Added a new MSR 195H, "IA32_Overclocking_Status", and an enumeration bit in the IA32_ARCH_CAPABILITIES MSR (address 10AH).

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 8.9.1, "Hierarchical Mapping of Shared Resources" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H leaf function to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Table 2-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_85H	Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture
06_57H	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series based on Knights Landing microarchitecture
06_8FH	Future Intel® Xeon® processors based on Sapphire Rapids microarchitecture
06_97H, 06_9AH	12th generation Intel® Core™ processors supporting Alder Lake performance hybrid architecture
06_8CH, 06_8DH	11th generation Intel® Core™ processors based on Tiger Lake microarchitecture
06_A7H, 06_A8H	11th generation Intel® Core™ processors based on Rocket Lake microarchitecture
06_7DH, 06_7EH	10th generation Intel® Core™ processors based on Ice Lake microarchitecture
06_A5H, 06_A6H	10th generation Intel® Core™ processors based on Comet Lake microarchitecture
06_66H	Intel® Core™ processors based on Cannon Lake microarchitecture
06_8EH, 06_9EH	7th generation Intel® Core™ processors based on Kaby Lake microarchitecture, 8th and 9th generation Intel® Core™ processors based on Coffee Lake microarchitecture, Intel® Xeon® E processors based on Coffee Lake microarchitecture
06_6AH, 06_6CH	3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture
06_55H	Intel® Xeon® Processor Scalable Family based on Skylake microarchitecture, 2nd generation Intel® Xeon® Processor Scalable Family based on Cascade Lake product, and 3rd generation Intel® Xeon® Processor Scalable Family based on Cooper Lake product
06_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family and E3-1200 v5 product family based on Skylake microarchitecture
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture

Table 2-1. CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_4FH	Intel Xeon processor E5 v4 Family based on Broadwell microarchitecture, Intel Xeon processor E7 v4 Family, Intel Core i7-69xx Processor Extreme Edition
06_47H	5th generation Intel Core processors, Intel Xeon processor E3-1200 v4 product family based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-4600/2600/1600 v3 product families, Intel Xeon processor E7 v3 product families based on Haswell-E microarchitecture, Intel Core i7-59xx Processor Extreme Edition
06_3CH, 06_45H, 06_46H	4th Generation Intel Core processor and Intel Xeon processor E3-1200 v3 product family based on Haswell microarchitecture
06_3EH	Intel Xeon processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture
06_3EH	Intel Xeon processor E5-2600/1600 v2 product families and Intel Xeon processor E5-2400 v2 product family based on Ivy Bridge-E microarchitecture, Intel Core i7-49xx Processor Extreme Edition
06_3AH	3rd Generation Intel Core Processor and Intel Xeon processor E3-1200 v2 product family based on Ivy Bridge microarchitecture
06_2DH	Intel Xeon processor E5 Family based on Sandy Bridge microarchitecture, Intel Core i7-39xx Processor Extreme Edition
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon processor E3-1200 product family; 2nd Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5 and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon processor 3400, 3500, 5500 series
06_1DH	Intel Xeon processor MP 7400 series
06_17H	Intel Xeon processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_86H, 06_96H, 06_9CH	Intel Atom® processors, Intel® Celeron® processors, Intel® Pentium® processors, and Intel® Pentium® Silver processors based on Tremont Microarchitecture
06_7AH	Intel Atom processors based on Goldmont Plus microarchitecture
06_5FH	Intel Atom processors based on Goldmont microarchitecture (Denverton)
06_5CH	Intel Atom processors based on Goldmont microarchitecture
06_4CH	Intel Atom processor X7-Z8000 and X5-Z8000 series based on Airmont microarchitecture
06_5DH	Intel Atom processor X3-C3000 based on Silvermont microarchitecture
06_5AH	Intel Atom processor Z3500 series
06_4AH	Intel Atom processor Z3400 series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 series
06_36H	Intel Atom processor S1000 Series

Table 2-1. CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	Intel Atom processor family, Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon processor, Intel Xeon processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon processor, Intel Pentium III processor
06_03H, 06_05H	Intel Pentium II Xeon processor, Intel Pentium II processor
06_01H	Intel Pentium Pro processor
05_01H, 05_02H, 05_04H	Intel Pentium processor, Intel Pentium processor with MMX Technology

The Intel® Quark™ SoC X1000 processor can be identified by the signature of DisplayFamily_DisplayModel = 05_09H and SteppingID = 0

2.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table 2-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 2-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a model-specific MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 2-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of “DF_DM” (see Table 2-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-2. “MAXPHYADDR” is reported by CPUID.8000_0008H leaf.

MSR address range between 40000000H - 4000FFFFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

Table 2-2. IA-32 Architectural MSRs

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Section 2.23, “MSRs in Pentium Processors.”	Pentium Processor (05_01H)
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 2.23, “MSRs in Pentium Processors.”	DF_DM = 05_01H
6H	6	IA32_MONITOR_FILTER_SIZE	See Section 8.10.5, “Monitor/Mwait Address Range Determination.”	0F_03H

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment	
Hex	Decimal				
10H	16	IA32_TIME_STAMP_COUNTER (TSC)	See Section 17.17, "Time-Stamp Counter."	05_01H	
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	Platform ID (R/O) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	06_01H	
		49:0	Reserved		
		52:50	Platform Id (R/O) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7		
		63:53	Reserved		
1BH	27	IA32_APIC_BASE (APIC_BASE)	This register holds the APIC base address, permitting the relocation of the APIC memory map. See Section 10.4.4, "Local APIC Status and Location" and Section 10.4.5, "Relocating the Local APIC Registers".	06_01H	
		7:0	Reserved		
		8	BSP flag (R/W)		
		9	Reserved		
		10	Enable x2APIC mode.		06_1AH
		11	APIC Global Enable (R/W)		
		(MAXPHYADDR - 1):12	APIC Base (R/W)		
63: MAXPHYADDR	Reserved				
3AH	58	IA32_FEATURE_CONTROL	Control Features in Intel 64 Processor (R/W)	If any one enumeration condition for defined bit field holds.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written; writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL contents are preserved across RESET when PWRGOOD is not deasserted.	If any one enumeration condition for defined bit field position greater than bit 0 holds.
		1	Enable VMX inside SMX operation (R/WL): This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology. BIOS must set this bit only when the CPUID function 1 returns VMX feature flag and SMX feature flag set (ECX bits 5 and 6 respectively).	If CPUID.01H:ECX[5] = 1 && CPUID.01H:ECX[6] = 1
		2	Enable VMX outside SMX operation (R/WL): This bit enables VMX for a system executive that does not require SMX. BIOS must set this bit only when the CPUID function 1 returns the VMX feature flag set (ECX bit 5).	If CPUID.01H:ECX[5] = 1
		7:3	Reserved	
		14:8	SENTER Local Function Enables (R/WL): When set, each bit in the field represents an enable control for a corresponding SENTER function. This field is supported only if CPUID.1:ECX.[bit 6] is set.	If CPUID.01H:ECX[6] = 1
		15	SENTER Global Enable (R/WL): This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.1:ECX.[bit 6] is set.	If CPUID.01H:ECX[6] = 1
		16	Reserved	
		17	SGX Launch Control Enable (R/WL): This bit must be set to enable runtime re-configuration of SGX Launch Control via the IA32_SGXLEPUBKEYHASHn MSR.	If CPUID.(EAX=07H, ECX=0H): ECX[30] = 1
		18	SGX Global Enable (R/WL): This bit must be set to enable SGX leaf functions.	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		19	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		20	LMCE On (R/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	
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H); EBX[1] = 1
		63:0	THREAD_ADJUST: Local offset value of the IA32_TSC for a logical processor. Reset value is zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
48H	72	IA32_SPEC_CTRL	Speculation Control (R/W) The MSR bits are defined as logical processor scope. On some core implementations, the bits may impact sibling logical processors on the same core. This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.	If any one of the enumeration conditions for defined bit field positions holds.
		0	Indirect Branch Restricted Speculation (IBRS). Restricts speculation of indirect branch.	If CPUID.(EAX=07H, ECX=0); EDX[26]=1
		1	Single Thread Indirect Branch Predictors (STIBP). Prevents indirect branch predictions on all logical processors on the core from being controlled by any sibling logical processor in the same core.	If CPUID.(EAX=07H, ECX=0); EDX[27]=1
		2	Speculative Store Bypass Disable (SSBD) delays speculative execution of a load until the addresses for all older stores are known.	If CPUID.(EAX=07H, ECX=0); EDX[31]=1
		63:3	Reserved	
49H	73	IA32_PRED_CMD	Prediction Command (WO) Gives software a way to issue commands that affect the state of predictors.	If any one of the enumeration conditions for defined bit field positions holds.
		0	Indirect Branch Prediction Barrier (IBPB).	If CPUID.(EAX=07H, ECX=0); EDX[26]=1
		63:1	Reserved	
4EH	78	IA32_PPIN_CTL	Protected Processor Inventory Number Enable Control (R/W)	If CPUID.(EAX=07H, ECX=01H); EBX[0]=1 ¹

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	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	
4FH	79	IA32_PPIN	Protected Processor Inventory Number (R/O)	If CPUID.(EAX=07H, ECX=01H):EBX[0]=1 ¹
		63:0	Protected Processor Inventory Number (R/O) A unique value within a given CPUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to IA32_PPIN is enabled. Access to IA32_PPIN is permitted only if IA32_PPIN_CTL[bits 1:0] = '10b'.	
79H	121	IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)	BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 9.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
8BH	139	IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)	BIOS Update Signature (R/W) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
		31:0	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:32	It is recommended that this field be pre-loaded with zero prior to executing CPUID. If the field remains zero following the execution of CPUID, this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature.	
8CH	140	IA32_SGXLEPUBKEYHASH0	IA32_SGXLEPUBKEYHASH[63:0] (R/W) Bits 63:0 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && CPUID.(EAX=07H, ECX=0H):ECX[30]=1. Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1.
8DH	141	IA32_SGXLEPUBKEYHASH1	IA32_SGXLEPUBKEYHASH[127:64] (R/W) Bits 127:64 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	
8EH	142	IA32_SGXLEPUBKEYHASH2	IA32_SGXLEPUBKEYHASH[191:128] (R/W) Bits 191:128 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	
8FH	143	IA32_SGXLEPUBKEYHASH3	IA32_SGXLEPUBKEYHASH[255:192] (R/W) Bits 255:192 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	
9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	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 31.14.4).	If IA32_VMX_MISC[28]
		11:3	Reserved	
		31:12	MSEG Base (R/W)	
		63:32	Reserved	
9EH	158	IA32_SMBASE	Base address of the logical processor's SMRAM image (R/O, SMM only).	If IA32_VMX_MISC[15]
BCH	188	IA32_MISC_PACKAGE_CTL5	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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	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.	
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
C3H	195	IA32_PMC2	General Performance Counter 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
C4H	196	IA32_PMC3	General Performance Counter 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
C5H	197	IA32_PMC4	General Performance Counter 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
C6H	198	IA32_PMC5	General Performance Counter 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5
C7H	199	IA32_PMC6	General Performance Counter 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
C8H	200	IA32_PMC7	General Performance Counter 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
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.
E1H	225	IA32_UMWAIT_CONTROL	UMWAIT Control (R/W)	
		0	C0.2 is not allowed by the OS. Value of "1" means all C0.2 requests revert to C0.1.	
		1	Reserved	
		31:2	Determines the maximum time in TSC-quanta that the processor can reside in either C0.1 or C0.2. A zero value indicates no maximum time. The maximum time value is a 32-bit value where the upper 30 bits come from this field and the lower two bits are zero.	
E7H	231	IA32_MPERF	TSC Frequency Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:0	CO_MCNT: CO TSC Frequency Clock Count Increments at fixed interval (relative to TSC freq.) when the logical processor is in CO. Cleared upon overflow / wrap-around of IA32_APERF.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	CO_ACNT: CO Actual Frequency Clock Count Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in CO. Cleared upon overflow / wrap-around of IA32_MPERF.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (R/O) See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	06_01H
		7:0	VCNT: The number of variable memory type ranges in the processor.	
		8	Fixed range MTRRs are supported when set.	
		9	Reserved	
		10	WC Supported when set.	
		11	SMRR Supported when set.	
		12	PRMRR supported when set.	
		63:13	Reserved	
10AH	266	IA32_ARCH_CAPABILITIES	Enumeration of Architectural Features (R/O)	If CPUID.(EAX=07H, ECX=0):EDX[29]=1
		0	RDCL_NO: The processor is not susceptible to Rogue Data Cache Load (RDCL).	
		1	IBRS_ALL: The processor supports enhanced IBRS.	
		2	RSBA: The processor supports RSB Alternate. Alternative branch predictors may be used by RET instructions when the RSB is empty. SW using retpoline may be affected by this behavior.	
		3	SKIP_L1DFL_VMENTRY: A value of 1 indicates the hypervisor need not flush the L1D on VM entry.	
		4	SSB_NO: Processor is not susceptible to Speculative Store Bypass.	
		5	MDS_NO: Processor is not susceptible to Microarchitectural Data Sampling (MDS).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6	IF_PSCCHANGE_MC_NO: The processor is not susceptible to a machine check error due to modifying the size of a code page without TLB invalidation.	
		7	TSX_CTRL: If 1, indicates presence of IA32_TSX_CTRL MSR.	
		8	TAA_NO: If 1, processor is not affected by TAA.	
		9	Reserved	
		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.	
		22:12	Reserved	
		23	OVERCLOCKING_STATUS: If set, the IA32_OVERCLOCKING_STATUS MSR exists.	
	63:24	Reserved		
10BH	267	IA32_FLUSH_CMD	Flush Command (w0) 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	
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	
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR (R/W)	06_01H
		15:0	CS Selector.	
		31:16	Not used.	Can be read and written.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:32	Not used.	Writes ignored; reads return zero.
175H	373	IA32_SYSENTER_ESP	SYSENTER_ESP_MSR (R/W)	06_01H
176H	374	IA32_SYSENTER_EIP	SYSENTER_EIP_MSR (R/W)	06_01H
179H	377	IA32_MCG_CAP (MCG_CAP)	Global Machine Check Capability (R/O)	06_01H
		7:0	Count: Number of reporting banks.	
		8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set.	
		9	MCG_EXT_P: Extended machine check state registers are present if this bit is set.	
		10	MCP_CMCI_P: Support for corrected MC error event is present.	06_01H
		11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
		15:12	Reserved	
		23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	
		24	MCG_SER_P: The processor supports software error recovery if this bit is set.	
		25	Reserved	
		26	MCG_ELOG_P: Indicates that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format "Generic Error Data Entry" that augments the data included in machine check bank registers.	06_3EH
		27	MCG_LMCE_P: Indicates that the processor supports extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (R/W0)	06_01H
		0	RIPV. Restart IP valid.	06_01H
		1	EIPV. Error IP valid.	06_01H
		2	MCIP. Machine check in progress.	06_01H
		3	LMCE_S	If IA32_MCG_CAP.LMCE_P[27]=1
		63:4	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	If IA32_MCG_CAP.CTL_P[8] = 1
180H-185H	384-389	Reserved		06_0EH ²
186H	390	IA32_PERFEVTSELO (PERFEVTSELO)	Performance Event Select Register 0 (R/W)	If CPUID.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		
187H	391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	Performance Event Select Register 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
188H	392	IA32_PERFEVTSEL2	Performance Event Select Register 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
189H	393	IA32_PERFEVTSEL3	Performance Event Select Register 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
18AH	394	IA32_PERFEVTSEL4	Performance Event Select Register 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
18BH	395	IA32_PERFEVTSEL5	Performance Event Select Register 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
18CH	396	IA32_PERFEVTSEL6	Performance Event Select Register 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
18DH	397	IA32_PERFEVTSEL7	Performance Event Select Register 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
18AH-194H	394-404	Reserved		06_0EH ³
195H	405	IA32_OVERCLOCKING_STATUS	Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR.	
		0	Overclocking Utilized Indicates if specific forms of overclocking have been enabled on this boot or reset cycle: 0 indicates no, 1 indicates yes.	
		1	Undervolt Protection Indicates if the "Dynamic OC Undervolt Protection" security feature is active: 0 indicates disabled, 1 indicates enabled.	
		2	Overclocking Secure Status Indicates that overclocking capabilities have been unlocked by BIOS, with or without overclocking: 0 indicates Not Secured, 1 indicates Secure.	
		63:4	Reserved	
196H-197H	406-407	Reserved		06_0EH ³
198H	408	IA32_PERF_STATUS	Current Performance Status (R/O) See Section 14.1.1, "Software Interface For Initiating Performance State Transitions".	0F_03H
		15:0	Current performance State Value.	
		63:16	Reserved	
199H	409	IA32_PERF_CTL	Performance Control MSR (R/W) Software makes a request for a new Performance state (P-State) by writing this MSR. See Section 14.1.1, "Software Interface For Initiating Performance State Transitions".	0F_03H
		15:0	Target performance State Value.	
		31:16	Reserved	
		32	IDA Engage (R/W) When set to 1: disengages IDA.	06_0FH (Mobile only)
		63:33	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 14.8.3, "Software Controlled Clock Modulation."	If CPUID.01H:EDX[22] = 1
		0	Extended On-Demand Clock Modulation Duty Cycle.	If CPUID.06H:EAX[5] = 1
		3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation.	If CPUID.01H:EDX[22] = 1
		4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	If CPUID.01H:EDX[22] = 1
		63:5	Reserved	
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 14.8.2, "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
		63:25	Reserved	
19CH	412	IA32_THERM_STATUS	Thermal Status Information (R/O) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 14.8.2, "Thermal Monitor".	If CPUID.01H:EDX[22] = 1
		0	Thermal Status (R/O)	If CPUID.01H:EDX[22] = 1
		1	Thermal Status Log (R/W)	If CPUID.01H:EDX[22] = 1
		2	PROCHOT # or FORCEPR# event (R/O)	If CPUID.01H:EDX[22] = 1
		3	PROCHOT # or FORCEPR# log (R/WCO)	If CPUID.01H:EDX[22] = 1
		4	Critical Temperature Status (R/O)	If CPUID.01H:EDX[22] = 1
		5	Critical Temperature Status log (R/WCO)	If CPUID.01H:EDX[22] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6	Thermal Threshold #1 Status (R/O)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WCO)	If CPUID.01H:ECX[8] = 1
		8	Thermal Threshold #2 Status (R/O)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #2 log (R/WCO)	If CPUID.01H:ECX[8] = 1
		10	Power Limitation Status (R/O)	If CPUID.06H:EAX[4] = 1
		11	Power Limitation log (R/WCO)	If CPUID.06H:EAX[4] = 1
		12	Current Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
		13	Current Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
		14	Cross Domain Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
		15	Cross Domain Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
		22:16	Digital Readout (R/O)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved	
		30:27	Resolution in Degrees Celsius (R/O)	If CPUID.06H:EAX[0] = 1
		31	Reading Valid (R/O)	If CPUID.06H:EAX[0] = 1
		63:32	Reserved	
1A0H	416	IA32_MISC_ENABLE	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.	
		0	Fast-Strings Enable When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default). When clear, fast-strings are disabled.	OF_OH
		2:1	Reserved	
		3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled. Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2 and adaptive thermal throttling will still be activated. The default value of this field varies with product . See respective tables where default value is listed.	OF_OH
		6:4	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.	0F_0H
		10:8	Reserved	
		11	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS). 0 = BTS is supported.	0F_0H
		12	Processor Event Based Sampling (PEBS) Unavailable (R/O) 1 = PEBS is not supported. 0 = PEBS is supported.	06_0FH
		15:13	Reserved	
		16	Enhanced Intel SpeedStep Technology Enable (R/w) 0= Enhanced Intel SpeedStep Technology disabled. 1 = Enhanced Intel SpeedStep Technology enabled.	If CPUID.01H: ECX[7] = 1
		17	Reserved	
		18	ENABLE MONITOR FSM (R/w) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
		21:19	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		22	<p>Limit CPUID Maxval (R/W)</p> <p>When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2.</p> <p>BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 2.</p> <p>Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 2, this bit is supported.</p> <p>Otherwise, this bit is not supported. Setting this bit when the maximum value is not greater than 2 may generate a #GP exception.</p> <p>Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 2.</p>	0F_03H
		23	<p>xTPR Message Disable (R/W)</p> <p>When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.</p>	If CPUID.01H:ECX[14] = 1
		33:24	Reserved	
		34	<p>XD Bit Disable (R/W)</p> <p>When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0).</p> <p>When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages.</p> <p>BIOS must not alter the contents of this bit location, if XD bit is not supported. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.</p>	If CPUID.80000001H:EDX[20] = 1
		63:35	Reserved	
1B0H	432	IA32_ENERGY_PERF_BIAS	Performance Energy Bias Hint (R/W)	If CPUID.6H:ECX[3] = 1
		3:0	<p>Power Policy Preference:</p> <p>0 indicates preference to highest performance.</p> <p>15 indicates preference to maximize energy saving.</p>	
		63:4	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package Thermal Status Information (R/O) Contains status information about the package's thermal sensor. See Section 14.9, "Package Level Thermal Management."	If CPUID.06H: EAX[6] = 1
		0	Pkg Thermal Status (R/O)	
		1	Pkg Thermal Status Log (R/W)	
		2	Pkg PROCHOT # event (R/O)	
		3	Pkg PROCHOT # log (R/WCO)	
		4	Pkg Critical Temperature Status (R/O)	
		5	Pkg Critical Temperature Status Log (R/WCO)	
		6	Pkg Thermal Threshold #1 Status (R/O)	
		7	Pkg Thermal Threshold #1 Log (R/WCO)	
		8	Pkg Thermal Threshold #2 Status (R/O)	
		9	Pkg Thermal Threshold #1 Log (R/WCO)	
		10	Pkg Power Limitation Status (R/O)	
		11	Pkg Power Limitation Log (R/WCO)	
		15:12	Reserved	
		22:16	Pkg Digital Readout (R/O)	
		25:23	Reserved	
		26	Hardware Feedback Interface Structure Change Status	If CPUID.06H:EAX.[19] = 1
63:27	Reserved			
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Pkg Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the package's thermal sensor. See Section 14.9, "Package Level Thermal Management."	If CPUID.06H: EAX[6] = 1
		0	Pkg High-Temperature Interrupt Enable	
		1	Pkg Low-Temperature Interrupt Enable	
		2	Pkg PROCHOT# Interrupt Enable	
		3	Reserved	
		4	Pkg Overheat Interrupt Enable	
		7:5	Reserved	
		14:8	Pkg Threshold #1 Value	
		15	Pkg Threshold #1 Interrupt Enable	
22:16	Pkg Threshold #2 Value			

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		23	Pkg Threshold #2 Interrupt Enable	
		24	Pkg Power Limit Notification Enable	
		25	Hardware Feedback Interrupt Enable	If CPUID.06H:EAX.[19] = 1
		63:26	Reserved	
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
		5:2	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)
63:16	Reserved			

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
1DDH	477	IA32_LER_FROM_IP	Last Event Record Source IP Register (R/W)	
		63:0	FROM_IP The source IP of the recorded branch or event, in canonical form.	Reset Value: 0
1DEH	478	IA32_LER_TO_IP	Last Event Record Destination IP Register (R/W)	
		63:0	TO_IP The destination IP of the recorded branch or event, in canonical form.	Reset Value: 0
1E0H	480	IA32_LER_INFO	Last Event Record Info Register (R/W)	
		55:0	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
		59:56	BR_TYPE The branch type recorded by this LBR. Encodings match those of IA32_LBR_x_INFO.	Reset Value: 0
		60	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
		61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel® TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	Reset Value: 0
		62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel® TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	Reset Value: 0
		63	MISPRED The recorded branch taken/not-taken resolution (for conditional branches) or target (for any indirect branch, including RETs) was mispredicted.	Reset Value: 0
1F2H	498	IA32_SMRR_PHYSBASE	SMRR Base Address (Writeable only in SMM) Base address of SMM memory range.	If IA32_MTRRCAP.SMRR[11] = 1
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved	
		31:12	PhysBase SMRR physical Base Address.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:32	Reserved	
1F3H	499	IA32_SMRR_PHYSMASK	SMRR Range Mask (Writeable only in SMM) Range Mask of SMM memory range.	If IA32_MTRRCAP[SMRR] = 1
		10:0	Reserved	
		11	Valid Enable range mask.	
		31:12	PhysMask SMRR address range mask.	
		63:32	Reserved	
1F8H	504	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	If CPUID.01H: ECX[18] = 1
1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	If CPUID.01H: ECX[18] = 1
1FAH	506	IA32_DCA_0_CAP	DCA type 0 Status and Control register.	If CPUID.01H: ECX[18] = 1
		0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	
		2:1	TRANSACTION	
		6:3	DCA_TYPE	
		10:7	DCA_QUEUE_SIZE	
		12:11	Reserved	
		16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	
		23:17	Reserved	
		24	SW_BLOCK: SW can request DCA block by setting this bit.	
		25	Reserved	
		26	HW_BLOCK: Set when DCA is blocked by HW (e.g., CR0.CD = 1).	
31:27	Reserved			
200H	512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	See Section 11.11.2.3, "Variable Range MTRRs."	If IA32_MTRRCAP[7:0] > 0
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	If IA32_MTRRCAP[7:0] > 0
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	If IA32_MTRRCAP[7:0] > 1
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	If IA32_MTRRCAP[7:0] > 1
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	If IA32_MTRRCAP[7:0] > 2
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	If IA32_MTRRCAP[7:0] > 2
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	If IA32_MTRRCAP[7:0] > 3
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	If IA32_MTRRCAP[7:0] > 3
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	If IA32_MTRRCAP[7:0] > 4
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	If IA32_MTRRCAP[7:0] > 4
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	If IA32_MTRRCAP[7:0] > 5

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	If IA32_MTRRCAP[7:0] > 5
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	If IA32_MTRRCAP[7:0] > 6
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	If IA32_MTRRCAP[7:0] > 6
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	If IA32_MTRRCAP[7:0] > 7
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	If IA32_MTRRCAP[7:0] > 7
210H	528	IA32_MTRR_PHYSBASE8	MTRRphysBase8	If IA32_MTRRCAP[7:0] > 8
211H	529	IA32_MTRR_PHYSMASK8	MTRRphysMask8	If IA32_MTRRCAP[7:0] > 8
212H	530	IA32_MTRR_PHYSBASE9	MTRRphysBase9	If IA32_MTRRCAP[7:0] > 9
213H	531	IA32_MTRR_PHYSMASK9	MTRRphysMask9	If IA32_MTRRCAP[7:0] > 9
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	If CPUID.01H: EDX.MTRR[12] = 1
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	If CPUID.01H: EDX.MTRR[12] = 1
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	If CPUID.01H: EDX.MTRR[12] = 1
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	See Section 11.11.2.2, "Fixed Range MTRRs."	If CPUID.01H: EDX.MTRR[12] = 1
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	If CPUID.01H: EDX.MTRR[12] = 1
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	If CPUID.01H: EDX.MTRR[12] = 1
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	If CPUID.01H: EDX.MTRR[12] = 1
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	If CPUID.01H: EDX.MTRR[12] = 1
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	If CPUID.01H: EDX.MTRR[12] = 1
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	If CPUID.01H: EDX.MTRR[12] = 1
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	If CPUID.01H: EDX.MTRR[12] = 1
277H	631	IA32_PAT	IA32_PAT (R/W)	If CPUID.01H: EDX.MTRR[16] = 1
		2:0	PA0	
		7:3	Reserved	
		10:8	PA1	
		15:11	Reserved	
		18:16	PA2	
		23:19	Reserved	
		26:24	PA3	
	31:27	Reserved		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		34:32	PA4	
		39:35	Reserved	
		42:40	PA5	
		47:43	Reserved	
		50:48	PA6	
		55:51	Reserved	
		58:56	PA7	
		63:59	Reserved	
280H	640	IA32_MCO_CTL2	MSR to enable/disable CMCI capability for bank 0. (R/W) See Section 15.3.2.5, "IA32_MCi_CTL2 MSRs".	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 0
		14:0	Corrected error count threshold.	
		29:15	Reserved	
		30	CMCI_EN	
		63:31	Reserved	
281H	641	IA32_MC1_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
282H	642	IA32_MC2_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
283H	643	IA32_MC3_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
284H	644	IA32_MC4_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
285H	645	IA32_MC5_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
286H	646	IA32_MC6_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
287H	647	IA32_MC7_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
288H	648	IA32_MC8_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
289H	649	IA32_MC9_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
28AH	650	IA32_MC10_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
28BH	651	IA32_MC11_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
28CH	652	IA32_MC12_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
28DH	653	IA32_MC13_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
28EH	654	IA32_MC14_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
28FH	655	IA32_MC15_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
290H	656	IA32_MC16_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
291H	657	IA32_MC17_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
292H	658	IA32_MC18_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
293H	659	IA32_MC19_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
294H	660	IA32_MC20_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
295H	661	IA32_MC21_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
296H	662	IA32_MC22_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
297H	663	IA32_MC23_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
298H	664	IA32_MC24_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
299H	665	IA32_MC25_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
29AH	666	IA32_MC26_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
29BH	667	IA32_MC27_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
29CH	668	IA32_MC28_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
29DH	669	IA32_MC29_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
29EH	670	IA32_MC30_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
29FH	671	IA32_MC31_CTL2	(R/W) Same fields as IA32_MCO_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType (R/W)	If CPUID.01H: EDX.MTRR[12] = 1
		2:0	Default Memory Type	
		9:3	Reserved	
		10	Fixed Range MTRR Enable	
		11	MTRR Enable	
	63:12	Reserved		
309H	777	IA32_FIXED_CTR0	Fixed-Function Performance Counter 0 (R/W); Counts Instr_Retired.Any.	If CPUID.0AH: EDX[4:0] > 0
30AH	778	IA32_FIXED_CTR1	Fixed-Function Performance Counter 1 (R/W); Counts CPU_CLK_Unhalted.Core.	If CPUID.0AH: EDX[4:0] > 1
30BH	779	IA32_FIXED_CTR2	Fixed-Function Performance Counter 2 (R/W); Counts CPU_CLK_Unhalted.Ref.	If CPUID.0AH: EDX[4:0] > 2
345H	837	IA32_PERF_CAPABILITIES	Read Only MSR that enumerates the existence of performance monitoring features. (R/O)	If CPUID.01H: ECX[15] = 1
		5:0	LBR format	
		6	PEBS Trap	
		7	PEBSSaveArchRegs	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		11:8	PEBS Record Format	
		12	1: Freeze while SMM is supported.	
		13	1: Full width of counter writable via IA32_A_PMCx.	
		14	PEBS_BASELINE	
		15	1: Performance metrics available.	
		16	1: PEBS output will be written into the Intel PT trace stream.	If CPUID.0x7.0.EBX[25]=1
		63:17	Reserved	
38DH	909	IA32_FIXED_CTR_CTRL	Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.	If CPUID.0AH: EAX[7:0] > 1
		0	ENO_OS: Enable Fixed Counter 0 to count while CPL = 0.	
		1	ENO_Usr: Enable Fixed Counter 0 to count while CPL > 0.	
		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	ENO_PMI: Enable PMI when fixed counter 0 overflows.	
		4	EN1_OS: Enable Fixed Counter 1 to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1 to count while CPL > 0.	
		6	AnyThr1: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15]=0
		7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
		8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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	
38EH	910	IA32_PERF_GLOBAL_STATUS	Global Performance Counter Status (R/O)	If CPUID.0AH: EAX[7:0] > 0 (CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):ECX[0] = 1)
		0	Ovf_PMC0: Overflow status of IA32_PMC0.	If CPUID.0AH: EAX[15:8] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH: EAX[15:8] > 1
		2	Ovf_PMC2: Overflow status of IA32_PMC2.	If CPUID.0AH: EAX[15:8] > 2
		3	Ovf_PMC3: Overflow status of IA32_PMC3.	If CPUID.0AH: EAX[15:8] > 3
		n	Ovf_PMCn: Overflow status of IA32_PMCn.	If CPUID.0AH: EAX[15:8] > n
		31:n+1	Reserved	
		32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0.	If CPUID.0AH: EAX[7:0] > 1
		33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1.	If CPUID.0AH: EAX[7:0] > 1
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH: EAX[7:0] > 1
		47:35	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		48	Ovf_PERF_METRICS: If this bit is set, it indicates that PERF_METRIC counter has overflowed and a PMI is triggered; however, an overflow of fixed counter 3 should normally happen first. If this bit is clear no overflow occurred.	
		54:49	Reserved	
		55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer that was completely filled.	If CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):ECX[0] = 1
		57:56	Reserved	
		58	LBR_Frz. LBRs are frozen due to: <ul style="list-style-type: none"> ▪ IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1. ▪ The LBR stack overflowed. 	If CPUID.OAH: EAX[7:0] > 3
		59	CTR_Frz. Performance counters in the core PMU are frozen due to: <ul style="list-style-type: none"> ▪ IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1. ▪ One or more core PMU counters overflowed. 	If CPUID.OAH: 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.OAH: EAX[7:0] > 2
		62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.OAH: EAX[7:0] > 0
		63	CondChgd: Status bits of this register have changed.	If CPUID.OAH: EAX[7:0] > 0
38FH	911	IA32_PERF_GLOBAL_CTRL	Global Performance Counter Control (R/W) Counter increments while the result of ANDing the respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.	If CPUID.OAH: EAX[7:0] > 0
		0	EN_PMC0	If CPUID.OAH: EAX[15:8] > 0
		1	EN_PMC1	If CPUID.OAH: EAX[15:8] > 1
		2	EN_PMC2	If CPUID.OAH: EAX[15:8] > 2
		n	EN_PMCn	If CPUID.OAH: EAX[15:8] > n
		31:n+1	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		32	EN_FIXED_CTR0	If CPUID.0AH: EDX[4:0] > 0
		33	EN_FIXED_CTR1	If CPUID.0AH: EDX[4:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EDX[4:0] > 2
		47:35	Reserved	
		48	EN_PERF_METRICS: If this bit is set and fixed counter 3 is effectively enabled, built-in performance metrics are enabled.	
		63:49	Reserved	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0 && CPUID.0AH: EAX[7:0] <= 3
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EDX[4:0] > 2
		54:35	Reserved	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If (CPUID.(EAX=07H, ECX=0);EBX[25] = 1) && IA32_RTIT_CTL.ToPA = 1
		60:56	Reserved	
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf bit.	If CPUID.0AH: EAX[7:0] > 0
63	Set 1 to clear CondChgd bit.	If CPUID.0AH: EAX[7:0] > 0		
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Global Performance Counter Overflow Reset Control (R/W)	If CPUID.0AH: EAX[7:0] > 3 (CPUID.(EAX=07H, ECX=0);EBX[25] = 1 && CPUID.(EAX=014H, ECX=0);ECX[0] = 1)
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH: EAX[15:8] > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EDX[4:0] > 2
		47:35	Reserved	
		48	RESET_OVF_PERF_METRICS: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
		54:49	Reserved	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):ECX[0] = 1
		57:56	Reserved	
		58	Set 1 to Clear LBR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to Clear CTR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to Clear ASCII bit.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf bit.	If CPUID.0AH: EAX[7:0] > 0
		63	Set 1 to clear CondChgd bit.	If CPUID.0AH: EAX[7:0] > 0
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Global Performance Counter Overflow Set Control (R/W)	If CPUID.0AH: EAX[7:0] > 3 (CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):ECX[0] = 1)
		0	Set 1 to cause Ovf_PMC0 = 1.	If CPUID.0AH: EAX[7:0] > 3
		1	Set 1 to cause Ovf_PMC1 = 1.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to cause Ovf_PMC2 = 1.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to cause Ovf_PMCn = 1.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved	
		32	Set 1 to cause Ovf_FIXED_CTR0 = 1.	If CPUID.0AH: EAX[7:0] > 3
		33	Set 1 to cause Ovf_FIXED_CTR1 = 1.	If CPUID.0AH: EAX[7:0] > 3
		34	Set 1 to cause Ovf_FIXED_CTR2 = 1.	If CPUID.0AH: EAX[7:0] > 3
		47:35	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.OAH: EAX[7:0] > 3
		59	Set 1 to cause CTR_Frz = 1.	If CPUID.OAH: EAX[7:0] > 3
		58	Set 1 to cause ASCII = 1.	If CPUID.OAH: EAX[7:0] > 3
		61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.OAH: EAX[7:0] > 3
		62	Set 1 to cause OvfBuf = 1.	If CPUID.OAH: EAX[7:0] > 3
		63	Reserved	
392H	914	IA32_PERF_GLOBAL_INUSE	Indicator that core perfmon interface is in use. (R/O)	If CPUID.OAH: EAX[7:0] > 3
		0	IA32_PERFEVTSEL0 in use.	
		1	IA32_PERFEVTSEL1 in use.	If CPUID.OAH: EAX[15:8] > 1
		2	IA32_PERFEVTSEL2 in use.	If CPUID.OAH: EAX[15:8] > 2
		n	IA32_PERFEVTSELn in use.	If CPUID.OAH: EAX[15:8] > n
		31:n+1	Reserved	
		32	IA32_FIXED_CTR0 in use.	
		33	IA32_FIXED_CTR1 in use.	
		34	IA32_FIXED_CTR2 in use.	
		62:35	Reserved or model specific.	
63	PMI in use.			
3F1H	1009	IA32_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	
400H	1024	IA32_MCO_CTL	MCO_CTL	If IA32_MCG_CAP.CNT > 0
401H	1025	IA32_MCO_STATUS	MCO_STATUS	If IA32_MCG_CAP.CNT > 0
402H	1026	IA32_MCO_ADDR ¹	MCO_ADDR	If IA32_MCG_CAP.CNT > 0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
403H	1027	IA32_MC0_MISC	MC0_MISC	If IA32_MCG_CAP.CNT >0
404H	1028	IA32_MC1_CTL	MC1_CTL	If IA32_MCG_CAP.CNT >1
405H	1029	IA32_MC1_STATUS	MC1_STATUS	If IA32_MCG_CAP.CNT >1
406H	1030	IA32_MC1_ADDR ²	MC1_ADDR	If IA32_MCG_CAP.CNT >1
407H	1031	IA32_MC1_MISC	MC1_MISC	If IA32_MCG_CAP.CNT >1
408H	1032	IA32_MC2_CTL	MC2_CTL	If IA32_MCG_CAP.CNT >2
409H	1033	IA32_MC2_STATUS	MC2_STATUS	If IA32_MCG_CAP.CNT >2
40AH	1034	IA32_MC2_ADDR ¹	MC2_ADDR	If IA32_MCG_CAP.CNT >2
40BH	1035	IA32_MC2_MISC	MC2_MISC	If IA32_MCG_CAP.CNT >2
40CH	1036	IA32_MC3_CTL	MC3_CTL	If IA32_MCG_CAP.CNT >3
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	If IA32_MCG_CAP.CNT >3
40EH	1038	IA32_MC3_ADDR ¹	MC3_ADDR	If IA32_MCG_CAP.CNT >3
40FH	1039	IA32_MC3_MISC	MC3_MISC	If IA32_MCG_CAP.CNT >3
410H	1040	IA32_MC4_CTL	MC4_CTL	If IA32_MCG_CAP.CNT >4
411H	1041	IA32_MC4_STATUS	MC4_STATUS	If IA32_MCG_CAP.CNT >4
412H	1042	IA32_MC4_ADDR ¹	MC4_ADDR	If IA32_MCG_CAP.CNT >4
413H	1043	IA32_MC4_MISC	MC4_MISC	If IA32_MCG_CAP.CNT >4
414H	1044	IA32_MC5_CTL	MC5_CTL	If IA32_MCG_CAP.CNT >5
415H	1045	IA32_MC5_STATUS	MC5_STATUS	If IA32_MCG_CAP.CNT >5
416H	1046	IA32_MC5_ADDR ¹	MC5_ADDR	If IA32_MCG_CAP.CNT >5
417H	1047	IA32_MC5_MISC	MC5_MISC	If IA32_MCG_CAP.CNT >5
418H	1048	IA32_MC6_CTL	MC6_CTL	If IA32_MCG_CAP.CNT >6
419H	1049	IA32_MC6_STATUS	MC6_STATUS	If IA32_MCG_CAP.CNT >6
41AH	1050	IA32_MC6_ADDR ¹	MC6_ADDR	If IA32_MCG_CAP.CNT >6
41BH	1051	IA32_MC6_MISC	MC6_MISC	If IA32_MCG_CAP.CNT >6
41CH	1052	IA32_MC7_CTL	MC7_CTL	If IA32_MCG_CAP.CNT >7
41DH	1053	IA32_MC7_STATUS	MC7_STATUS	If IA32_MCG_CAP.CNT >7
41EH	1054	IA32_MC7_ADDR ¹	MC7_ADDR	If IA32_MCG_CAP.CNT >7
41FH	1055	IA32_MC7_MISC	MC7_MISC	If IA32_MCG_CAP.CNT >7
420H	1056	IA32_MC8_CTL	MC8_CTL	If IA32_MCG_CAP.CNT >8
421H	1057	IA32_MC8_STATUS	MC8_STATUS	If IA32_MCG_CAP.CNT >8
422H	1058	IA32_MC8_ADDR ¹	MC8_ADDR	If IA32_MCG_CAP.CNT >8
423H	1059	IA32_MC8_MISC	MC8_MISC	If IA32_MCG_CAP.CNT >8
424H	1060	IA32_MC9_CTL	MC9_CTL	If IA32_MCG_CAP.CNT >9
425H	1061	IA32_MC9_STATUS	MC9_STATUS	If IA32_MCG_CAP.CNT >9
426H	1062	IA32_MC9_ADDR ¹	MC9_ADDR	If IA32_MCG_CAP.CNT >9
427H	1063	IA32_MC9_MISC	MC9_MISC	If IA32_MCG_CAP.CNT >9

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
428H	1064	IA32_MC10_CTL	MC10_CTL	If IA32_MCG_CAP.CNT >10
429H	1065	IA32_MC10_STATUS	MC10_STATUS	If IA32_MCG_CAP.CNT >10
42AH	1066	IA32_MC10_ADDR ¹	MC10_ADDR	If IA32_MCG_CAP.CNT >10
42BH	1067	IA32_MC10_MISC	MC10_MISC	If IA32_MCG_CAP.CNT >10
42CH	1068	IA32_MC11_CTL	MC11_CTL	If IA32_MCG_CAP.CNT >11
42DH	1069	IA32_MC11_STATUS	MC11_STATUS	If IA32_MCG_CAP.CNT >11
42EH	1070	IA32_MC11_ADDR ¹	MC11_ADDR	If IA32_MCG_CAP.CNT >11
42FH	1071	IA32_MC11_MISC	MC11_MISC	If IA32_MCG_CAP.CNT >11
430H	1072	IA32_MC12_CTL	MC12_CTL	If IA32_MCG_CAP.CNT >12
431H	1073	IA32_MC12_STATUS	MC12_STATUS	If IA32_MCG_CAP.CNT >12
432H	1074	IA32_MC12_ADDR ¹	MC12_ADDR	If IA32_MCG_CAP.CNT >12
433H	1075	IA32_MC12_MISC	MC12_MISC	If IA32_MCG_CAP.CNT >12
434H	1076	IA32_MC13_CTL	MC13_CTL	If IA32_MCG_CAP.CNT >13
435H	1077	IA32_MC13_STATUS	MC13_STATUS	If IA32_MCG_CAP.CNT >13
436H	1078	IA32_MC13_ADDR ¹	MC13_ADDR	If IA32_MCG_CAP.CNT >13
437H	1079	IA32_MC13_MISC	MC13_MISC	If IA32_MCG_CAP.CNT >13
438H	1080	IA32_MC14_CTL	MC14_CTL	If IA32_MCG_CAP.CNT >14
439H	1081	IA32_MC14_STATUS	MC14_STATUS	If IA32_MCG_CAP.CNT >14
43AH	1082	IA32_MC14_ADDR ¹	MC14_ADDR	If IA32_MCG_CAP.CNT >14
43BH	1083	IA32_MC14_MISC	MC14_MISC	If IA32_MCG_CAP.CNT >14
43CH	1084	IA32_MC15_CTL	MC15_CTL	If IA32_MCG_CAP.CNT >15
43DH	1085	IA32_MC15_STATUS	MC15_STATUS	If IA32_MCG_CAP.CNT >15
43EH	1086	IA32_MC15_ADDR ¹	MC15_ADDR	If IA32_MCG_CAP.CNT >15
43FH	1087	IA32_MC15_MISC	MC15_MISC	If IA32_MCG_CAP.CNT >15
440H	1088	IA32_MC16_CTL	MC16_CTL	If IA32_MCG_CAP.CNT >16
441H	1089	IA32_MC16_STATUS	MC16_STATUS	If IA32_MCG_CAP.CNT >16
442H	1090	IA32_MC16_ADDR ¹	MC16_ADDR	If IA32_MCG_CAP.CNT >16
443H	1091	IA32_MC16_MISC	MC16_MISC	If IA32_MCG_CAP.CNT >16
444H	1092	IA32_MC17_CTL	MC17_CTL	If IA32_MCG_CAP.CNT >17
445H	1093	IA32_MC17_STATUS	MC17_STATUS	If IA32_MCG_CAP.CNT >17
446H	1094	IA32_MC17_ADDR ¹	MC17_ADDR	If IA32_MCG_CAP.CNT >17
447H	1095	IA32_MC17_MISC	MC17_MISC	If IA32_MCG_CAP.CNT >17
448H	1096	IA32_MC18_CTL	MC18_CTL	If IA32_MCG_CAP.CNT >18
449H	1097	IA32_MC18_STATUS	MC18_STATUS	If IA32_MCG_CAP.CNT >18
44AH	1098	IA32_MC18_ADDR ¹	MC18_ADDR	If IA32_MCG_CAP.CNT >18
44BH	1099	IA32_MC18_MISC	MC18_MISC	If IA32_MCG_CAP.CNT >18
44CH	1100	IA32_MC19_CTL	MC19_CTL	If IA32_MCG_CAP.CNT >19

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
44DH	1101	IA32_MC19_STATUS	MC19_STATUS	If IA32_MCG_CAP.CNT >19
44EH	1102	IA32_MC19_ADDR ¹	MC19_ADDR	If IA32_MCG_CAP.CNT >19
44FH	1103	IA32_MC19_MISC	MC19_MISC	If IA32_MCG_CAP.CNT >19
450H	1104	IA32_MC20_CTL	MC20_CTL	If IA32_MCG_CAP.CNT >20
451H	1105	IA32_MC20_STATUS	MC20_STATUS	If IA32_MCG_CAP.CNT >20
452H	1106	IA32_MC20_ADDR ¹	MC20_ADDR	If IA32_MCG_CAP.CNT >20
453H	1107	IA32_MC20_MISC	MC20_MISC	If IA32_MCG_CAP.CNT >20
454H	1108	IA32_MC21_CTL	MC21_CTL	If IA32_MCG_CAP.CNT >21
455H	1109	IA32_MC21_STATUS	MC21_STATUS	If IA32_MCG_CAP.CNT >21
456H	1110	IA32_MC21_ADDR ¹	MC21_ADDR	If IA32_MCG_CAP.CNT >21
457H	1111	IA32_MC21_MISC	MC21_MISC	If IA32_MCG_CAP.CNT >21
458H	1112	IA32_MC22_CTL	MC22_CTL	If IA32_MCG_CAP.CNT >22
459H	1113	IA32_MC22_STATUS	MC22_STATUS	If IA32_MCG_CAP.CNT >22
45AH	1114	IA32_MC22_ADDR ¹	MC22_ADDR	If IA32_MCG_CAP.CNT >22
45BH	1115	IA32_MC22_MISC	MC22_MISC	If IA32_MCG_CAP.CNT >22
45CH	1116	IA32_MC23_CTL	MC23_CTL	If IA32_MCG_CAP.CNT >23
45DH	1117	IA32_MC23_STATUS	MC23_STATUS	If IA32_MCG_CAP.CNT >23
45EH	1118	IA32_MC23_ADDR ¹	MC23_ADDR	If IA32_MCG_CAP.CNT >23
45FH	1119	IA32_MC23_MISC	MC23_MISC	If IA32_MCG_CAP.CNT >23
460H	1120	IA32_MC24_CTL	MC24_CTL	If IA32_MCG_CAP.CNT >24
461H	1121	IA32_MC24_STATUS	MC24_STATUS	If IA32_MCG_CAP.CNT >24
462H	1122	IA32_MC24_ADDR ¹	MC24_ADDR	If IA32_MCG_CAP.CNT >24
463H	1123	IA32_MC24_MISC	MC24_MISC	If IA32_MCG_CAP.CNT >24
464H	1124	IA32_MC25_CTL	MC25_CTL	If IA32_MCG_CAP.CNT >25
465H	1125	IA32_MC25_STATUS	MC25_STATUS	If IA32_MCG_CAP.CNT >25
466H	1126	IA32_MC25_ADDR ¹	MC25_ADDR	If IA32_MCG_CAP.CNT >25
467H	1127	IA32_MC25_MISC	MC25_MISC	If IA32_MCG_CAP.CNT >25
468H	1128	IA32_MC26_CTL	MC26_CTL	If IA32_MCG_CAP.CNT >26
469H	1129	IA32_MC26_STATUS	MC26_STATUS	If IA32_MCG_CAP.CNT >26
46AH	1130	IA32_MC26_ADDR ¹	MC26_ADDR	If IA32_MCG_CAP.CNT >26
46BH	1131	IA32_MC26_MISC	MC26_MISC	If IA32_MCG_CAP.CNT >26
46CH	1132	IA32_MC27_CTL	MC27_CTL	If IA32_MCG_CAP.CNT >27
46DH	1133	IA32_MC27_STATUS	MC27_STATUS	If IA32_MCG_CAP.CNT >27
46EH	1134	IA32_MC27_ADDR ¹	MC27_ADDR	If IA32_MCG_CAP.CNT >27
46FH	1135	IA32_MC27_MISC	MC27_MISC	If IA32_MCG_CAP.CNT >27
470H	1136	IA32_MC28_CTL	MC28_CTL	If IA32_MCG_CAP.CNT >28
471H	1137	IA32_MC28_STATUS	MC28_STATUS	If IA32_MCG_CAP.CNT >28

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
472H	1138	IA32_MC28_ADDR ¹	MC28_ADDR	If IA32_MCG_CAP.CNT >28
473H	1139	IA32_MC28_MISC	MC28_MISC	If IA32_MCG_CAP.CNT >28
480H	1152	IA32_VMX_BASIC	Reporting Register of Basic VMX Capabilities (R/O) See Appendix A.1, "Basic VMX Information."	If CPUID.01H:ECX.[5] = 1
481H	1153	IA32_VMX_PINBASED_CTL5	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
482H	1154	IA32_VMX_PROCBASED_CTL5	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
483H	1155	IA32_VMX_EXIT_CTL5	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
484H	1156	IA32_VMX_ENTRY_CTL5	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If CPUID.01H:ECX.[5] = 1
485H	1157	IA32_VMX_MISC	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data."	If CPUID.01H:ECX.[5] = 1
486H	1158	IA32_VMX_CRO_FIXED0	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CRO."	If CPUID.01H:ECX.[5] = 1
487H	1159	IA32_VMX_CRO_FIXED1	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CRO."	If CPUID.01H:ECX.[5] = 1
488H	1160	IA32_VMX_CR4_FIXED0	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[5] = 1
489H	1161	IA32_VMX_CR4_FIXED1	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."	If CPUID.01H:ECX.[5] = 1
48AH	1162	IA32_VMX_VMCS_ENUM	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration."	If CPUID.01H:ECX.[5] = 1
48BH	1163	IA32_VMX_PROCBASED_CTL52	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."	If (CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTL5[63])

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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_C TLS[63] && (IA32_VMX_PROCBASED_C TLS2[33] IA32_VMX_PROCBASED_C TLS2[37]))
48DH	1165	IA32_VMX_TRUE_PINBASED_CTL	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] = 1 && IA32_VMX_BASIC[55])
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTL	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] = 1 && IA32_VMX_BASIC[55])
48FH	1167	IA32_VMX_TRUE_EXIT_CTL	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Appendix A.4, "VM-Exit Controls."	If (CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
490H	1168	IA32_VMX_TRUE_ENTRY_CTL	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If (CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
491H	1169	IA32_VMX_VMFUNC	Capability Reporting Register of VM-Function Controls (R/O)	If (CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
492H	1170	IA32_VMX_PROCBASED_C TLS3	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_C TLS[49])
493H	1171	IA32_VMX_EXIT_C TLS2	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_C TLS[63])
4C1H	1217	IA32_A_PMC0	Full Width Writable IA32_PMC0 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 0) && IA32_PERF_CAPABILITIES[13] = 1
4C2H	1218	IA32_A_PMC1	Full Width Writable IA32_PMC1 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 1) && IA32_PERF_CAPABILITIES[13] = 1
4C3H	1219	IA32_A_PMC2	Full Width Writable IA32_PMC2 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 2) && IA32_PERF_CAPABILITIES[13] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
4C4H	1220	IA32_A_PMC3	Full Width Writable IA32_PMC3 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 3) && IA32_PERF_CAPABILITIES[13] = 1
4C5H	1221	IA32_A_PMC4	Full Width Writable IA32_PMC4 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 4) && IA32_PERF_CAPABILITIES[13] = 1
4C6H	1222	IA32_A_PMC5	Full Width Writable IA32_PMC5 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 5) && IA32_PERF_CAPABILITIES[13] = 1
4C7H	1223	IA32_A_PMC6	Full Width Writable IA32_PMC6 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 6) && IA32_PERF_CAPABILITIES[13] = 1
4C8H	1224	IA32_A_PMC7	Full Width Writable IA32_PMC7 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 7) && IA32_PERF_CAPABILITIES[13] = 1
4D0H	1232	IA32_MCG_EXT_CTL	Allows software to signal some MCEs to only a single logical processor in the system. (R/W) See Section 15.3.1.4, "IA32_MCG_EXT_CTL MSR".	If IA32_MCG_CAP.LMCE_P = 1
		0	LMCE_EN	
		63:1	Reserved	
500H	1280	IA32_SGX_SVN_STATUS	Status and SVN Threshold of SGX Support for ACM (R/O).	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		0	Lock	See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1	Reserved	
		23:16	SGX_SVN_SINIT	See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		63:24	Reserved	
560H	1376	IA32_RTIT_OUTPUT_BASE	Trace Output Base Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && ((CPUID.(EAX=14H, ECX=0):ECX[0] = 1) (CPUID.(EAX=14H, ECX=0):ECX[2] = 1)))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6:0	Reserved	
		MAXPHYADDR ⁴ -1:7	Base physical address.	
		63:MAXPHYADDR	Reserved	
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Trace Output Mask Pointers Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && (CPUID.(EAX=14H, ECX=0):ECX[0] = 1) (CPUID.(EAX=14H, ECX=0):ECX[2] = 1))
		6:0	Reserved	
		31:7	MaskOrTableOffset	
		63:32	Output Offset	
570H	1392	IA32_RTIT_CTL	Trace Control Register (R/W)	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		0	TraceEn	
		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	CR3 filter	
		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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		27:24	PSBFreq	If (CPUID.(EAX=07H, ECX=0);EBX[1] = 1)
		31:28	Reserved, must be zero.	
		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)
		55:48	Reserved, must be zero.	
		56	InjectPsbPmiOnEnable	If (CPUID.(EAX=07H, ECX=1);EBX[6] = 1)
		63:57	Reserved, must be zero.	
571H	1393	IA32_RTIT_STATUS	Tracing Status Register (R/W)	If (CPUID.(EAX=07H, ECX=0);EBX[25] = 1)
		0	FilterEn (writes ignored)	If (CPUID.(EAX=07H, ECX=0);EBX[2] = 1)
		1	ContexEn (writes ignored)	
		2	TriggerEn (writes ignored)	
		3	Reserved	
		4	Error	
		5	Stopped	
		6	PendPSB	If (CPUID.(EAX=07H, ECX=0);EBX[6] = 1)
		7	PendToPAPMI	If (CPUID.(EAX=07H, ECX=0);EBX[6] = 1)
		31:8	Reserved, must be zero.	
		48:32	PacketByteCnt	If (CPUID.(EAX=07H, ECX=0);EBX[1] > 3)
		63:49	Reserved	
		572H	1394	IA32_RTIT_CR3_MATCH
4:0	Reserved			
63:5	CR3[63:5] value to match.			
580H	1408	IA32_RTIT_ADDRO_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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
581H	1409	IA32_RTIT_ADDR0_B	Region 0 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
582H	1410	IA32_RTIT_ADDR1_A	Region 1 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	
583H	1411	IA32_RTIT_ADDR1_B	Region 1 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	
584H	1412	IA32_RTIT_ADDR2_A	Region 2 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	
		63:48	SignExt_VA	
585H	1413	IA32_RTIT_ADDR2_B	Region 2 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	
		63:48	SignExt_VA	
586H	1414	IA32_RTIT_ADDR3_A	Region 3 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		47:0	Virtual Address	
		63:48	SignExt_VA	
587H	1415	IA32_RTIT_ADDR3_B	Region 3 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		47:0	Virtual Address	
		63:48	SignExt_VA	
600H	1536	IA32_DS_AREA	DS Save Area (R/W) Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 19.6.3.4, "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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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.	
		10	SUPPRESS: When set to 1, indirect branch tracking is suppressed. This bit can be written to 1 only if TRACKER is written as IDLE.	
		11	TRACKER: Value of the indirect branch tracking state machine. Values: IDLE (0), WAIT_FOR_ENDBRANCH(1).	
	63:12	EB_LEG_BITMAP_BASE: Linear address bits 63:12 of a legacy code page bitmap used for legacy compatibility when indirect branch tracking is enabled. If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).		
6A2H	1698	IA32_S_CET	Configure Supervisor Mode CET (R/W)	See IA32_U_CET (6A0H) for reference; similar format.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
6A4H	1700	IA32_PLO_SSP	Linear address to be loaded into SSP on transition to privilege level 0. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6A5H	1701	IA32_PL1_SSP	Linear address to be loaded into SSP on transition to privilege level 1. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6A6H	1702	IA32_PL2_SSP	Linear address to be loaded into SSP on transition to privilege level 2. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6A7H	1703	IA32_PL3_SSP	Linear address to be loaded into SSP on transition to privilege level 3. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
6A8H	1704	IA32_INTERRUPT_SSP_TABLE_ADDR	Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) This MSR is not present on processors that do not support Intel 64 architecture. This field cannot represent a non-canonical address.	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
6E0H	1760	IA32_TSC_DEADLINE	TSC Target of Local APIC's TSC Deadline Mode (R/W)	If CPUID.01H:ECX.[24] = 1
6E1H	1761	IA32_PKRS	Specifies the PK permissions associated with each protection domain for supervisor pages (R/W)	If CPUID.(EAX=07H, ECX=0H):ECX.PKS [31] = 1
		31:0	For domain i (i between 0 and 15), bits 2i and 2i+1 contain the AD and WD permissions, respectively.	
		63:32	Reserved.	
770H	1904	IA32_PM_ENABLE	Enable/disable HWP (R/W)	If CPUID.06H:EAX.[7] = 1
		0	HWP_ENABLE (R/W1-Once) See Section 14.4.2, "Enabling HWP".	
		63:1	Reserved	
771H	1905	IA32_HWP_CAPABILITIES	HWP Performance Range Enumeration (R/O)	If CPUID.06H:EAX.[7] = 1
		7:0	Highest_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	
		15:8	Guaranteed_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	
		23:16	Most_Efficient_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	
		31:24	Lowest_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	
		63:32	Reserved	
772H	1906	IA32_HWP_REQUEST_PKG	Power Management Control Hints for All Logical Processors in a Package (R/W)	If CPUID.06H:EAX.[11] = 1
		7:0	Minimum_Performance See Section 14.4.4, "Managing HWP".	
		15:8	Maximum_Performance See Section 14.4.4, "Managing HWP".	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		23:16	Desired_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1
		31:24	Energy_Performance_Preference See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	Activity_Window See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[9] = 1
		63:42	Reserved	
773H	1907	IA32_HWP_INTERRUPT	Control HWP Native Interrupts (R/W)	If CPUID.06H:EAX.[8] = 1
		0	EN_Guaranteed_Performance_Change See Section 14.4.6, "HWP Notifications".	If CPUID.06H:EAX.[8] = 1
		1	EN_Excursion_Minimum See Section 14.4.6, "HWP Notifications".	If CPUID.06H:EAX.[8] = 1
		63:2	Reserved	
774H	1908	IA32_HWP_REQUEST	Power Management Control Hints to a Logical Processor (R/W)	If CPUID.06H:EAX.[7] = 1
		7:0	Minimum_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1
		15:8	Maximum_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1
		23:16	Desired_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1
		31:24	Energy_Performance_Preference See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	Activity_Window See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[9] = 1
		42	Package_Control See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[11] = 1
		63:43	Reserved	
775H	1909	IA32_PECI_HWP_REQUEST_INFO	IA32_PECI_HWP_REQUEST_INFO	
		7:0	Minimum Performance (MINIMUM_PERFORMANCE): Used by OS to read the latest value of Peci minimum performance input. Default value is 0.	
		15:8	Maximum Performance (MAXIMUM_PERFORMANCE): Used by OS to read the latest value of Peci maximum performance input. Default value is 0.	
		23:16	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		31:24	Energy Performance Preference (ENERGY_PERFORMANCE_PREFERENCE): Used by OS to read the latest value of PECC Energy Performance Preference input. Default value is 0.	
		59:32	Reserved.	
		60	EPP PECC Override (EPP_PECC_OVERRIDE): Indicates whether PECC is currently overriding the Energy Performance Preference input. If set to '1', PECC 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 PECC Override (MAX_PECC_OVERRIDE): Indicates whether PECC is currently overriding the Maximum Performance input. If set to '1', PECC is overriding the Maximum Performance input. If clear (0), OS has control over Maximum Performance input. Default value is 0.	
		63	Min PECC Override (MIN_PECC_OVERRIDE): Indicates whether PECC is currently overriding the Minimum Performance input. If set to '1', PECC is overriding the Minimum Performance input. If clear (0), OS has control over Minimum Performance input. Default value is 0.	
776H	1910	IA32_HWP_CTL	IA32_HWP_CTL	If CPUID.06H:EAX.[22] = 1
		0	PKG_CTL_POLARITY Defines which HWP Request MSR is used whether Thread level or package level. When package MSR is used, the thread MSR valid bits define which thread MSR fields override the package. Default value is 0.	If CPUID.06H:EAX.[22] = 1
		63:1	Reserved	
777H	1911	IA32_HWP_STATUS	Log bits indicating changes to Guaranteed & excursions to Minimum (R/W)	If CPUID.06H:EAX.[7] = 1
		0	Guaranteed_Performance_Change (R/WCO) See Section 14.4.5, "HWP Feedback".	If CPUID.06H:EAX.[7] = 1
		1	Reserved	
		2	Excursion_To_Minimum (R/WCO) See Section 14.4.5, "HWP Feedback".	If CPUID.06H:EAX.[7] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:3	Reserved	
802H	2050	IA32_X2APIC_APICID	x2APIC ID Register (R/O) See x2APIC Specification.	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
803H	2051	IA32_X2APIC_VERSION	x2APIC Version Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
808H	2056	IA32_X2APIC_TPR	x2APIC Task Priority Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80AH	2058	IA32_X2APIC_PPR	x2APIC Processor Priority Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80BH	2059	IA32_X2APIC_EOI	x2APIC EOI Register (W/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80DH	2061	IA32_X2APIC_LDR	x2APIC Logical Destination Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80FH	2063	IA32_X2APIC_SIVR	x2APIC Spurious Interrupt Vector Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
810H	2064	IA32_X2APIC_ISR0	x2APIC In-Service Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
811H	2065	IA32_X2APIC_ISR1	x2APIC In-Service Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
812H	2066	IA32_X2APIC_ISR2	x2APIC In-Service Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
813H	2067	IA32_X2APIC_ISR3	x2APIC In-Service Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
814H	2068	IA32_X2APIC_ISR4	x2APIC In-Service Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
815H	2069	IA32_X2APIC_ISR5	x2APIC In-Service Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
816H	2070	IA32_X2APIC_ISR6	x2APIC In-Service Register Bits 223:192 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
817H	2071	IA32_X2APIC_ISR7	x2APIC In-Service Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
819H	2073	IA32_X2APIC_TMR1	x2APIC Trigger Mode Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81AH	2074	IA32_X2APIC_TMR2	x2APIC Trigger Mode Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81BH	2075	IA32_X2APIC_TMR3	x2APIC Trigger Mode Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81CH	2076	IA32_X2APIC_TMR4	x2APIC Trigger Mode Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81DH	2077	IA32_X2APIC_TMR5	x2APIC Trigger Mode Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81EH	2078	IA32_X2APIC_TMR6	x2APIC Trigger Mode Register Bits 223:192 (R/O)	If (CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1)
81FH	2079	IA32_X2APIC_TMR7	x2APIC Trigger Mode Register Bits 255:224 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
820H	2080	IA32_X2APIC_IRR0	x2APIC Interrupt Request Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
821H	2081	IA32_X2APIC_IRR1	x2APIC Interrupt Request Register Bits 63:32 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
822H	2082	IA32_X2APIC_IRR2	x2APIC Interrupt Request Register Bits 95:64 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
823H	2083	IA32_X2APIC_IRR3	x2APIC Interrupt Request Register Bits 127:96 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
824H	2084	IA32_X2APIC_IRR4	x2APIC Interrupt Request Register Bits 159:128 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
825H	2085	IA32_X2APIC_IRR5	x2APIC Interrupt Request Register Bits 191:160 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
826H	2086	IA32_X2APIC_IRR6	x2APIC Interrupt Request Register Bits 223:192 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
828H	2088	IA32_X2APIC_ESR	x2APIC Error Status Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
82FH	2095	IA32_X2APIC_LVT_CMCI	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
830H	2096	IA32_X2APIC_ICR	x2APIC Interrupt Command Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
832H	2098	IA32_X2APIC_LVT_TIMER	x2APIC LVT Timer Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
833H	2099	IA32_X2APIC_LVT_THERMAL	x2APIC LVT Thermal Sensor Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
834H	2100	IA32_X2APIC_LVT_PMI	x2APIC LVT Performance Monitor Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
835H	2101	IA32_X2APIC_LVT_LINT0	x2APIC LVT LINT0 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
836H	2102	IA32_X2APIC_LVT_LINT1	x2APIC LVT LINT1 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
837H	2103	IA32_X2APIC_LVT_ERROR	x2APIC LVT Error Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
838H	2104	IA32_X2APIC_INIT_COUNT	x2APIC Initial Count Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
839H	2105	IA32_X2APIC_CUR_COUNT	x2APIC Current Count Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
83EH	2110	IA32_X2APIC_DIV_CONF	x2APIC Divide Configuration Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
83FH	2111	IA32_X2APIC_SELF_IPI	x2APIC Self IPI Register (W/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
981H	2433	IA32_TME_CAPABILITY	Memory Encryption Capability MSR	If CPUID.07H:ECX.[13] = 1
		0	Support for AES-XTS 128-bit encryption algorithm. (NIST standard)	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		1	Support for AES-XTS 128-bit encryption with integrity algorithm.	
		2	Support for AES-XTS 256-bit encryption algorithm.	
		30:3	Reserved.	
		31	TME encryption bypass supported.	
		35:32	MK_TME_MAX_KEYID_BITS Number of bits which can be allocated for usage as key identifiers for multi-key memory encryption. 4 bits allow for a maximum value of 15, which could address 32K keys. Zero if MKTME is not supported.	
		50:36	MK_TME_MAX_KEYS Indicates the maximum number of keys which are available for usage. This value may not be a power of 2. KeyID 0 is specially reserved and is not accounted for in this field.	
		63:51	Reserved.	
982H	2434	IA32_TME_ACTIVATE	Memory Encryption Activation MSR This MSR is used to lock the MSRs listed below. Any write to the following MSRs will be ignored after they are locked. The lock is reset when CPU is reset. <ul style="list-style-type: none"> ▪ IA32_TME_ACTIVATE ▪ IA32_TME_EXCLUDE_MASK ▪ IA32_TME_EXCLUDE_BASE Note that IA32_TME_EXCLUDE_MASK and IA32_TME_EXCLUDE_BASE must be configured before IA32_TME_ACTIVATE.	If CPUID.07H:ECX.[13] = 1
		0	Lock R/O - Will be set upon successful WRMSR (or first SMI); written value ignored.	
		1	Hardware Encryption Enable This bit also enables MKTME; MKTME cannot be enabled without enabling encryption hardware.	
		2	Key Select 0: Create a new TME key (expected cold/warm boot). 1: Restore the TME key from storage (Expected when resume from standby).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		3	Save TME Key for Standby Save key into storage to be used when resume from standby. Note: This may not be supported in all processors.	
		7:4	TME Policy/Encryption Algorithm Only algorithms enumerated in IA32_TME_CAPABILITY are allowed. For example: 0000 - AES-XTS-128. 0001 - AES-XTS-128 with integrity. 0010 - AES-XTS-256. Other values are invalid.	
		30:8	Reserved.	
		31	TME Encryption Bypass Enable When encryption hardware is enabled: <ul style="list-style-type: none"> ▪ Total Memory Encryption is enabled using a CPU generated ephemeral key based on a hardware random number generator when this bit is set to 0. ▪ Total Memory Encryption is bypassed (no encryption/decryption for KeyID0) when this bit is set to 1. Software must inspect Hardware Encryption Enable (bit 1) and TME encryption bypass Enable (bit 31) to determine if TME encryption is enabled.	
		35:32	MK_TME_KEYID_BITS Reserved if MKTME is not enumerated, otherwise: The number of key identifier bits to allocate to MKTME usage. Similar to enumeration, this is an encoded value. Writing a value greater than MK_TME_MAX_KEYID_BITS will result in #GP. Writing a non-zero value to this field will #GP if bit 1 of EAX (Hardware Encryption Enable) is not also set to '1, as encryption hardware must be enabled to use MKTME. Example: To support 255 keys, this field would be set to a value of 8.	
		47:36	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:48	MK_TME_CRYPTO_ALGS Reserved if MKTME is not enumerated, otherwise: Bit 48: AES-XTS 128. Bit 49: AES-XTS 128 with integrity. Bit 50: AES-XTS 256. Bit 63:51: Reserved (#GP) Bitmask for BIOS to set which encryption algorithms are allowed for MKTME, would be later enforced by the key loading ISA ('1 = allowed).	
983H	2435	IA32_TME_EXCLUDE_MASK	Memory Encryption Exclude Mask	If CPUID.07H:ECX.[13] = 1
		10:0	Reserved.	
		11	Enable: When set to '1', then TME_EXCLUDE_BASE and TME_EXCLUDE_MASK are used to define an exclusion region for TME/MKTME (for KeyID=0).	
		MAXPHYSADDR-1:12	TMEEMASK: This field indicates the bits that must match TMEEBASE in order to qualify as a TME/MKTME (for KeyID=0) exclusion memory range access.	
		63:MAXPHYSADDR	Reserved; must be zero.	
984H	2436	IA32_TME_EXCLUDE_BASE	Memory Encryption Exclude Base	If CPUID.07H:ECX.[13] = 1
		11:0	Reserved.	
		MAXPHYSADDR-1:12	TMEEBASE: Base physical address to be excluded for TME/MKTME (for KeyID=0) encryption.	
		63:MAXPHYSADDR	Reserved; must be zero.	
990H	2448	IA32_COPY_STATUS ⁵	Status of Most Recent Platform to Local or Local to Platform Copies (R/O)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		0	lwKEY_COPY_SUCCESSFUL: Status of most recent copy to or from lwKeyBackup	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		63:1	Reserved	
991H	2449	IA32_lwKEYBACKUP_STATUS ⁴	Information about lwKeyBackup Register (R/O)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Backup/Restore Valid Cleared when a write to <code>lWKeyBackup</code> is initiated, and then set when the latest write of <code>lWKeyBackup</code> has been written to storage that persists across S3/S4 sleep state. If S3/S4 is entered between when an <code>lWKeyBackup</code> write occurs and when this bit is set, then <code>lWKeyBackup</code> may not be recovered after S3/S4 exit. During S3/S4 sleep state exit (system wake up), this bit is cleared. It is set again when <code>lWKeyBackup</code> is restored from persistent storage and thus available to be copied to <code>lWKey</code> using <code>IA32_COPY_PLATFORM_TO_LOCAL</code> MSR. Another write to <code>lWKeyBackup</code> (via <code>IA32_COPY_LOCAL_TO_PLATFORM</code> MSR) may fail if a previous write has not yet set this bit.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		1	Reserved	
		2	Backup Key Storage Read/Write Error Updated prior to backup/restore valid being set. Set when an error is encountered while backing up or restoring a key to persistent storage.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		3	<code>lWKeyBackup</code> Consumed Set after the previous backup operation has been consumed by the platform. This does not indicate that the system is ready for a second <code>lWKeyBackup</code> write as the previous <code>lWKeyBackup</code> write may still need to set Backup/restore valid.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		63:4	Reserved	
C80H	3200	IA32_DEBUG_INTERFACE	Silicon Debug Feature Control (R/W)	If CPUID.01H:ECX.[11] = 1
		0	Enable (R/W) BIOS set 1 to enable Silicon debug features. Default is 0.	If CPUID.01H:ECX.[11] = 1
		29:1	Reserved	
		30	Lock (R/W): If 1, locks any further change to the MSR. The lock bit is set automatically on the first SMI assertion even if not explicitly set by BIOS. Default is 0.	If CPUID.01H:ECX.[11] = 1
		31	Debug Occurred (R/O): This “sticky bit” is set by hardware to indicate the status of bit 0. Default is 0.	If CPUID.01H:ECX.[11] = 1
		63:32	Reserved	
C81H	3201	IA32_L3_QOS_CFG	L3 QOS Configuration (R/W)	If (CPUID.(EAX=10H, ECX=1):ECX.[2] = 1)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Enable (R/W) Set 1 to enable L3 CAT masks and COS to operate in Code and Data Prioritization (CDP) mode.	
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
C82H	3202	IA32_L2_QOS_CFG	L2 QOS Configuration (R/W)	If (CPUID.(EAX=10H, ECX=2):ECX.[2] = 1)
		0	Enable (R/W) Set 1 to enable L2 CAT masks and COS to operate in Code and Data Prioritization (CDP) mode.	
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
C8DH	3213	IA32_QM_EVTSEL	Monitoring Event Select Register (R/W)	If (CPUID.(EAX=07H, ECX=0):EBX.[12] = 1)
		7:0	Event ID: ID of a supported monitoring event to report via IA32_QM_CTR.	
		31: 8	Reserved	
		N+31:32	Resource Monitoring ID: ID for monitoring hardware to report monitored data via IA32_QM_CTR.	N = Ceil (Log ₂ (CPUID.(EAX= 0FH, ECX=0H).EBX[31:0] +1))
		63:N+32	Reserved	
C8EH	3214	IA32_QM_CTR	Monitoring Counter Register (R/O)	If (CPUID.(EAX=07H, ECX=0):EBX.[12] = 1)
		61:0	Resource Monitored Data	
		62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
		63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
C8FH	3215	IA32_PQR_ASSOC	Resource Association Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[12] = 1) or (CPUID.(EAX=07H, ECX=0):EBX[15] = 1))
		N-1:0	Resource Monitoring ID (R/W): ID for monitoring hardware to track internal operation, e.g., memory access.	N = Ceil (Log ₂ (CPUID.(EAX= 0FH, ECX=0H).EBX[31:0] +1))
		31:N	Reserved	
		63:32	COS (R/W): The class of service (COS) to enforce (on writes); returns the current COS when read.	If (CPUID.(EAX=07H, ECX=0):EBX.[15] = 1)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C90H - D8FH	3216 - 3471	Reserved MSR Address Space for CAT Mask Registers	See Section 17.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology".	
C90H	3216	IA32_L3_MASK_0	L3 CAT Mask for COS0 (R/W)	If (CPUID.(EAX=10H, ECX=0H);EBX[1] != 0)
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
C90H+n	3216+n	IA32_L3_MASK_n	L3 CAT Mask for COSn (R/W)	n = CPUID.(EAX=10H, ECX=1H);EDX[15:0]
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D10H - D4FH	3344 - 3407	Reserved MSR Address Space for L2 CAT Mask Registers	See Section 17.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology".	
D10H	3344	IA32_L2_MASK_0	L2 CAT Mask for COS0 (R/W)	If (CPUID.(EAX=10H, ECX=0H);EBX[2] != 0)
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D10H+n	3344+n	IA32_L2_MASK_n	L2 CAT Mask for COSn (R/W)	n = CPUID.(EAX=10H, ECX=2H);EDX[15:0]
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D90H	3472	IA32_BNDCFGS	Supervisor State of MPX Configuration (R/W)	If (CPUID.(EAX=07H, ECX=0H);EBX[14] = 1)
		0	EN: Enable Intel MPX in supervisor mode.	
		1	BNDPRESERVE: Preserve the bounds registers for near branch instructions in the absence of the BND prefix.	
		11:2	Reserved, must be zero.	
		63:12	Base Address of Bound Directory.	
D91H	3473	IA32_COPY_LOCAL_TO_PLATFORM ⁴	Copy Local State to Platform State (W)	If ((CPUID.19H;EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H);ECX[23] = 1))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	lWKeyBackup Copy lWKey to lWKeyBackup	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		63:1	Reserved	
D92H	3474	IA32_COPY_PLATFORM_TO_LOCAL ⁴	Copy Platform State to Local State (W)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		0	lWKeyBackup Copy lWKeyBackup to lWKey	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		63:1	Reserved	
DA0H	3488	IA32_XSS	Extended Supervisor State Mask (R/W)	If (CPUID.(0DH, 1):EAX.[3] = 1)
		7:0	Reserved.	
		8	Trace Packet Configuration State (R/W)	
		10:9	Reserved.	
		11	CET_U State (R/W)	
		12	CET_S State (R/W)	
		13	HDC State (R/W)	
		63:14	Reserved.	
DB0H	3504	IA32_PKG_HDC_CTL	Package Level Enable/disable HDC (R/W)	If CPUID.06H:EAX.[13] = 1
		0	HDC_Pkg_Enable (R/W) Force HDC idling or wake up HDC-idled logical processors in the package. See Section 14.5.2, "Package level Enabling HDC".	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved	
DB1H	3505	IA32_PM_CTL1	Enable/disable HWP (R/W)	If CPUID.06H:EAX.[13] = 1
		0	HDC_Allow_Block (R/W) Allow/Block this logical processor for package level HDC control. See Section 14.5.3.	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved	
DB2H	3506	IA32_THREAD_STALL	Per-Logical_Processor HDC Idle Residency (R/O)	If CPUID.06H:EAX.[13] = 1
		63:0	Stall_Cycle_Cnt (R/W) Stalled cycles due to HDC forced idle on this logical processor. See Section 14.5.4.1.	If CPUID.06H:EAX.[13] = 1
1200H - 121FH	4608 - 4639	IA32_LBR_x_INFO	Last Branch Record Entry X Info Register (R/W) An attempt to read or write IA32_LBR_x_INFO such that x ≥ IA32_LBR_DEPTH.DEPTH will #GP.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		15:0	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 18.1.3.3.	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
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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		2	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 18.1.2.4.	Reset Value: 0
		15:4	Reserved	Reset Value: 0
		16	COND When set, records taken conditional branches. See Section 18.1.2.3.	
		17	NEAR_REL_JMP When set, records near relative JMPs. See Section 18.1.2.3.	
		18	NEAR_IND_JMP When set, records near indirect JMPs. See Section 18.1.2.3.	
		19	NEAR_REL_CALL When set, records near relative CALLs. See Section 18.1.2.3.	
		20	NEAR_IND_CALL When set, records near indirect CALLs. See Section 18.1.2.3.	
		21	NEAR_RET When set, records near RETs. See Section 18.1.2.3.	
		22	OTHER_BRANCH When set, records other branches. See Section 18.1.2.3.	
		63:23	Reserved	
		14CFH	5327	IA32_LBR_DEPTH
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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
1500H -151FH	5376 -5407	IA32_LBR_x_FROM_IP	Last Branch Record entry X source IP register (R/W). An attempt to read or write IA32_LBR_x_FROM_IP such that $x \geq$ IA32_LBR_DEPTH.DEPTH will #GP.	
		63:0	FROM_IP The source IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
1600H -161FH	5632 -5663	IA32_LBR_x_TO_IP	Last Branch Record Entry X Destination IP Register (R/W) An attempt to read or write IA32_LBR_x_TO_IP such that $x \geq$ IA32_LBR_DEPTH.DEPTH will #GP.	
		63:0	TO_IP The destination IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
17D0H	6096	IA32_HW_FEEDBACK_PTR	Hardware Feedback Interface Pointer	If CPUID.06H:EAX.[19] = 1
		0	Valid (R/W) When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.	
		11:1	Reserved	
		(MAXPHYADDR-1):12	ADDR (R/W) Physical address of the page frame of the first page of the hardware feedback interface structure.	
		63:MAXPHYADDR	Reserved	
17D1H	6097	IA32_HW_FEEDBACK_CONFIG	Hardware Feedback Interface Configuration	If CPUID.06H:EAX.[19] = 1
		0	Enable (R/W) When set to 1, enables the hardware feedback interface.	
		63:1	Reserved	
17D2H	6098	IA32_THREAD_FEEDBACK_CHAR	Thread Feedback Characteristics (R/O)	If CPUID.06H:EAX.[23] = 1
		7:0	Application Class ID, pointing into the Intel Thread Director structure.	
		62:8	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63	Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.	
17D4H	6100	IA32_HW_FEEDBACK_THREAD_CONFIG	Hardware Feedback Thread Configuration (R/W)	
		0	Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).	
		63:1	Reserved	
17DAH	6106	IA32_HRESET_ENABLE	History Reset Enable (R/W)	
		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	
4000_0000H - 4000_00FFH		Reserved MSR Address Space	All existing and future processors will not implement MSRs in this range.	
C000_0080H		IA32_EFER	Extended Feature Enables	If (CPUID.80000001H:EDX.[20]) CPUID.80000001H:EDX.[29])
	0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.		
	7:1	Reserved		
	8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.		
	9	Reserved		
	10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.		
	11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W)		
	63:12	Reserved		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C000_0081H		IA32_STAR	System Call Target Address (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0082H		IA32_LSTAR	IA-32e Mode System Call Target Address (R/W) Target RIP for the called procedure when SYSCALL is executed in 64-bit mode.	If CPUID.80000001:EDX.[29] = 1
C000_0083H		IA32_CSTAR	IA-32e Mode System Call Target Address (R/W) Not used, as the SYSCALL instruction is not recognized in compatibility mode.	If CPUID.80000001:EDX.[29] = 1
C000_0084H		IA32_FMASK	System Call Flag Mask (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0100H		IA32_FS_BASE	Map of BASE Address of FS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0101H		IA32_GS_BASE	Map of BASE Address of GS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0102H		IA32_KERNEL_GS_BASE	Swap Target of BASE Address of GS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0103H		IA32_TSC_AUX	Auxiliary TSC (R/W)	If CPUID.80000001H: EDX[27] = 1 or CPUID.(EAX=7,ECX=0):ECX[bit 22] = 1
		31:0	AUX: Auxiliary signature of TSC.	
		63:32	Reserved	

NOTES:

- Some older processors may have supported this MSR as model-specific and do not enumerate it with CPUID.
- 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.
- The *_ADDR MSRs may or may not be present; this depends on flag settings in IA32_MC_i_STATUS. See Section 15.3.2.3 and Section 15.3.2.4 for more information.
- MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].
- 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 with DisplayFamily_DisplayModel of 06_0FH, see Table 2-1.

MODEL-SPECIFIC REGISTERS (MSRS)

MSRs listed in Table 2-2 and Table 2-3 are also supported by processors based on the Enhanced Intel Core micro-architecture. Processors based on the Enhanced Intel Core microarchitecture have the CPUID signature DisplayFamily_DisplayModel of 06_17H.

The column “Shared/Unique” applies to multi-core processors based on Intel Core microarchitecture. “Unique” means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. “Shared” means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Unique	See Section 2.23, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Unique	See Section 2.23, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, “Monitor/Mwait Address Range Determination.” and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, “Time-Stamp Counter,” and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		7:0		Reserved
		12:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		49:13		Reserved
		52:50		See Table 2-2.
		63:53		Reserved
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, “Local APIC Status and Location” and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processors implement R/W.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processors implement R/W.
		4		Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processors implement R/W.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		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)
3AH	58	MSR_FEATURE_CONTROL	Unique	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		3	Unique	SMRR Enable (R/WL) When this bit is set and the lock bit is set, this makes the SMRR_PHYS_BASE and SMRR_PHYS_MASK registers read visible and writeable while in SMM.
40H	64	MSR_LASTBRANCH_0_FROM_IP	Unique	Last Branch Record 0 From IP (R/W) One of four pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.5.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	Last Branch Record 0 To IP (R/W) One of four pairs of last branch record registers on the last branch record stack. This To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (w) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (R/w) See Table 2-2.
A0H	160	MSR_SMRR_PHYSBASE	Unique	System Management Mode Base Address register (wO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		11:0		Reserved
		31:12		PhysBase: SMRR physical Base Address.
		63:32		Reserved
A1H	161	MSR_SMRR_PHYSMASK	Unique	System Management Mode Physical Address Mask register (wO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		10:0		Reserved
		11		Valid: Physical address base and range mask are valid.
		31:12		PhysMask: SMRR physical address range mask.
		63:32		Reserved
C1H	193	IA32_PMC0	Unique	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (R/O) This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture.
		2:0		<ul style="list-style-type: none"> ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333)
				<p>133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.</p> <p>166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.</p> <p>266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B.</p> <p>333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.</p>
		63:3		Reserved
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (R/O) This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture.
		2:0		<ul style="list-style-type: none"> ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600)
				<p>133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.</p> <p>166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.</p> <p>266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B.</p> <p>333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.</p>
		63:3		Reserved
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
		11	Unique	SMRR Capability Using MSR 0A0H and 0A1H (R)

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Current performance status. See Section 14.1.1, "Software Interface For Initiating Performance State Transitions".
		15:0		Current Performance State Value
		30:16		Reserved
		31		XE Operation (R/O). If set, XE operation is enabled. Default is cleared.
		39:32		Reserved
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		45		Reserved
		46		Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		63:47		Reserved
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2.
19DH	413	MSR_THERM2_CTL	Unique	Thermal Monitor 2 Control
		15:0		Reserved
		16		TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:16		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0		Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.
		8		Reserved
		9		Hardware Prefetcher Disable (R/W) When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Shared	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		13	Shared	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19	Shared	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: <ul style="list-style-type: none"> Enhanced Intel SpeedStep Technology Select Lock (this bit). Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved
		22	Shared	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Shared	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Unique	XD Bit Disable (R/W) See Table 2-2.
		36:35		Reserved
		37	Unique	DCU Prefetcher Disable (R/W) When set to 1, the DCU L1 data cache prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The DCU prefetcher is an L1 data cache prefetcher. When the DCU prefetcher detects multiple loads from the same line done within a time limit, the DCU prefetcher assumes the next line will be required. The next line is prefetched in to the L1 data cache from memory or L2.
		38	Shared	IDA Disable (R/W) When set to 1 on processors that support IDA, the Intel Dynamic Acceleration feature (IDA) is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of IDA is enabled. Note: The power-on default value is used by BIOS to detect hardware support of IDA. If the power-on default value is 1, IDA is available in the processor. If the power-on default value is 0, IDA is not available.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		39	Unique	IP Prefetcher Disable (R/W) When set to 1, the IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.
		63:40		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Unique	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Unique	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Unique	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Unique	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Unique	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Unique	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Unique	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Unique	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Unique	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Unique	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Unique	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Unique	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Unique	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Unique	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Unique	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Unique	See Table 2-2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
250H	592	IA32_MTRR_FIX64K_00000	Unique	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Unique	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Unique	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Unique	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Unique	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Unique	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Unique	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Unique	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Unique	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
345H	837	MSR_PERF_CAPABILITIES	Unique	R/O. This applies to processors that do not support architectural perfmon version 2.
		5:0		LBR Format. See Table 2-2.
		6		PEBS Record Format
		7		PEBSSaveArchRegs. See Table 2-2.
		63:8		Reserved
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Unique	See Section 19.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38FH	911	MSR_PERF_GLOBAL_CTRL	Unique	See Section 19.6.2.2, "Global Counter Control Facilities."

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, “Global Counter Control Facilities.”
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Unique	See Section 19.6.2.2, “Global Counter Control Facilities.”
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Unique	See Table 2-2. See Section 19.6.2.4, “Processor Event Based Sampling (PEBS).”
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MCO_CTL	Unique	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
401H	1025	IA32_MCO_STATUS	Unique	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
402H	1026	IA32_MCO_ADDR	Unique	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
405H	1029	IA32_MC1_STATUS	Unique	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
406H	1030	IA32_MC1_ADDR	Unique	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	Unique	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
409H	1033	IA32_MC2_STATUS	Unique	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
40AH	1034	IA32_MC2_ADDR	Unique	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC4_CTL	Unique	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
40DH	1037	IA32_MC4_STATUS	Unique	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
40EH	1038	IA32_MC4_ADDR	Unique	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
410H	1040	IA32_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC3_MISC	Unique	Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.
414H	1044	IA32_MC5_CTL	Unique	Machine Check Error Reporting Register: Controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
415H	1045	IA32_MC5_STATUS	Unique	Machine Check Error Reporting Register: Contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
416H	1046	IA32_MC5_ADDR	Unique	Machine Check Error Reporting Register: Contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCi_STATUS register is set.
417H	1047	IA32_MC5_MISC	Unique	Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.
419H	1045	IA32_MC6_STATUS	Unique	Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 23.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
484H	1156	IA32_VMX_ENTRY_CTL5	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Unique	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Unique	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
107CH	67532	MSR_EMON_L3_CTR_CTL0	Unique	GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CDH	67533	MSR_EMON_L3_CTR_CTL1	Unique	GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CEH	67534	MSR_EMON_L3_CTR_CTL2	Unique	GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
107CF H	67535	MSR_EMON_L3_CTR_CTL3	Unique	GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D0 H	67536	MSR_EMON_L3_CTR_CTL4	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D1 H	67537	MSR_EMON_L3_CTR_CTL5	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D2 H	67538	MSR_EMON_L3_CTR_CTL6	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D3 H	67539	MSR_EMON_L3_CTR_CTL7	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D8 H	67544	MSR_EMON_L3_GL_CTL	Unique	L3/FSB Common Control Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
C000_0080H		IA32_EFER	Unique	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

2.3 MSRS IN THE 45 NM AND 32 NM INTEL ATOM® PROCESSOR FAMILY

Table 2-4 lists model-specific registers (MSRs) for 45 nm and 32 nm Intel Atom processors, architectural MSR addresses are also included in Table 2-4. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1CH, 06_26H, 06_27H, 06_35H and 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 45 nm and 32 nm Intel Atom® Processor Family

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 2.23, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Shared	See Section 2.23, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, “Monitor/Mwait Address Range Determination.” and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, “Time-Stamp Counter,” and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		7:0		Reserved
		12:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		63:13		Reserved
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, “Local APIC Status and Location” and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		3		AERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		4		BERR# Enable for initiator bus requests (R/W) 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved
		6		Reserved
		7		BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		9		Execute BIST (R/O) 1 = Enabled; 0 = Disabled
		10		AERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved
		12		BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0.
		13		Reserved
		14		1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		APIC Cluster ID (R/O) Always 00B.
		19: 18		Reserved
		21: 20		Symmetric Arbitration ID (R/O) Always 00B.
		26:22		Integer Bus Frequency Ratio (R/O)
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in Intel 64Processor (R/W) See Table 2-2.
40H	64	MSR_LASTBRANCH_0_FROM_IP	Unique	Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction . See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.5.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Unique	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Unique	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Unique	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
47H	71	MSR_LASTBRANCH_7_FROM_IP	Unique	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Unique	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Unique	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Unique	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_LASTBRANCH_7_TO_IP	Unique	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Shared	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Unique	Performance counter register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (R/O) This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture.
		2:0		<ul style="list-style-type: none"> ▪ 111B: 083 MHz (FSB 333) ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667)
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
		63:3		Reserved

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Shared	Memory Type Range Register (R) See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (R/O) 1 = Indicates the L2 is hardware-enabled. 0 = Indicates the L2 is hardware-disabled.
		7:1		Reserved
		8		L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set, the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved
		23		L2 Not Present (R/O) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Performance Status
		15:0		Current Performance State Value
		39:16		Reserved
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		63:45		Reserved
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2.
19DH	413	MSR_THERM2_CTL	Shared	Thermal Monitor 2 Control
		15:0		Reserved
		16		TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:17		Reserved
1A0H	416	IA32_MISC_ENABLE	Unique	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0		Fast-Strings Enable See Table 2-2.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		2:1		Reserved
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.
		8		Reserved
		9		Reserved
		10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Shared	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		13	Shared	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is cleared (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19		Reserved

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	Enhanced Intel SpeedStep Technology Select Lock (R/W) When set, this bit causes the following bits to become read-only: <ul style="list-style-type: none"> ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit). ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved
		22	Unique	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Shared	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Unique	XD Bit Disable (R/W) See Table 2-2.
		63:35		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYSBASE0	Shared	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Shared	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Shared	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Shared	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Shared	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Shared	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Shared	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Shared	See Table 2-2.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
208H	520	IA32_MTRR_PHYSBASE4	Shared	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Shared	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Shared	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Shared	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Shared	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Shared	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Shared	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Shared	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Shared	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Shared	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Shared	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Shared	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Shared	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Shared	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Shared	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Unique	See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0 (R/W)

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
400H	1024	IA32_MCO_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
481H	1153	IA32_VMX_PINBASED_CTLX	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLX	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLX	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLX	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Unique	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Unique	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLX2	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
C000_0080H		IA32_EFER	Unique	Extended Feature Enables See Table 2-2.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
C000_0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

Table 2-5 lists model-specific registers (MSRs) that are specific to Intel Atom® processor with the CPUID signature with DisplayFamily_DisplayModel of 06_27H.

Table 2-5. MSRs Supported by Intel Atom® Processors with CPUID Signature 06_27H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3F8H	1016	MSR_PKG_C2_RESIDENCY	Package	Package C2 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0	Package	Package C2 Residency Counter (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.
3F9H	1017	MSR_PKG_C4_RESIDENCY	Package	Package C4 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0	Package	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	Package C6 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0	Package	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.

2.4 MSRS IN INTEL PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 2-6 lists model-specific registers (MSRs) common to Intel processors based on the Silvermont microarchitecture. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_37H, 06_4AH, 06_4DH, 06_5AH, and 06_5DH; see Table 2-1. The MSRs listed in Table 2-6 are also common to processors based on the Airmont microarchitecture and newer microarchitectures for next generation Intel Atom processors.

Table 2-7 lists MSRs common to processors based on the Silvermont and Airmont microarchitectures, but not newer microarchitectures.

Table 2-8, Table 2-9, and Table 2-10 lists MSRs that are model-specific across processors based on the Silvermont microarchitecture.

In the Silvermont microarchitecture, the scope column indicates the following: "Core" means each processor core has a separate MSR, or a bit field not shared with another processor core. "Module" means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Silvermont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID leaf 04H enumeration can be used to figure out which processors are in the same module. "Package" means all processor cores in the physical package share the same MSR or bit interface.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Core	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Core	See Section 17.17, "Time-Stamp Counter," and Table 2-2.
1BH	27	IA32_APIC_BASE	Core	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Module	Processor Hard Power-On Configuration (R/W) Writes ignored.
		63:0		Reserved
34H	52	MSR_SMI_COUNT	Core	SMI Counter (R/O)
		31:0		SMI Count (R/O) Running count of SMI events since last RESET.
		63:32		Reserved
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Core	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Core	Performance counter register See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C2H	194	IA32_PMC1	Core	Performance Counter Register See Table 2-2.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 100b - C4 is the max C-State to include 110b - C6 is the max C-State to include 111b - C7 is the max C-State to include
		63:19		Reserved
E7H	231	IA32_MPERF	Core	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Core	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Core	Memory Type Range Register (R) See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction sets availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note: AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
174H	372	IA32_SYSENTER_CS	Core	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Core	See Table 2-2.
179H	377	IA32_MCG_CAP	Core	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Core	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Core	See Table 2-2.
		7:0		Event Select
		15:8		UMask
		16		USR
		17		OS
		18		Edge
		19		PC
		20		INT
		21		Reserved
		22		EN
		23		INV
		31:24		CMASK
		63:32		Reserved
187H	391	IA32_PERFEVTSEL1	Core	See Table 2-2.
198H	408	IA32_PERF_STATUS	Module	See Table 2-2.
199H	409	IA32_PERF_CTL	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_MODULATION	Core	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R) The default thermal throttling or PROCHOT# activation temperature in degrees C. The effective temperature for thermal throttling or PROCHOT# activation is "Temperature Target" + "Target Offset".
		29:24		Target Offset (R/W) Specifies an offset in degrees C to adjust the throttling and PROCHOT# activation temperature from the default target specified in TEMPERATURE_TARGET (bits 23:16).
		63:30		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Module	Offcore Response Event Select Register (R/W)
1A7H	423	MSR_OFFCORE_RSP_1	Module	Offcore Response Event Select Register (R/W)
1B0H	432	IA32_ENERGY_PERF_BIAS	Core	See Table 2-2.
1D9H	473	IA32_DEBUGCTL	Core	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Core	Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Core	Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Core	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Core	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Core	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Core	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Core	Fixed-Function-Counter Control Register (R/W) See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
38FH	911	IA32_PERF_GLOBAL_CTRL	Core	See Table 2-2. See Section 19.6.2.2, “Global Counter Control Facilities.”
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Counts at the TSC Frequency.
400H	1024	IA32_MCO_CTL	Module	See Section 15.3.2.1, “IA32_MCI_CTL MSRs.”
401H	1025	IA32_MCO_STATUS	Module	See Section 15.3.2.2, “IA32_MCI_STATUS MSRS.”
402H	1026	IA32_MCO_ADDR	Module	See Section 15.3.2.3, “IA32_MCI_ADDR MSRs.” The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Module	See Section 15.3.2.1, “IA32_MCI_CTL MSRs.”
405H	1029	IA32_MC1_STATUS	Module	See Section 15.3.2.2, “IA32_MCI_STATUS MSRS.”
408H	1032	IA32_MC2_CTL	Module	See Section 15.3.2.1, “IA32_MCI_CTL MSRs.”
409H	1033	IA32_MC2_STATUS	Module	See Section 15.3.2.2, “IA32_MCI_STATUS MSRS.”
40AH	1034	IA32_MC2_ADDR	Module	See Section 15.3.2.3, “IA32_MCI_ADDR MSRs.” The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, “IA32_MCI_CTL MSRs.”
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, “IA32_MCI_STATUS MSRS.”
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, “IA32_MCI_ADDR MSRs.” The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, “IA32_MCI_CTL MSRs.”
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, “IA32_MCI_STATUS MSRS.”

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Core	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Core	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Core	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Core	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Core	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Core	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Core	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_FIXED1	Core	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Core	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Core	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Core	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLSS2	Core	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLSS	Core	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLSS	Core	Capability Reporting Register of Primary Processor-based VM-Execution Flex Controls (R/O) See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLSS	Core	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLSS	Core	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2
491H	1169	IA32_VMX_FMFUNC	Core	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2
4C1H	1217	IA32_A_PMC0	Core	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Core	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C1 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C1 states. Counts at the TSC frequency.
6E0H	1760	IA32_TSC_DEADLINE	Core	TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.
C000_0080H		IA32_EFER	Core	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Core	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Core	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Core	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Core	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Core	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Core	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Core	AUXILIARY TSC Signature (R/W) See Table 2-2

Table 2-7 lists model-specific registers (MSRs) that are common to Intel Atom® processors based on the Silvermont and Airmont microarchitectures but not newer microarchitectures.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	Model Specific Platform ID (R)
		7:0		Reserved
		13:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		49:13		Reserved
		52:50		See Table 2-2.
		63:33		Reserved

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Reserved
		2		Enable VMX outside SMX operation (R/WL)
40H	64	MSR_LASTBRANCH_0_FROM_IP	Core	Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.5 and record format in Section 17.4.8.1.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Core	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Core	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Core	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Core	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Core	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Core	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_LASTBRANCH_7_FROM_IP	Core	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Core	Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.
61H	97	MSR_LASTBRANCH_1_TO_IP	Core	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Core	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Core	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Core	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Core	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Core	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
67H	103	MSR_LASTBRANCH_7_TO_IP	Core	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information: Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.
		63:16		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-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
11EH	281	MSR_BBL_CR_CTL3	Module	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
		7:1		Reserved

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		8		L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved
		23		L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.
		63:24		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Core	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Module	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.
		6:4		Reserved
		7	Core	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Core	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Core	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Module	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Core	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
		22	Core	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Module	xTPR Message Disable (R/W) See Table 2-2.
33:24		Reserved		
34	Core	XD Bit Disable (R/W) See Table 2-2.		

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		37:35		Reserved
		38	Module	<p>Turbo Mode Disable (R/W)</p> <p>When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H: EAX[1]=0).</p> <p>When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled.</p> <p>Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.</p>
		63:39		Reserved
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
		63:9		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Core	<p>Last Branch Record Stack TOS (R/W)</p> <p>Contains an index (bits 0-2) that points to the MSR containing the most recent branch record.</p> <p>See MSR_LASTBRANCH_0_FROM_IP.</p>
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Core	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS for precise event on IA32_PMC0 (R/W)
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Counts at the TSC Frequency.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.

2.4.1 MSRs with Model-Specific Behavior in the Silvermont Microarchitecture

Table 2-8 lists model-specific registers (MSRs) that are specific to Intel Atom® processor E3000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_37H) and Intel Atom processors (CPUID signatures with DisplayFamily_DisplayModel of 06_4AH, 06_5AH, 06_5DH).

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with CPUID Signatures 06_37H, 06_4AH, 06_5AH, 06_5DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	Scaleable Bus Speed (R/O) This field indicates the intended scaleable bus clock speed for processors based on Silvermont microarchitecture.
		2:0		<ul style="list-style-type: none"> ▪ 100B: 080.0 MHz ▪ 000B: 083.3 MHz ▪ 001B: 100.0 MHz ▪ 010B: 133.3 MHz ▪ 011B: 116.7 MHz
		63:3		Reserved
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."
		3:0		Power Units Power related information (in milliWatts) is based on the multiplier, 2^PU; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units Energy related information (in microJoules) is based on the multiplier, 2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with CPUID Signatures 06_37H, 06_4AH, 06_5AH, 06_5DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15:13		Reserved
		19:16		Time Unit The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W)
		14:0		Package Power Limit #1 (R/W) See Section 14.10.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1 (R/W) See Section 14.10.3, "Package RAPL Domain."
		16		Package Clamping Limitation #1 (R/W) See Section 14.10.3, "Package RAPL Domain."
		23:17		Time Window for Power Limit #1 (R/W) In unit of second. If 0 is specified in bits [23:17], defaults to 1 second window.
		63:24		Reserved
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 2-8.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains." and MSR_RAPL_POWER_UNIT in Table 2-8.

Table 2-9 lists model-specific registers (MSRs) that are specific to Intel Atom® processor E3000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_37H).

Table 2-9. Specific MSRs Supported by Intel Atom® Processor E3000 Series with CPUID Signature 06_37H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
668H	1640	MSR_CC6_DEMOTION_POLICY_CONFIG	Package	Core C6 Demotion Policy Config MSR
		63:0		Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.
669H	1641	MSR_MC6_DEMOTION_POLICY_CONFIG	Package	Module C6 Demotion Policy Config MSR
		63:0		Controls module (i.e., two cores sharing the second-level cache) C6 demotion policy. Writing a value of 0 disables module level HW demotion policy.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-9. Specific MSRs Supported by Intel Atom® Processor E3000 Series (Contd.)with CPUID Signature 06_37H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.

Table 2-10 lists model-specific registers (MSRs) that are specific to Intel Atom® processor C2000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_4DH).

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with CPUID Signature 06_4DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		Reserved
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		63:3		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode (R/W)
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series (Contd.)with CPUID Signature

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		3:0		Power Units Power related information (in milliwatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliwatts increment.
		7:4		Reserved
		12:8		Energy Status Units. Energy related information (in microjoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microjoules increment.
		15:13		Reserved
		19:16		Time Unit The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
66EH	1646	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameter (R/O)
		14:0		Thermal Spec Power (R/O) The unsigned integer value is the equivalent of the thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
		63:15		Reserved

2.4.2 MSRs In Intel Atom® Processors Based on Airmont Microarchitecture

Intel Atom processor X7-Z8000 and X5-Z8000 series are based on the Airmont microarchitecture. These processors support MSRs listed in Table 2-6, Table 2-7, Table 2-8, and Table 2-11. These processors have a CPUID signature with DisplayFamily_DisplayModel including 06_4CH; see Table 2-1.

Table 2-11. MSRs in Intel Atom® Processors Based on the Airmont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	Scaleable Bus Speed (R/O) This field indicates the intended scaleable bus clock speed for processors based on Airmont microarchitecture.

Table 2-11. MSRs in Intel Atom® Processors Based on the Airmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		3:0		<ul style="list-style-type: none"> ▪ 0000B: 083.3 MHz ▪ 0001B: 100.0 MHz ▪ 0010B: 133.3 MHz ▪ 0011B: 116.7 MHz ▪ 0100B: 080.0 MHz ▪ 0101B: 093.3 MHz ▪ 0110B: 090.0 MHz ▪ 0111B: 088.9 MHz ▪ 1000B: 087.5 MHz
		63:5		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: No limit 001b: C1 010b: C2 110b: C6 111b: C7
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		63:16		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.

Table 2-11. MSRs in Intel Atom® Processors Based on the Airmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		18:16		C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include. 001b - Deep Power Down Technology is the max C-State. 010b - C7 is the max C-State to include.
		63:19		Reserved
638H	1592	MSR_PPO_POWER_LIMIT	Package	PPO RAPL Power Limit Control (R/W)
		14:0		PPO Power Limit #1 (R/W) See Section 14.10.4, "PPO/PP1 RAPL Domains" and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1 (R/W) See Section 14.10.4, "PPO/PP1 RAPL Domains."
		16		Reserved
		23:17		Time Window for Power Limit #1 (R/W) Specifies the time duration over which the average power must remain below PPO_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.
		63:24		Reserved

2.5 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Intel Atom processors based on the Goldmont microarchitecture support MSRs listed in Table 2-6 and Table 2-12. These processors have a CPUID signature with DisplayFamily_DisplayModel 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 the Goldmont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	Model Specific Platform ID (R)
		49:0		Reserved
		52:50		See Table 2-2.
		63:33		Reserved
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX inside SMX operation (R/WL)
		2		Enable VMX outside SMX operation (R/WL)
		14:8		SENTER local functions enables (R/WL)
		15		SENTER global functions enable (R/WL)
		18		SGX global functions enable (R/WL)
		63:19		Reserved
3BH	59	IA32_TSC_ADJUST	Core	Per-Core TSC ADJUST (R/W) See Table 2-2.
C3H	195	IA32_PMC2	Core	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Core	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		30	Package	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		3:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: No limit 0001b: C1 0010b: C3 0011b: C6 0100b: C7 0101b: C7S 0110b: C8 0111b: C9 1000b: C10
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:16		Reserved
17DH	381	MSR_SMM_MCA_CAP	Core	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved
188H	392	IA32_PERFEVTSEL2	Core	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Core	See Table 2-2.
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Core	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Package	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.
		6:4		Reserved
		7	Core	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Core	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Core	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Core	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
22	Core	Limit CPUID Maxval (R/W) See Table 2-2.		

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		23	Package	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Core	XD Bit Disable (R/W) See Table 2-2.
		37:35		Reserved
		38	Package	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.
		63:39		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1		Reserved
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		63:3		Reserved
1AAH	426	MSR_MISC_PWR_MGMT	Package	Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .
		0		EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.
		21:1		Reserved
		22		Thermal Interrupt Coordination Enable (R/W) If set, then thermal interrupt on one core is routed to all cores.
		63:23		Reserved

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode by Core Groups (R/W) Specifies Maximum Ratio Limit for each Core Group. Max ratio for groups with more cores must decrease monotonically. For groups with less than 4 cores, the max ratio must be 32 or less. For groups with 4-5 cores, the max ratio must be 22 or less. For groups with more than 5 cores, the max ratio must be 16 or less.
		7:0	Package	Maximum Ratio Limit for Active Cores in Group 0 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 0 threshold.
		15:8	Package	Maximum Ratio Limit for Active Cores in Group 1 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 1 threshold, and greater than the Group 0 threshold.
		23:16	Package	Maximum Ratio Limit for Active Cores in Group 2 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 2 threshold, and greater than the Group 1 threshold.
		31:24	Package	Maximum Ratio Limit for Active Cores in Group 3 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 3 threshold, and greater than the Group 2 threshold.
		39:32	Package	Maximum Ratio Limit for Active Cores in Group 4 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 4 threshold, and greater than the Group 3 threshold.
		47:40	Package	Maximum Ratio Limit for Active Cores in Group 5 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 5 threshold, and greater than the Group 4 threshold.
		55:48	Package	Maximum Ratio Limit for Active Cores in Group 6 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 6 threshold, and greater than the Group 5 threshold.
		63:56	Package	Maximum Ratio Limit for Active Cores in Group 7 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 7 threshold, and greater than the Group 6 threshold.
1AEH	430	MSR_TURBO_GROUP_CORECNT	Package	Group Size of Active Cores for Turbo Mode Operation (R/W) Writes of 0 threshold is ignored.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7:0	Package	Group 0 Core Count Threshold Maximum number of active cores to operate under the Group 0 Max Turbo Ratio limit.
		15:8	Package	Group 1 Core Count Threshold Maximum number of active cores to operate under the Group 1 Max Turbo Ratio limit. Must be greater than the Group 0 Core Count.
		23:16	Package	Group 2 Core Count Threshold Maximum number of active cores to operate under the Group 2 Max Turbo Ratio limit. Must be greater than the Group 1 Core Count.
		31:24	Package	Group 3 Core Count Threshold Maximum number of active cores to operate under the Group 3 Max Turbo Ratio limit. Must be greater than the Group 2 Core Count.
		39:32	Package	Group 4 Core Count Threshold Maximum number of active cores to operate under the Group 4 Max Turbo Ratio limit. Must be greater than the Group 3 Core Count.
		47:40	Package	Group 5 Core Count Threshold Maximum number of active cores to operate under the Group 5 Max Turbo Ratio limit. Must be greater than the Group 4 Core Count.
		55:48	Package	Group 6 Core Count Threshold Maximum number of active cores to operate under the Group 6 Max Turbo Ratio limit. Must be greater than the Group 5 Core Count.
		63:56	Package	Group 7 Core Count Threshold Maximum number of active cores to operate under the Group 7 Max Turbo Ratio limit. Must be greater than the Group 6 Core Count, and not less than the total number of processor cores in the package. E.g., specify 255.
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		8		FAR_BRANCH
		9		EN_CALL_STACK
		63:10		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Core	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See http://biosbits.org .
		0		Reserved
		1	Package	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved
210H	528	IA32_MTRR_PHYSBASE8	Core	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Core	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Core	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Core	See Table 2-2.
280H	640	IA32_MC0_CTL2	Module	See Table 2-2.
281H	641	IA32_MC1_CTL2	Module	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Module	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
300H	768	MSR_SGXOWNEREPOCH0	Package	Lower 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	769	MSR_SGXOWNEREPOCH1	Package	Upper 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0		Ovf_PMCO

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1		Ovf_PMC1
		2		Ovf_PMC2
		3		Ovf_PMC3
		31:4		Reserved
		32		Ovf_FixedCtr0
		33		Ovf_FixedCtr1
		34		Ovf_FixedCtr2
		54:35		Reserved
		55		Trace_ToPA_PMI
		57:56		Reserved
		58		LBR_Frz.
		59		CTR_Frz.
		60		ASCI
		61		Ovf_Uncore
		62		Ovf_BufDSSAVE
63		CondChgd		
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Core	See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0		Set 1 to clear Ovf_PMC0.
		1		Set 1 to clear Ovf_PMC1.
		2		Set 1 to clear Ovf_PMC2.
		3		Set 1 to clear Ovf_PMC3.
		31:4		Reserved
		32		Set 1 to clear Ovf_FixedCtr0.
		33		Set 1 to clear Ovf_FixedCtr1.
		34		Set 1 to clear Ovf_FixedCtr2.
		54:35		Reserved
		55		Set 1 to clear Trace_ToPA_PMI.
		57:56		Reserved
		58		Set 1 to clear LBR_Frz.
		59		Set 1 to clear CTR_Frz.
		60		Set 1 to clear ASCI.
61		Set 1 to clear Ovf_Uncore.		
62		Set 1 to clear Ovf_BufDSSAVE.		
63		Set 1 to clear CondChgd.		
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Core	See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		Set 1 to cause Ovf_PMC0 = 1.
		1		Set 1 to cause Ovf_PMC1 = 1.
		2		Set 1 to cause Ovf_PMC2 = 1.
		3		Set 1 to cause Ovf_PMC3 = 1.
		31:4		Reserved
		32		Set 1 to cause Ovf_FixedCtr0 = 1.
		33		Set 1 to cause Ovf_FixedCtr1 = 1.
		34		Set 1 to cause Ovf_FixedCtr2 = 1.
		54:35		Reserved
		55		Set 1 to cause Trace_ToPA_PMI = 1.
		57:56		Reserved
		58		Set 1 to cause LBR_Frz = 1.
		59		Set 1 to cause CTR_Frz = 1.
		60		Set 1 to cause ASCI = 1.
		61		Set 1 to cause Ovf_Uncore.
		62		Set 1 to cause Ovf_BufDSSAVE.
		63		Reserved
392H	914	IA32_PERF_GLOBAL_INUSE	Core	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
406H	1030	IA32_MC1_ADDR	Module	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
4C3H	1219	IA32_A_PMC2	Core	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Core	See Table 2-2.
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.
		0		Lock (SMM-RW0) When set to '1' locks this register from further changes.
		1		Reserved
		2		SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is OFFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
500H	1280	IA32_SGX_SVN_STATUS	Core	Status and SVN Threshold of SGX Support for ACM (R/O)
		0		Lock See Section 38.1.1.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1		Reserved
		23:16		SGX_SVN_SINIT See Section 38.1.1.3, "Interactions with Authenticated Code Modules (ACMs)".
		63:24		Reserved
560H	1376	IA32_RTIT_OUTPUT_BASE	Core	Trace Output Base Register (R/W) See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Core	Trace Output Mask Pointers Register (R/W) See Table 2-2.
570H	1392	IA32_RTIT_CTL	Core	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		6:4		Reserved, must be zero.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7		CR3 filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.
		9		MTCEn
		10		TSCEn
		11		DisRETC
		12		Reserved, must be zero.
		13		BranchEn
		17:14		MTCFreq
		18		Reserved, must be zero.
		22:19		CYCThresh
		23		Reserved, must be zero.
		27:24		PSBFreq
		31:28		Reserved, must be zero.
		35:32		ADDRO_CFG
		39:36		ADDR1_CFG
		63:40		Reserved, must be zero.
571H	1393	IA32_RTIT_STATUS	Core	Tracing Status Register (R/W)
		0		FilterEn Writes ignored.
		1		ContexEn Writes ignored.
		2		TriggerEn Writes ignored.
		3		Reserved
		4		Error (R/W)
		5		Stopped
		31:6		Reserved, must be zero.
		48:32		PacketByteCnt
		63:49		Reserved, must be zero.
572H	1394	IA32_RTIT_CR3_MATCH	Core	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match.
580H	1408	IA32_RTIT_ADDRO_A	Core	Region 0 Start Address (R/W)
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDRO_B	Core	Region 0 End Address (R/W)
		63:0		See Table 2-2.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
582H	1410	IA32_RTIT_ADDR1_A	Core	Region 1 Start Address (R/W)
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Core	Region 1 End Address (R/W)
		63:0		See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."
		3:0		Power Units Power related information (in Watts) is in unit of $1W/2^{PU}$; where PU is an unsigned integer represented by bits 3:0. Default value is 1000b, indicating power unit is in 3.9 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units Energy related information (in Joules) is in unit of $1Joule/2^{ESU}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 01110b, indicating energy unit is in 61 microJoules.
		15:13		Reserved
		19:16		Time Unit Time related information (in seconds) is in unit of $1S/2^{TU}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating power unit is in 0.977 millisecond.
		63:20		Reserved
60AH	1546	MSR_PKGC3_IRTL	Package	Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. 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

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60BH	1547	MSR_PKG_C2_IRTL1	Package	Package C6/C7S Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7S state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7S state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
60CH	1548	MSR_PKG_C2_IRTL2	Package	Package C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 14.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W)
		14:0		Thermal Spec Power (R/W) See Section 14.10.3, "Package RAPL Domain."
		15		Reserved
		30:16		Minimum Power (R/W) See Section 14.10.3, "Package RAPL Domain."
		31		Reserved
		46:32		Maximum Power (R/W) See Section 14.10.3, "Package RAPL Domain."
		47		Reserved
		54:48		Maximum Time Window (R/W) Specified by $2^Y * (1.0 + Z/4.0) * \text{Time_Unit}$, where "Y" is the unsigned integer value represented by bits 52:48, "Z" is an unsigned integer represented by bits 54:53. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
63:55		Reserved		
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
632H	1586		Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C10 Residency Counter (R/O) Value since last reset that the entire SOC is in an S0i3 state. Count at the same frequency as the TSC.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		7:0		MAX_NON_TURBO_RATIO (R/W/L) System BIOS can program this field.
		30:8		Reserved
		31		TURBO_ACTIVATION_RATIO_Lock (R/W/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		2		Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		3		Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		8:4		Reserved
		9		Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
11		Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.		

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		12		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		13		Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		14		Maximum Efficiency Frequency Status (R0) When set, frequency is reduced below the maximum efficiency frequency.
		15		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24:20		Reserved
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		Maximum Efficiency Frequency Log When set, indicates that the Maximum Efficiency Frequency Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:31		Reserved
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Core	Last Branch Record 0 From IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction . See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.6 and record format in Section 17.4.8.1.
		0:47		From Linear Address (R/W)
		62:48		Signed extension of bits 47:0.
		63		Mispred
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Core	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Core	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Core	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Core	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Core	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Core	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Core	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Core	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Core	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Core	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Core	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Core	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Core	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Core	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Core	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Core	Last Branch Record 16 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Core	Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Core	Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Core	Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Core	Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Core	Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Core	Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Core	Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Core	Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Core	Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Core	Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Core	Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Core	Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Core	Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Core	Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Core	Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Core	Last Branch Record 0 To IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the Destination instruction and elapsed cycles from last LBR update. See Section 17.6.
		0:47		Target Linear Address (R/W)
		63:48		Elapsed cycles from last update to the LBR.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Core	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Core	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Core	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Core	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Core	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Core	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Core	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Core	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Core	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Core	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Core	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Core	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Core	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Core	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Core	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DOH	1744	MSR_LASTBRANCH_16_TO_IP	Core	Last Branch Record 16 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Core	Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Core	Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Core	Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Core	Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Core	Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Core	Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Core	Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Core	Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Core	Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Core	Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Core	Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Core	Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Core	Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Core	Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Core	Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Core	x2APIC ID register (R/O)
803H	2051	IA32_X2APIC_VERSION	Core	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Core	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Core	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Core	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Core	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Core	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Core	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Core	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Core	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Core	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Core	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Core	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Core	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Core	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Core	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Core	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Core	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Core	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Core	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Core	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Core	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Core	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Core	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Core	x2APIC Interrupt Request register bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Core	x2APIC Interrupt Request register bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Core	x2APIC Interrupt Request register bits [127:96] (R/O)

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
824H	2084	IA32_X2APIC_IRR4	Core	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Core	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Core	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Core	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Core	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Core	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Core	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Core	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Core	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Core	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Core	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Core	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Core	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Core	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Core	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Core	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Core	x2APIC Self IPI register (W/O)
C8FH	3215	IA32_PQR_ASSOC	Core	Resource Association Register (R/W)
		31:0		Reserved
		33:32		COS (R/W)
		63: 34		Reserved
D10H	3344	IA32_L2_QOS_MASK_0	Module	L2 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.
		63:8		Reserved
D11H	3345	IA32_L2_QOS_MASK_1	Module	L2 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.
		63:8		Reserved
D12H	3346	IA32_L2_QOS_MASK_2	Module	L2 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:8		Reserved
D13H	3347	IA32_L2_QOS_MASK_3	Package	L2 Class Of Service Mask - COS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L2 ways for COS 3 enforcement.
		63:20		Reserved
D90H	3472	IA32_BNDCFGS	Core	See Table 2-2.
DA0H	3488	IA32_XSS	Core	See Table 2-2.

See Table 2-6, and Table 2-12 for MSR definitions applicable to processors with CPUID signature 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 with DisplayFamily_DisplayModel of 06_7AH; see Table 2-1. For an MSR listed in Table 2-13 that also appears in the model-specific tables of prior generations, Table 2-13 supersedes prior generation tables.

In the Goldmont Plus microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont Plus microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID leaf 04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX inside SMX operation (R/WL)
		2		Enable VMX outside SMX operation (R/WL)
		14:8		SENTER local functions enables (R/WL)
		15		SENTER global functions enable (R/WL)
		17		SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Valid if CPUID.(EAX=07H, ECX=0H): ECX[30] = 1.
		18		SGX global functions enable (R/WL)

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:19		Reserved
8CH	140	IA32_SGXLEPUBKEYHASH0	Core	See Table 2-2.
8DH	141	IA32_SGXLEPUBKEYHASH1	Core	See Table 2-2.
8EH	142	IA32_SGXLEPUBKEYHASH2	Core	See Table 2-2.
8FH	143	IA32_SGXLEPUBKEYHASH3	Core	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	(R/W) See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0.
		1		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC1.
		2		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC2.
		3		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC3.
		31:4		Reserved
		32		Enable PEBS trigger and recording for IA32_FIXED_CTR0.
		33		Enable PEBS trigger and recording for IA32_FIXED_CTR1.
		34		Enable PEBS trigger and recording for IA32_FIXED_CTR2.
		63:35		Reserved
		570H	1392	IA32_RTIT_CTL
0				TraceEn
1				CYCEn
2				OS
3				User
4				PwrEvtEn
5				FUPonPTW
6				FabricEn
7				CR3 filter
8				ToPA Writing 0 will #GP if also setting TraceEn.
9				MTCEn
10		TSCEn		

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Core	<p>Last Branch Record 0 From IP (R/W)</p> <p>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:</p> <ul style="list-style-type: none"> Last Branch Record Stack TOS at 1C9H. Section 17.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."
681H - 69FH	1665 - 1695	MSR_LASTBRANCH_i_FROM_IP	Core	<p>Last Branch Record <i>i</i> From IP (R/W)</p> <p>See description of MSR_LASTBRANCH_0_FROM_IP; <i>i</i> = 1-31.</p>
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Core	<p>Last Branch Record 0 To IP (R/W)</p> <p>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:</p> <ul style="list-style-type: none"> Section 17.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."
6C1H - 6DFH	1729 - 1759	MSR_LASTBRANCH_i_TO_IP	Core	<p>Last Branch Record <i>i</i> To IP (R/W)</p> <p>See description of MSR_LASTBRANCH_0_TO_IP; <i>i</i> = 1-31.</p>
DC0H	3520	MSR_LASTBRANCH_INFO_0	Core	<p>Last Branch Record 0 Additional Information (R/W)</p> <p>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:</p> <ul style="list-style-type: none"> Last Branch Record Stack TOS at 1C9H. Section 17.9.1, "LBR Stack."
DC1H	3521	MSR_LASTBRANCH_INFO_1	Core	<p>Last Branch Record 1 Additional Information (R/W)</p> <p>See description of MSR_LASTBRANCH_INFO_0.</p>

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DC2H	3522	MSR_LASTBRANCH_INFO_2	Core	Last Branch Record 2 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC3H	3523	MSR_LASTBRANCH_INFO_3	Core	Last Branch Record 3 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC4H	3524	MSR_LASTBRANCH_INFO_4	Core	Last Branch Record 4 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC5H	3525	MSR_LASTBRANCH_INFO_5	Core	Last Branch Record 5 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC6H	3526	MSR_LASTBRANCH_INFO_6	Core	Last Branch Record 6 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC7H	3527	MSR_LASTBRANCH_INFO_7	Core	Last Branch Record 7 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC8H	3528	MSR_LASTBRANCH_INFO_8	Core	Last Branch Record 8 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC9H	3529	MSR_LASTBRANCH_INFO_9	Core	Last Branch Record 9 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCAH	3530	MSR_LASTBRANCH_INFO_10	Core	Last Branch Record 10 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCBH	3531	MSR_LASTBRANCH_INFO_11	Core	Last Branch Record 11 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCCH	3532	MSR_LASTBRANCH_INFO_12	Core	Last Branch Record 12 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCDH	3533	MSR_LASTBRANCH_INFO_13	Core	Last Branch Record 13 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCEH	3534	MSR_LASTBRANCH_INFO_14	Core	Last Branch Record 14 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCFH	3535	MSR_LASTBRANCH_INFO_15	Core	Last Branch Record 15 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD0H	3536	MSR_LASTBRANCH_INFO_16	Core	Last Branch Record 16 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD1H	3537	MSR_LASTBRANCH_INFO_17	Core	Last Branch Record 17 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD2H	3538	MSR_LASTBRANCH_INFO_18	Core	Last Branch Record 18 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD3H	3539	MSR_LASTBRANCH_INFO_19	Core	Last Branch Record 19 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD4H	3520	MSR_LASTBRANCH_INFO_20	Core	Last Branch Record 20 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD5H	3521	MSR_LASTBRANCH_INFO_21	Core	Last Branch Record 21 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD6H	3522	MSR_LASTBRANCH_INFO_22	Core	Last Branch Record 22 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD7H	3523	MSR_LASTBRANCH_INFO_23	Core	Last Branch Record 23 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD8H	3524	MSR_LASTBRANCH_INFO_24	Core	Last Branch Record 24 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD9H	3525	MSR_LASTBRANCH_INFO_25	Core	Last Branch Record 25 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDAH	3526	MSR_LASTBRANCH_INFO_26	Core	Last Branch Record 26 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDBH	3527	MSR_LASTBRANCH_INFO_27	Core	Last Branch Record 27 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDCH	3528	MSR_LASTBRANCH_INFO_28	Core	Last Branch Record 28 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDDH	3529	MSR_LASTBRANCH_INFO_29	Core	Last Branch Record 29 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDEH	3530	MSR_LASTBRANCH_INFO_30	Core	Last Branch Record 30 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDFH	3531	MSR_LASTBRANCH_INFO_31	Core	Last Branch Record 31 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.

See Table 2-6, Table 2-12 and Table 2-13 for MSR definitions applicable to processors with CPUID signature 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 with DisplayFamily_DisplayModel of 06_86H, 06_96H, or 06_9CH; see Table 2-1. For an MSR listed in Table 2-14 that also appears in the model-specific tables of prior generations, Table 2-14 supersedes prior generation tables.

In the Tremont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Tremont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID leaf 04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-14. MSRs in Intel Atom® Processors Based on the Tremont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		28:0		Reserved.
		29		Enable #AC(0) exception for split locked accesses: Cause #AC(0) exception for split locked access at all CPL irrespective of CRO.AM or EFLAGS.AC. If bits 29 and 31 are both set, bit 29 takes precedence.
		30		Reserved.
		31		Reserved.
CFH	207	IA32_CORE_CAPABILITIES	Core	IA32 Core Capabilities Register If CPUID.(EAX=07H, ECX=0):EDX[30] = 1.
		4:0		Reserved.
		5		Bit 29 of MSR_MEMORY_CTRL (address 33H) supported.
		63:6		Reserved.
2A0H	672	MSR_PRMRR_BASE_0	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MEMTYPE: PRMRR BASE Memory Type.
		3		CONFIGURED: PRMRR BASE Configured.
		11:4		Reserved.
		51:12		BASE: PRMRR Base Address.
		63:52		Reserved.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	(R/W) See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)".
		<i>n</i> :0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMCx. The maximum value <i>n</i> can be determined from CPUID.0AH:EAX[15:8].
		31: <i>n</i> +1		Reserved.
		32+ <i>m</i> :32		Enable PEBS trigger and recording for IA32_FIXED_CTRx. The maximum value <i>m</i> can be determined from CPUID.0AH:EDX[4:0].
		59:33+ <i>m</i>		Reserved.
		60		Pend a PerfMon Interrupt (PMI) after each PEBS event.
		62:61		Specifies PEBS output destination. Encodings: 00B: DS Save Area 01B: Intel PT trace output. Supported if IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] and CPUID.07H.0.EBX[25] are set. 10B: Reserved 11B: Reserved

Table 2-14. MSRs in Intel Atom® Processors Based on the Tremont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63		Reserved.
1309H -	4873 -	MSR_RELOAD_FIXED_CTRx		Reload value for IA32_FIXED_CTRx (R/W)
130BH	4875	47:0		Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.
		63:48		Reserved.
14C1H -	5313 -	MSR_RELOAD_PMCx	Core	Reload value for IA32_PMCx (R/W)
14C4H	5316	47:0		Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.
		63:48		Reserved.

See Table 2-6, Table 2-12, Table 2-13 and Table 2-14 for MSR definitions applicable to processors with CPUID signature 06_86H.

2.8 MSRS IN THE 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 with DisplayFamily_DisplayModel of 06_1AH, 06_1EH, 06_1FH, 06_2EH, see Table 2-1. Additional MSRs specific to 06_1AH, 06_1EH, 06_1FH are listed in Table 2-16. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at <http://biosbits.org>.

The column "Scope" represents the package/core/thread scope of individual bit field of an MSR. "Thread" means this bit field must be programmed on each logical processor independently. "Core" means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. "Package" means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination" and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Package	Model Specific Platform ID (R)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		49:0		Reserved
		52:50		See Table 2-2.
		63:53		Reserved
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Running count of SMI events since last RESET.
		63:32		Reserved
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64Processor (R/W) See Table 2-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (w) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (R/w) See Table 2-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 2-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDC-TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDC and TDP Limits for Turbo mode are programmable. When set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 133.33MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		23:16		Reserved

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		24		Interrupt filtering enable (R/W) When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.
		25		C3 state auto demotion enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		C1 state auto demotion enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.
		63:19		Reserved
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Thread	See Table 2-2.
		7:0		Event Select
		15:8		UMask
		16		USR
		17		OS
		18		Edge
		19		PC
		20		INT
		21		AnyThread
		22		EN
		23		INV
		31:24		CMASK
		63:32		Reserved
		187H	391	IA32_PERFEVTSEL1
188H	392	IA32_PERFEVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Core	See Table 2-2.
		15:0		Current Performance State Value.
		63:16		Reserved
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved
		3:1		On demand Clock Modulation Duty Cycle (R/W)
		4		On demand Clock Modulation Enable (R/W)
		63:5		Reserved
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Thread	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.
		6:4		Reserved
		7	Thread	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Thread	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Thread	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 2-2.
		21:19		Reserved
		22	Thread	Limit CPUID Maxval (R/W) See Table 2-2.
23	Thread	xTPR Message Disable (R/W) See Table 2-2.		
33:24		Reserved		

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		34	Thread	XD Bit Disable (R/W) See Table 2-2.
		37:35		Reserved
		38	Package	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Thread	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.
		63:24		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1	Core	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.
		63:4		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0	Package	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.
		1	Thread	Energy/Performance Bias Enable (R/W) This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.(EAX=06h):ECX[3].
		63:2		Reserved
1ACH	428	MSR_TURBO_POWER_CURRENT_LIMIT		See http://biosbits.org .
		14:0	Package	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.
		15	Package	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.
		30:16	Package	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.
		31	Package	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.
		63:32		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
		63:9		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register See http://biosbits.org .
		0		Reserved
		1	Package	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MC0_CTL2	Package	See Table 2-2.
281H	641	IA32_MC1_CTL2	Package	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Core	See Table 2-2.
285H	645	IA32_MC5_CTL2	Core	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format See Table 2-2.
		6		PEBS Record Format
		7		PEBSSaveArchRegs See Table 2-2.
		11:8		PEBS_REC_FORMAT See Table 2-2.
		12		SMM_FREEZE See Table 2-2.
		63:13		Reserved
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Thread	Provides single-bit status used by software to query the overflow condition of each performance counter. (R/O)
		61		UNC_Ovf Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities." Allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Thread	(R/W)
		61		CLR_UNC_Ovf Set 1 to clear UNC_Ovf.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 19.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0 (R/W)
		1		Enable PEBS on IA32_PMC1 (R/W)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		2		Enable PEBS on IA32_PMC2 (R/W)
		3		Enable PEBS on IA32_PMC3 (R/W)
		31:4		Reserved
		32		Enable Load Latency on IA32_PMC0 (R/W)
		33		Enable Load Latency on IA32_PMC1 (R/W)
		34		Enable Load Latency on IA32_PMC2 (R/W)
		35		Enable Load Latency on IA32_PMC3 (R/W)
		63:36		Reserved
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 19.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
400H	1024	IA32_MCO_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
402H	1026	IA32_MCO_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MCO_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
406H	1030	IA32_MC1_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	IA32_MC1_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40BH	1035	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC4_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
414H	1044	IA32_MC5_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	IA32_MC5_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs" and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	IA32_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	IA32_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs" and Chapter 16.
41EH	1054	IA32_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	IA32_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs" and Chapter 16.
422H	1058	IA32_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
483H	1155	IA32_VMX_EXIT_CTL5	Thread	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL5	Thread	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CRO_FIXED0	Thread	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
487H	1159	IA32_VMX_CRO_FIXED1	Thread	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CRO."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTL52	Thread	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.9.1 and record format in Section 17.4.8.1.
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID Register (R/O)
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version Register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority Register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority Register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI Register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination Register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector Register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service Register Bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service Register Bits [63:32] (R/O)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service Register Bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service Register Bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service Register Bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service Register Bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service Register Bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service Register Bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode Register Bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode Register Bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode Register Bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode Register Bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode Register Bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode Register Bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode Register Bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode Register Bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request Register Bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request Register Bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request Register Bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request Register Bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request Register Bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request Register Bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request Register Bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request Register Bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status Register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command Register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt Register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt Register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor Register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 Register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 Register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error Register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count Register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count Register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration Register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI Register (W/O)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 17.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

2.8.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Intel Xeon Processor 5500 and 3400 series support additional model-specific registers listed in Table 2-16. These MSRs also apply to Intel Core i7 and i5 processor family CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH and 06_1FH, see Table 2-1.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 5500 and 3400 Series

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Actual maximum turbo frequency is multiplied by 133.33MHz. (Not available in model 06_2EH.)
		7:0		Maximum Turbo Ratio Limit 1C (R/O) Maximum Turbo mode ratio limit with 1 core active.
		15:8		Maximum Turbo Ratio Limit 2C (R/O) Maximum Turbo mode ratio limit with 2 cores active.
		23:16		Maximum Turbo Ratio Limit 3C (R/O) Maximum Turbo mode ratio limit with 3 cores active.
		31:24		Maximum Turbo Ratio Limit 4C (R/O) Maximum Turbo mode ratio limit with 4 cores active.
		63:32		Reserved
301H	769	MSR_GQ_SNOOP_MESF	Package	

Table 2-16. Additional MSRs in Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		From M to S (R/W)
		1		From E to S (R/W)
		2		From S to S (R/W)
		3		From F to S (R/W)
		4		From M to I (R/W)
		5		From E to I (R/W)
		6		From S to I (R/W)
		7		From F to I (R/W)
		63:8		Reserved
391H	913	MSR_UNCORE_PERF_GLOBAL_CTRL	Package	See Section 19.3.1.2.1, "Uncore Performance Monitoring Management Facility."
392H	914	MSR_UNCORE_PERF_GLOBAL_STATUS	Package	See Section 19.3.1.2.1, "Uncore Performance Monitoring Management Facility."
393H	915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	Package	See Section 19.3.1.2.1, "Uncore Performance Monitoring Management Facility."
394H	916	MSR_UNCORE_FIXED_CTRL0	Package	See Section 19.3.1.2.1, "Uncore Performance Monitoring Management Facility."
395H	917	MSR_UNCORE_FIXED_CTRL_CTRL	Package	See Section 19.3.1.2.1, "Uncore Performance Monitoring Management Facility."
396H	918	MSR_UNCORE_ADDR_OPCODE_MATCH	Package	See Section 19.3.1.2.3, "Uncore Address/Opcode Match MSR."
3B0H	960	MSR_UNCORE_PMC0	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B1H	961	MSR_UNCORE_PMC1	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B2H	962	MSR_UNCORE_PMC2	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B3H	963	MSR_UNCORE_PMC3	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B4H	964	MSR_UNCORE_PMC4	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B6H	966	MSR_UNCORE_PMC6	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B7H	967	MSR_UNCORE_PMC7	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C0H	944	MSR_UNCORE_PERFEVTSELO	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C1H	945	MSR_UNCORE_PERFEVTSSEL1	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."

Table 2-16. Additional MSRs in Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3C2H	946	MSR_UNCORE_PERFEVTSEL2	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C3H	947	MSR_UNCORE_PERFEVTSEL3	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C4H	948	MSR_UNCORE_PERFEVTSEL4	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C5H	949	MSR_UNCORE_PERFEVTSEL5	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C6H	950	MSR_UNCORE_PERFEVTSEL6	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C7H	951	MSR_UNCORE_PERFEVTSEL7	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."

2.8.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

Intel Xeon Processor 7500 series support 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 with DisplayFamily_DisplayModel of 06_2EH.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved Attempt to read/write will cause #UD.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
394H	816	MSR_W_PMON_FIXED_CTR	Package	Uncore W-box perfmon fixed counter.
395H	817	MSR_W_PMON_FIXED_CTR_CTL	Package	Uncore U-box perfmon fixed counter control MSR.
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
425H	1061	IA32_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
426H	1062	IA32_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
427H	1063	IA32_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
429H	1065	IA32_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42AH	1066	IA32_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
42BH	1067	IA32_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
42DH	1069	IA32_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42EH	1070	IA32_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
42FH	1071	IA32_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
431H	1073	IA32_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
432H	1074	IA32_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
433H	1075	IA32_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
435H	1077	IA32_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
436H	1078	IA32_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
437H	1079	IA32_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
439H	1081	IA32_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
43AH	1082	IA32_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
43BH	1083	IA32_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
43DH	1085	IA32_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
43EH	1086	IA32_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
43FH	1087	IA32_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
441H	1089	IA32_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
442H	1090	IA32_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
443H	1091	IA32_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
446H	1094	IA32_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44AH	1098	IA32_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44EH	1102	IA32_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	IA32_MC20_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
452H	1106	IA32_MC20_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
453H	1107	IA32_MC20_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
455H	1109	IA32_MC21_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
456H	1110	IA32_MC21_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
457H	1111	IA32_MC21_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
C00H	3072	MSR_U_PMON_GLOBAL_CTRL	Package	Uncore U-box perfmon global control MSR.
C01H	3073	MSR_U_PMON_GLOBAL_STATUS	Package	Uncore U-box perfmon global status MSR.
C02H	3074	MSR_U_PMON_GLOBAL_OVF_CTRL	Package	Uncore U-box perfmon global overflow control MSR.
C10H	3088	MSR_U_PMON_EVNT_SEL	Package	Uncore U-box perfmon event select MSR.
C11H	3089	MSR_U_PMON_CTR	Package	Uncore U-box perfmon counter MSR.
C20H	3104	MSR_B0_PMON_BOX_CTRL	Package	Uncore B-box 0 perfmon local box control MSR.
C21H	3105	MSR_B0_PMON_BOX_STATUS	Package	Uncore B-box 0 perfmon local box status MSR.
C22H	3106	MSR_B0_PMON_BOX_OVF_CTRL	Package	Uncore B-box 0 perfmon local box overflow control MSR.
C30H	3120	MSR_B0_PMON_EVNT_SELO	Package	Uncore B-box 0 perfmon event select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C31H	3121	MSR_B0_PMON_CTRL0	Package	Uncore B-box 0 perfmon counter MSR.
C32H	3122	MSR_B0_PMON_EVNT_SEL1	Package	Uncore B-box 0 perfmon event select MSR.
C33H	3123	MSR_B0_PMON_CTRL1	Package	Uncore B-box 0 perfmon counter MSR.
C34H	3124	MSR_B0_PMON_EVNT_SEL2	Package	Uncore B-box 0 perfmon event select MSR.
C35H	3125	MSR_B0_PMON_CTRL2	Package	Uncore B-box 0 perfmon counter MSR.
C36H	3126	MSR_B0_PMON_EVNT_SEL3	Package	Uncore B-box 0 perfmon event select MSR.
C37H	3127	MSR_B0_PMON_CTRL3	Package	Uncore B-box 0 perfmon counter MSR.
C40H	3136	MSR_S0_PMON_BOX_CTRL	Package	Uncore S-box 0 perfmon local box control MSR.
C41H	3137	MSR_S0_PMON_BOX_STATUS	Package	Uncore S-box 0 perfmon local box status MSR.
C42H	3138	MSR_S0_PMON_BOX_OVF_CTRL	Package	Uncore S-box 0 perfmon local box overflow control MSR.
C50H	3152	MSR_S0_PMON_EVNT_SELO	Package	Uncore S-box 0 perfmon event select MSR.
C51H	3153	MSR_S0_PMON_CTRL0	Package	Uncore S-box 0 perfmon counter MSR.
C52H	3154	MSR_S0_PMON_EVNT_SEL1	Package	Uncore S-box 0 perfmon event select MSR.
C53H	3155	MSR_S0_PMON_CTRL1	Package	Uncore S-box 0 perfmon counter MSR.
C54H	3156	MSR_S0_PMON_EVNT_SEL2	Package	Uncore S-box 0 perfmon event select MSR.
C55H	3157	MSR_S0_PMON_CTRL2	Package	Uncore S-box 0 perfmon counter MSR.
C56H	3158	MSR_S0_PMON_EVNT_SEL3	Package	Uncore S-box 0 perfmon event select MSR.
C57H	3159	MSR_S0_PMON_CTRL3	Package	Uncore S-box 0 perfmon counter MSR.
C60H	3168	MSR_B1_PMON_BOX_CTRL	Package	Uncore B-box 1 perfmon local box control MSR.
C61H	3169	MSR_B1_PMON_BOX_STATUS	Package	Uncore B-box 1 perfmon local box status MSR.
C62H	3170	MSR_B1_PMON_BOX_OVF_CTRL	Package	Uncore B-box 1 perfmon local box overflow control MSR.
C70H	3184	MSR_B1_PMON_EVNT_SELO	Package	Uncore B-box 1 perfmon event select MSR.
C71H	3185	MSR_B1_PMON_CTRL0	Package	Uncore B-box 1 perfmon counter MSR.
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR.
C73H	3187	MSR_B1_PMON_CTRL1	Package	Uncore B-box 1 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C74H	3188	MSR_B1_PMON_EVNT_SEL2	Package	Uncore B-box 1 perfmon event select MSR.
C75H	3189	MSR_B1_PMON_CTR2	Package	Uncore B-box 1 perfmon counter MSR.
C76H	3190	MSR_B1_PMON_EVNT_SEL3	Package	Uncore B-box 1 vperfmon event select MSR.
C77H	3191	MSR_B1_PMON_CTR3	Package	Uncore B-box 1 perfmon counter MSR.
C80H	3120	MSR_W_PMON_BOX_CTRL	Package	Uncore W-box perfmon local box control MSR.
C81H	3121	MSR_W_PMON_BOX_STATUS	Package	Uncore W-box perfmon local box status MSR.
C82H	3122	MSR_W_PMON_BOX_OVF_CTRL	Package	Uncore W-box perfmon local box overflow control MSR.
C90H	3136	MSR_W_PMON_EVNT_SELO	Package	Uncore W-box perfmon event select MSR.
C91H	3137	MSR_W_PMON_CTR0	Package	Uncore W-box perfmon counter MSR.
C92H	3138	MSR_W_PMON_EVNT_SEL1	Package	Uncore W-box perfmon event select MSR.
C93H	3139	MSR_W_PMON_CTR1	Package	Uncore W-box perfmon counter MSR.
C94H	3140	MSR_W_PMON_EVNT_SEL2	Package	Uncore W-box perfmon event select MSR.
C95H	3141	MSR_W_PMON_CTR2	Package	Uncore W-box perfmon counter MSR.
C96H	3142	MSR_W_PMON_EVNT_SEL3	Package	Uncore W-box perfmon event select MSR.
C97H	3143	MSR_W_PMON_CTR3	Package	Uncore W-box perfmon counter MSR.
CA0H	3232	MSR_M0_PMON_BOX_CTRL	Package	Uncore M-box 0 perfmon local box control MSR.
CA1H	3233	MSR_M0_PMON_BOX_STATUS	Package	Uncore M-box 0 perfmon local box status MSR.
CA2H	3234	MSR_M0_PMON_BOX_OVF_CTRL	Package	Uncore M-box 0 perfmon local box overflow control MSR.
CA4H	3236	MSR_M0_PMON_TIMESTAMP	Package	Uncore M-box 0 perfmon time stamp unit select MSR.
CA5H	3237	MSR_M0_PMON_DSP	Package	Uncore M-box 0 perfmon DSP unit select MSR.
CA6H	3238	MSR_M0_PMON_ISS	Package	Uncore M-box 0 perfmon ISS unit select MSR.
CA7H	3239	MSR_M0_PMON_MAP	Package	Uncore M-box 0 perfmon MAP unit select MSR.
CA8H	3240	MSR_M0_PMON_MSC_THR	Package	Uncore M-box 0 perfmon MIC THR select MSR.
CA9H	3241	MSR_M0_PMON_PGT	Package	Uncore M-box 0 perfmon PGT unit select MSR.
CAAH	3242	MSR_M0_PMON_PLD	Package	Uncore M-box 0 perfmon PLD unit select MSR.
CABH	3243	MSR_M0_PMON_ZDP	Package	Uncore M-box 0 perfmon ZDP unit select MSR.
CB0H	3248	MSR_M0_PMON_EVNT_SELO	Package	Uncore M-box 0 perfmon event select MSR.
CB1H	3249	MSR_M0_PMON_CTR0	Package	Uncore M-box 0 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CB2H	3250	MSR_M0_PMON_EVNT_SEL1	Package	Uncore M-box 0 perfmon event select MSR.
CB3H	3251	MSR_M0_PMON_CTR1	Package	Uncore M-box 0 perfmon counter MSR.
CB4H	3252	MSR_M0_PMON_EVNT_SEL2	Package	Uncore M-box 0 perfmon event select MSR.
CB5H	3253	MSR_M0_PMON_CTR2	Package	Uncore M-box 0 perfmon counter MSR.
CB6H	3254	MSR_M0_PMON_EVNT_SEL3	Package	Uncore M-box 0 perfmon event select MSR.
CB7H	3255	MSR_M0_PMON_CTR3	Package	Uncore M-box 0 perfmon counter MSR.
CB8H	3256	MSR_M0_PMON_EVNT_SEL4	Package	Uncore M-box 0 perfmon event select MSR.
CB9H	3257	MSR_M0_PMON_CTR4	Package	Uncore M-box 0 perfmon counter MSR.
CBAH	3258	MSR_M0_PMON_EVNT_SEL5	Package	Uncore M-box 0 perfmon event select MSR.
CBBH	3259	MSR_M0_PMON_CTR5	Package	Uncore M-box 0 perfmon counter MSR.
CC0H	3264	MSR_S1_PMON_BOX_CTRL	Package	Uncore S-box 1 perfmon local box control MSR.
CC1H	3265	MSR_S1_PMON_BOX_STATUS	Package	Uncore S-box 1 perfmon local box status MSR.
CC2H	3266	MSR_S1_PMON_BOX_OVF_CTRL	Package	Uncore S-box 1 perfmon local box overflow control MSR.
CDOH	3280	MSR_S1_PMON_EVNT_SELO	Package	Uncore S-box 1 perfmon event select MSR.
CD1H	3281	MSR_S1_PMON_CTR0	Package	Uncore S-box 1 perfmon counter MSR.
CD2H	3282	MSR_S1_PMON_EVNT_SEL1	Package	Uncore S-box 1 perfmon event select MSR.
CD3H	3283	MSR_S1_PMON_CTR1	Package	Uncore S-box 1 perfmon counter MSR.
CD4H	3284	MSR_S1_PMON_EVNT_SEL2	Package	Uncore S-box 1 perfmon event select MSR.
CD5H	3285	MSR_S1_PMON_CTR2	Package	Uncore S-box 1 perfmon counter MSR.
CD6H	3286	MSR_S1_PMON_EVNT_SEL3	Package	Uncore S-box 1 perfmon event select MSR.
CD7H	3287	MSR_S1_PMON_CTR3	Package	Uncore S-box 1 perfmon counter MSR.
CE0H	3296	MSR_M1_PMON_BOX_CTRL	Package	Uncore M-box 1 perfmon local box control MSR.
CE1H	3297	MSR_M1_PMON_BOX_STATUS	Package	Uncore M-box 1 perfmon local box status MSR.
CE2H	3298	MSR_M1_PMON_BOX_OVF_CTRL	Package	Uncore M-box 1 perfmon local box overflow control MSR.
CE4H	3300	MSR_M1_PMON_TIMESTAMP	Package	Uncore M-box 1 perfmon time stamp unit select MSR.
CE5H	3301	MSR_M1_PMON_DSP	Package	Uncore M-box 1 perfmon DSP unit select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CE6H	3302	MSR_M1_PMON_ISS	Package	Uncore M-box 1 perfmon ISS unit select MSR.
CE7H	3303	MSR_M1_PMON_MAP	Package	Uncore M-box 1 perfmon MAP unit select MSR.
CE8H	3304	MSR_M1_PMON_MSC_THR	Package	Uncore M-box 1 perfmon MIC THR select MSR.
CE9H	3305	MSR_M1_PMON_PGT	Package	Uncore M-box 1 perfmon PGT unit select MSR.
CEAH	3306	MSR_M1_PMON_PLD	Package	Uncore M-box 1 perfmon PLD unit select MSR.
CEBH	3307	MSR_M1_PMON_ZDP	Package	Uncore M-box 1 perfmon ZDP unit select MSR.
CF0H	3312	MSR_M1_PMON_EVNT_SEL0	Package	Uncore M-box 1 perfmon event select MSR.
CF1H	3313	MSR_M1_PMON_CTRL0	Package	Uncore M-box 1 perfmon counter MSR.
CF2H	3314	MSR_M1_PMON_EVNT_SEL1	Package	Uncore M-box 1 perfmon event select MSR.
CF3H	3315	MSR_M1_PMON_CTRL1	Package	Uncore M-box 1 perfmon counter MSR.
CF4H	3316	MSR_M1_PMON_EVNT_SEL2	Package	Uncore M-box 1 perfmon event select MSR.
CF5H	3317	MSR_M1_PMON_CTRL2	Package	Uncore M-box 1 perfmon counter MSR.
CF6H	3318	MSR_M1_PMON_EVNT_SEL3	Package	Uncore M-box 1 perfmon event select MSR.
CF7H	3319	MSR_M1_PMON_CTRL3	Package	Uncore M-box 1 perfmon counter MSR.
CF8H	3320	MSR_M1_PMON_EVNT_SEL4	Package	Uncore M-box 1 perfmon event select MSR.
CF9H	3321	MSR_M1_PMON_CTRL4	Package	Uncore M-box 1 perfmon counter MSR.
CFAH	3322	MSR_M1_PMON_EVNT_SEL5	Package	Uncore M-box 1 perfmon event select MSR.
CFBH	3323	MSR_M1_PMON_CTRL5	Package	Uncore M-box 1 perfmon counter MSR.
D00H	3328	MSR_C0_PMON_BOX_CTRL	Package	Uncore C-box 0 perfmon local box control MSR.
D01H	3329	MSR_C0_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon local box status MSR.
D02H	3330	MSR_C0_PMON_BOX_OVF_CTRL	Package	Uncore C-box 0 perfmon local box overflow control MSR.
D10H	3344	MSR_C0_PMON_EVNT_SEL0	Package	Uncore C-box 0 perfmon event select MSR.
D11H	3345	MSR_C0_PMON_CTRL0	Package	Uncore C-box 0 perfmon counter MSR.
D12H	3346	MSR_C0_PMON_EVNT_SEL1	Package	Uncore C-box 0 perfmon event select MSR.
D13H	3347	MSR_C0_PMON_CTRL1	Package	Uncore C-box 0 perfmon counter MSR.
D14H	3348	MSR_C0_PMON_EVNT_SEL2	Package	Uncore C-box 0 perfmon event select MSR.
D15H	3349	MSR_C0_PMON_CTRL2	Package	Uncore C-box 0 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D16H	3350	MSR_C0_PMON_EVNT_SEL3	Package	Uncore C-box 0 perfmon event select MSR.
D17H	3351	MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter MSR.
D18H	3352	MSR_C0_PMON_EVNT_SEL4	Package	Uncore C-box 0 perfmon event select MSR.
D19H	3353	MSR_C0_PMON_CTR4	Package	Uncore C-box 0 perfmon counter MSR.
D1AH	3354	MSR_C0_PMON_EVNT_SEL5	Package	Uncore C-box 0 perfmon event select MSR.
D1BH	3355	MSR_C0_PMON_CTR5	Package	Uncore C-box 0 perfmon counter MSR.
D20H	3360	MSR_C4_PMON_BOX_CTRL	Package	Uncore C-box 4 perfmon local box control MSR.
D21H	3361	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon local box status MSR.
D22H	3362	MSR_C4_PMON_BOX_OVF_CTRL	Package	Uncore C-box 4 perfmon local box overflow control MSR.
D30H	3376	MSR_C4_PMON_EVNT_SELO	Package	Uncore C-box 4 perfmon event select MSR.
D31H	3377	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter MSR.
D32H	3378	MSR_C4_PMON_EVNT_SEL1	Package	Uncore C-box 4 perfmon event select MSR.
D33H	3379	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter MSR.
D34H	3380	MSR_C4_PMON_EVNT_SEL2	Package	Uncore C-box 4 perfmon event select MSR.
D35H	3381	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter MSR.
D36H	3382	MSR_C4_PMON_EVNT_SEL3	Package	Uncore C-box 4 perfmon event select MSR.
D37H	3383	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter MSR.
D38H	3384	MSR_C4_PMON_EVNT_SEL4	Package	Uncore C-box 4 perfmon event select MSR.
D39H	3385	MSR_C4_PMON_CTR4	Package	Uncore C-box 4 perfmon counter MSR.
D3AH	3386	MSR_C4_PMON_EVNT_SEL5	Package	Uncore C-box 4 perfmon event select MSR.
D3BH	3387	MSR_C4_PMON_CTR5	Package	Uncore C-box 4 perfmon counter MSR.
D40H	3392	MSR_C2_PMON_BOX_CTRL	Package	Uncore C-box 2 perfmon local box control MSR.
D41H	3393	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon local box status MSR.
D42H	3394	MSR_C2_PMON_BOX_OVF_CTRL	Package	Uncore C-box 2 perfmon local box overflow control MSR.
D50H	3408	MSR_C2_PMON_EVNT_SELO	Package	Uncore C-box 2 perfmon event select MSR.
D51H	3409	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D52H	3410	MSR_C2_PMON_EVNT_SEL1	Package	Uncore C-box 2 perfmon event select MSR.
D53H	3411	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter MSR.
D54H	3412	MSR_C2_PMON_EVNT_SEL2	Package	Uncore C-box 2 perfmon event select MSR.
D55H	3413	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter MSR.
D56H	3414	MSR_C2_PMON_EVNT_SEL3	Package	Uncore C-box 2 perfmon event select MSR.
D57H	3415	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter MSR.
D58H	3416	MSR_C2_PMON_EVNT_SEL4	Package	Uncore C-box 2 perfmon event select MSR.
D59H	3417	MSR_C2_PMON_CTR4	Package	Uncore C-box 2 perfmon counter MSR.
D5AH	3418	MSR_C2_PMON_EVNT_SEL5	Package	Uncore C-box 2 perfmon event select MSR.
D5BH	3419	MSR_C2_PMON_CTR5	Package	Uncore C-box 2 perfmon counter MSR.
D60H	3424	MSR_C6_PMON_BOX_CTRL	Package	Uncore C-box 6 perfmon local box control MSR.
D61H	3425	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon local box status MSR.
D62H	3426	MSR_C6_PMON_BOX_OVF_CTRL	Package	Uncore C-box 6 perfmon local box overflow control MSR.
D70H	3440	MSR_C6_PMON_EVNT_SELO	Package	Uncore C-box 6 perfmon event select MSR.
D71H	3441	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter MSR.
D72H	3442	MSR_C6_PMON_EVNT_SEL1	Package	Uncore C-box 6 perfmon event select MSR.
D73H	3443	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter MSR.
D74H	3444	MSR_C6_PMON_EVNT_SEL2	Package	Uncore C-box 6 perfmon event select MSR.
D75H	3445	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter MSR.
D76H	3446	MSR_C6_PMON_EVNT_SEL3	Package	Uncore C-box 6 perfmon event select MSR.
D77H	3447	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter MSR.
D78H	3448	MSR_C6_PMON_EVNT_SEL4	Package	Uncore C-box 6 perfmon event select MSR.
D79H	3449	MSR_C6_PMON_CTR4	Package	Uncore C-box 6 perfmon counter MSR.
D7AH	3450	MSR_C6_PMON_EVNT_SEL5	Package	Uncore C-box 6 perfmon event select MSR.
D7BH	3451	MSR_C6_PMON_CTR5	Package	Uncore C-box 6 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D80H	3456	MSR_C1_PMON_BOX_CTRL	Package	Uncore C-box 1 perfmon local box control MSR.
D81H	3457	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon local box status MSR.
D82H	3458	MSR_C1_PMON_BOX_OVF_CTRL	Package	Uncore C-box 1 perfmon local box overflow control MSR.
D90H	3472	MSR_C1_PMON_EVNT_SELO	Package	Uncore C-box 1 perfmon event select MSR.
D91H	3473	MSR_C1_PMON_CTRL0	Package	Uncore C-box 1 perfmon counter MSR.
D92H	3474	MSR_C1_PMON_EVNT_SEL1	Package	Uncore C-box 1 perfmon event select MSR.
D93H	3475	MSR_C1_PMON_CTRL1	Package	Uncore C-box 1 perfmon counter MSR.
D94H	3476	MSR_C1_PMON_EVNT_SEL2	Package	Uncore C-box 1 perfmon event select MSR.
D95H	3477	MSR_C1_PMON_CTRL2	Package	Uncore C-box 1 perfmon counter MSR.
D96H	3478	MSR_C1_PMON_EVNT_SEL3	Package	Uncore C-box 1 perfmon event select MSR.
D97H	3479	MSR_C1_PMON_CTRL3	Package	Uncore C-box 1 perfmon counter MSR.
D98H	3480	MSR_C1_PMON_EVNT_SEL4	Package	Uncore C-box 1 perfmon event select MSR.
D99H	3481	MSR_C1_PMON_CTRL4	Package	Uncore C-box 1 perfmon counter MSR.
D9AH	3482	MSR_C1_PMON_EVNT_SEL5	Package	Uncore C-box 1 perfmon event select MSR.
D9BH	3483	MSR_C1_PMON_CTRL5	Package	Uncore C-box 1 perfmon counter MSR.
DA0H	3488	MSR_C5_PMON_BOX_CTRL	Package	Uncore C-box 5 perfmon local box control MSR.
DA1H	3489	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon local box status MSR.
DA2H	3490	MSR_C5_PMON_BOX_OVF_CTRL	Package	Uncore C-box 5 perfmon local box overflow control MSR.
DB0H	3504	MSR_C5_PMON_EVNT_SELO	Package	Uncore C-box 5 perfmon event select MSR.
DB1H	3505	MSR_C5_PMON_CTRL0	Package	Uncore C-box 5 perfmon counter MSR.
DB2H	3506	MSR_C5_PMON_EVNT_SEL1	Package	Uncore C-box 5 perfmon event select MSR.
DB3H	3507	MSR_C5_PMON_CTRL1	Package	Uncore C-box 5 perfmon counter MSR.
DB4H	3508	MSR_C5_PMON_EVNT_SEL2	Package	Uncore C-box 5 perfmon event select MSR.
DB5H	3509	MSR_C5_PMON_CTRL2	Package	Uncore C-box 5 perfmon counter MSR.
DB6H	3510	MSR_C5_PMON_EVNT_SEL3	Package	Uncore C-box 5 perfmon event select MSR.
DB7H	3511	MSR_C5_PMON_CTRL3	Package	Uncore C-box 5 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DB8H	3512	MSR_C5_PMON_EVNT_SEL4	Package	Uncore C-box 5 perfmon event select MSR.
DB9H	3513	MSR_C5_PMON_CTR4	Package	Uncore C-box 5 perfmon counter MSR.
DBAH	3514	MSR_C5_PMON_EVNT_SEL5	Package	Uncore C-box 5 perfmon event select MSR.
DBBH	3515	MSR_C5_PMON_CTR5	Package	Uncore C-box 5 perfmon counter MSR.
DC0H	3520	MSR_C3_PMON_BOX_CTRL	Package	Uncore C-box 3 perfmon local box control MSR.
DC1H	3521	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon local box status MSR.
DC2H	3522	MSR_C3_PMON_BOX_OVF_CTRL	Package	Uncore C-box 3 perfmon local box overflow control MSR.
DD0H	3536	MSR_C3_PMON_EVNT_SELO	Package	Uncore C-box 3 perfmon event select MSR.
DD1H	3537	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter MSR.
DD2H	3538	MSR_C3_PMON_EVNT_SEL1	Package	Uncore C-box 3 perfmon event select MSR.
DD3H	3539	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter MSR.
DD4H	3540	MSR_C3_PMON_EVNT_SEL2	Package	Uncore C-box 3 perfmon event select MSR.
DD5H	3541	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter MSR.
DD6H	3542	MSR_C3_PMON_EVNT_SEL3	Package	Uncore C-box 3 perfmon event select MSR.
DD7H	3543	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter MSR.
DD8H	3544	MSR_C3_PMON_EVNT_SEL4	Package	Uncore C-box 3 perfmon event select MSR.
DD9H	3545	MSR_C3_PMON_CTR4	Package	Uncore C-box 3 perfmon counter MSR.
DDAH	3546	MSR_C3_PMON_EVNT_SEL5	Package	Uncore C-box 3 perfmon event select MSR.
DDBH	3547	MSR_C3_PMON_CTR5	Package	Uncore C-box 3 perfmon counter MSR.
DE0H	3552	MSR_C7_PMON_BOX_CTRL	Package	Uncore C-box 7 perfmon local box control MSR.
DE1H	3553	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon local box status MSR.
DE2H	3554	MSR_C7_PMON_BOX_OVF_CTRL	Package	Uncore C-box 7 perfmon local box overflow control MSR.
DF0H	3568	MSR_C7_PMON_EVNT_SELO	Package	Uncore C-box 7 perfmon event select MSR.
DF1H	3569	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter MSR.
DF2H	3570	MSR_C7_PMON_EVNT_SEL1	Package	Uncore C-box 7 perfmon event select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DF3H	3571	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter MSR.
DF4H	3572	MSR_C7_PMON_EVNT_SEL2	Package	Uncore C-box 7 perfmon event select MSR.
DF5H	3573	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter MSR.
DF6H	3574	MSR_C7_PMON_EVNT_SEL3	Package	Uncore C-box 7 perfmon event select MSR.
DF7H	3575	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter MSR.
DF8H	3576	MSR_C7_PMON_EVNT_SEL4	Package	Uncore C-box 7 perfmon event select MSR.
DF9H	3577	MSR_C7_PMON_CTR4	Package	Uncore C-box 7 perfmon counter MSR.
DFAH	3578	MSR_C7_PMON_EVNT_SEL5	Package	Uncore C-box 7 perfmon event select MSR.
DFBH	3579	MSR_C7_PMON_CTR5	Package	Uncore C-box 7 perfmon counter MSR.
E00H	3584	MSR_R0_PMON_BOX_CTRL	Package	Uncore R-box 0 perfmon local box control MSR.
E01H	3585	MSR_R0_PMON_BOX_STATUS	Package	Uncore R-box 0 perfmon local box status MSR.
E02H	3586	MSR_R0_PMON_BOX_OVF_CTRL	Package	Uncore R-box 0 perfmon local box overflow control MSR.
E04H	3588	MSR_R0_PMON_IPERFO_P0	Package	Uncore R-box 0 perfmon IPERFO unit Port 0 select MSR.
E05H	3589	MSR_R0_PMON_IPERFO_P1	Package	Uncore R-box 0 perfmon IPERFO unit Port 1 select MSR.
E06H	3590	MSR_R0_PMON_IPERFO_P2	Package	Uncore R-box 0 perfmon IPERFO unit Port 2 select MSR.
E07H	3591	MSR_R0_PMON_IPERFO_P3	Package	Uncore R-box 0 perfmon IPERFO unit Port 3 select MSR.
E08H	3592	MSR_R0_PMON_IPERFO_P4	Package	Uncore R-box 0 perfmon IPERFO unit Port 4 select MSR.
E09H	3593	MSR_R0_PMON_IPERFO_P5	Package	Uncore R-box 0 perfmon IPERFO unit Port 5 select MSR.
E0AH	3594	MSR_R0_PMON_IPERFO_P6	Package	Uncore R-box 0 perfmon IPERFO unit Port 6 select MSR.
E0BH	3595	MSR_R0_PMON_IPERFO_P7	Package	Uncore R-box 0 perfmon IPERFO unit Port 7 select MSR.
E0CH	3596	MSR_R0_PMON_QLX_P0	Package	Uncore R-box 0 perfmon QLX unit Port 0 select MSR.
E0DH	3597	MSR_R0_PMON_QLX_P1	Package	Uncore R-box 0 perfmon QLX unit Port 1 select MSR.
E0EH	3598	MSR_R0_PMON_QLX_P2	Package	Uncore R-box 0 perfmon QLX unit Port 2 select MSR.
E0FH	3599	MSR_R0_PMON_QLX_P3	Package	Uncore R-box 0 perfmon QLX unit Port 3 select MSR.
E10H	3600	MSR_R0_PMON_EVNT_SELO	Package	Uncore R-box 0 perfmon event select MSR.
E11H	3601	MSR_R0_PMON_CTR0	Package	Uncore R-box 0 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E12H	3602	MSR_R0_PMON_EVNT_SEL1	Package	Uncore R-box 0 perfmon event select MSR.
E13H	3603	MSR_R0_PMON_CTR1	Package	Uncore R-box 0 perfmon counter MSR.
E14H	3604	MSR_R0_PMON_EVNT_SEL2	Package	Uncore R-box 0 perfmon event select MSR.
E15H	3605	MSR_R0_PMON_CTR2	Package	Uncore R-box 0 perfmon counter MSR.
E16H	3606	MSR_R0_PMON_EVNT_SEL3	Package	Uncore R-box 0 perfmon event select MSR.
E17H	3607	MSR_R0_PMON_CTR3	Package	Uncore R-box 0 perfmon counter MSR.
E18H	3608	MSR_R0_PMON_EVNT_SEL4	Package	Uncore R-box 0 perfmon event select MSR.
E19H	3609	MSR_R0_PMON_CTR4	Package	Uncore R-box 0 perfmon counter MSR.
E1AH	3610	MSR_R0_PMON_EVNT_SEL5	Package	Uncore R-box 0 perfmon event select MSR.
E1BH	3611	MSR_R0_PMON_CTR5	Package	Uncore R-box 0 perfmon counter MSR.
E1CH	3612	MSR_R0_PMON_EVNT_SEL6	Package	Uncore R-box 0 perfmon event select MSR.
E1DH	3613	MSR_R0_PMON_CTR6	Package	Uncore R-box 0 perfmon counter MSR.
E1EH	3614	MSR_R0_PMON_EVNT_SEL7	Package	Uncore R-box 0 perfmon event select MSR.
E1FH	3615	MSR_R0_PMON_CTR7	Package	Uncore R-box 0 perfmon counter MSR.
E20H	3616	MSR_R1_PMON_BOX_CTRL	Package	Uncore R-box 1 perfmon local box control MSR.
E21H	3617	MSR_R1_PMON_BOX_STATUS	Package	Uncore R-box 1 perfmon local box status MSR.
E22H	3618	MSR_R1_PMON_BOX_OVF_CTRL	Package	Uncore R-box 1 perfmon local box overflow control MSR.
E24H	3620	MSR_R1_PMON_IPERF1_P8	Package	Uncore R-box 1 perfmon IPERF1 unit Port 8 select MSR.
E25H	3621	MSR_R1_PMON_IPERF1_P9	Package	Uncore R-box 1 perfmon IPERF1 unit Port 9 select MSR.
E26H	3622	MSR_R1_PMON_IPERF1_P10	Package	Uncore R-box 1 perfmon IPERF1 unit Port 10 select MSR.
E27H	3623	MSR_R1_PMON_IPERF1_P11	Package	Uncore R-box 1 perfmon IPERF1 unit Port 11 select MSR.
E28H	3624	MSR_R1_PMON_IPERF1_P12	Package	Uncore R-box 1 perfmon IPERF1 unit Port 12 select MSR.
E29H	3625	MSR_R1_PMON_IPERF1_P13	Package	Uncore R-box 1 perfmon IPERF1 unit Port 13 select MSR.
E2AH	3626	MSR_R1_PMON_IPERF1_P14	Package	Uncore R-box 1 perfmon IPERF1 unit Port 14 select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2BH	3627	MSR_R1_PMON_IPERF1_P15	Package	Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR.
E2CH	3628	MSR_R1_PMON_QLX_P4	Package	Uncore R-box 1 perfmon QLX unit Port 4 select MSR.
E2DH	3629	MSR_R1_PMON_QLX_P5	Package	Uncore R-box 1 perfmon QLX unit Port 5 select MSR.
E2EH	3630	MSR_R1_PMON_QLX_P6	Package	Uncore R-box 1 perfmon QLX unit Port 6 select MSR.
E2FH	3631	MSR_R1_PMON_QLX_P7	Package	Uncore R-box 1 perfmon QLX unit Port 7 select MSR.
E30H	3632	MSR_R1_PMON_EVNT_SEL8	Package	Uncore R-box 1 perfmon event select MSR.
E31H	3633	MSR_R1_PMON_CTR8	Package	Uncore R-box 1 perfmon counter MSR.
E32H	3634	MSR_R1_PMON_EVNT_SEL9	Package	Uncore R-box 1 perfmon event select MSR.
E33H	3635	MSR_R1_PMON_CTR9	Package	Uncore R-box 1 perfmon counter MSR.
E34H	3636	MSR_R1_PMON_EVNT_SEL10	Package	Uncore R-box 1 perfmon event select MSR.
E35H	3637	MSR_R1_PMON_CTR10	Package	Uncore R-box 1 perfmon counter MSR.
E36H	3638	MSR_R1_PMON_EVNT_SEL11	Package	Uncore R-box 1 perfmon event select MSR.
E37H	3639	MSR_R1_PMON_CTR11	Package	Uncore R-box 1 perfmon counter MSR.
E38H	3640	MSR_R1_PMON_EVNT_SEL12	Package	Uncore R-box 1 perfmon event select MSR.
E39H	3641	MSR_R1_PMON_CTR12	Package	Uncore R-box 1 perfmon counter MSR.
E3AH	3642	MSR_R1_PMON_EVNT_SEL13	Package	Uncore R-box 1 perfmon event select MSR.
E3BH	3643	MSR_R1_PMON_CTR13	Package	Uncore R-box 1 perfmon counter MSR.
E3CH	3644	MSR_R1_PMON_EVNT_SEL14	Package	Uncore R-box 1 perfmon event select MSR.
E3DH	3645	MSR_R1_PMON_CTR14	Package	Uncore R-box 1 perfmon counter MSR.
E3EH	3646	MSR_R1_PMON_EVNT_SEL15	Package	Uncore R-box 1 perfmon event select MSR.
E3FH	3647	MSR_R1_PMON_CTR15	Package	Uncore R-box 1 perfmon counter MSR.
E45H	3653	MSR_B0_PMON_MATCH	Package	Uncore B-box 0 perfmon local box match MSR.
E46H	3654	MSR_B0_PMON_MASK	Package	Uncore B-box 0 perfmon local box mask MSR.
E49H	3657	MSR_S0_PMON_MATCH	Package	Uncore S-box 0 perfmon local box match MSR.
E4AH	3658	MSR_S0_PMON_MASK	Package	Uncore S-box 0 perfmon local box mask MSR.
E4DH	3661	MSR_B1_PMON_MATCH	Package	Uncore B-box 1 perfmon local box match MSR.
E4EH	3662	MSR_B1_PMON_MASK	Package	Uncore B-box 1 perfmon local box mask MSR.
E54H	3668	MSR_M0_PMON_MM_CONFIG	Package	Uncore M-box 0 perfmon local box address match/mask config MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E55H	3669	MSR_M0_PMON_ADDR_MATCH	Package	Uncore M-box 0 perfmon local box address match MSR.
E56H	3670	MSR_M0_PMON_ADDR_MASK	Package	Uncore M-box 0 perfmon local box address mask MSR.
E59H	3673	MSR_S1_PMON_MATCH	Package	Uncore S-box 1 perfmon local box match MSR.
E5AH	3674	MSR_S1_PMON_MASK	Package	Uncore S-box 1 perfmon local box mask MSR.
E5CH	3676	MSR_M1_PMON_MM_CONFIG	Package	Uncore M-box 1 perfmon local box address match/mask config MSR.
E5DH	3677	MSR_M1_PMON_ADDR_MATCH	Package	Uncore M-box 1 perfmon local box address match MSR.
E5EH	3678	MSR_M1_PMON_ADDR_MASK	Package	Uncore M-box 1 perfmon local box address mask MSR.
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."

2.9 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES (BASED ON WESTMERE MICROARCHITECTURE)

The Intel® Xeon® Processor 5600 Series (based on Westmere microarchitecture) supports the MSR interfaces listed in Table 2-15, Table 2-16, plus additional MSR listed in Table 2-18. These MSRs apply to Intel Core i7, i5 and i3 processor family with CPUID signature DisplayFamily_DisplayModel of 06_25H and 06_2CH, see Table 2-1.

Table 2-18. Additional MSRs Supported by Intel Processors Based on Westmere Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)

Table 2-18. Additional MSRs Supported by Intel Processors Based on Westmere Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		63:48		Reserved
1BOH	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.

2.10 MSRS IN THE INTEL® XEON® PROCESSOR E7 FAMILY (BASED ON WESTMERE MICROARCHITECTURE)

The Intel® Xeon® Processor E7 Family (based on Westmere microarchitecture) 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 with DisplayFamily_DisplayModel of 06_2FH.

Table 2-19. Additional MSRs Supported by Intel® Xeon® Processor E7 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.

Table 2-19. Additional MSRs Supported by Intel® Xeon® Processor E7 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:2		Reserved
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved Attempt to read/write will cause #UD.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
F40H	3904	MSR_C8_PMON_BOX_CTRL	Package	Uncore C-box 8 perfmon local box control MSR.
F41H	3905	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-box 8 perfmon local box status MSR.
F42H	3906	MSR_C8_PMON_BOX_OVF_CTRL	Package	Uncore C-box 8 perfmon local box overflow control MSR.
F50H	3920	MSR_C8_PMON_EVNT_SELO	Package	Uncore C-box 8 perfmon event select MSR.
F51H	3921	MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter MSR.
F52H	3922	MSR_C8_PMON_EVNT_SEL1	Package	Uncore C-box 8 perfmon event select MSR.
F53H	3923	MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter MSR.
F54H	3924	MSR_C8_PMON_EVNT_SEL2	Package	Uncore C-box 8 perfmon event select MSR.
F55H	3925	MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter MSR.
F56H	3926	MSR_C8_PMON_EVNT_SEL3	Package	Uncore C-box 8 perfmon event select MSR.
F57H	3927	MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter MSR.
F58H	3928	MSR_C8_PMON_EVNT_SEL4	Package	Uncore C-box 8 perfmon event select MSR.
F59H	3929	MSR_C8_PMON_CTR4	Package	Uncore C-box 8 perfmon counter MSR.
F5AH	3930	MSR_C8_PMON_EVNT_SEL5	Package	Uncore C-box 8 perfmon event select MSR.
F5BH	3931	MSR_C8_PMON_CTR5	Package	Uncore C-box 8 perfmon counter MSR.
FC0H	4032	MSR_C9_PMON_BOX_CTRL	Package	Uncore C-box 9 perfmon local box control MSR.
FC1H	4033	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-box 9 perfmon local box status MSR.
FC2H	4034	MSR_C9_PMON_BOX_OVF_CTRL	Package	Uncore C-box 9 perfmon local box overflow control MSR.
FD0H	4048	MSR_C9_PMON_EVNT_SELO	Package	Uncore C-box 9 perfmon event select MSR.
FD1H	4049	MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter MSR.
FD2H	4050	MSR_C9_PMON_EVNT_SEL1	Package	Uncore C-box 9 perfmon event select MSR.
FD3H	4051	MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter MSR.

Table 2-19. Additional MSRs Supported by Intel® Xeon® Processor E7 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
FD4H	4052	MSR_C9_PMON_EVNT_SEL2	Package	Uncore C-box 9 perfmon event select MSR.
FD5H	4053	MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter MSR.
FD6H	4054	MSR_C9_PMON_EVNT_SEL3	Package	Uncore C-box 9 perfmon event select MSR.
FD7H	4055	MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter MSR.
FD8H	4056	MSR_C9_PMON_EVNT_SEL4	Package	Uncore C-box 9 perfmon event select MSR.
FD9H	4057	MSR_C9_PMON_CTR4	Package	Uncore C-box 9 perfmon counter MSR.
FDAH	4058	MSR_C9_PMON_EVNT_SEL5	Package	Uncore C-box 9 perfmon event select MSR.
FDBH	4059	MSR_C9_PMON_CTR5	Package	Uncore C-box 9 perfmon counter MSR.

2.11 MSRS IN 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 with DisplayFamily_DisplayModel of 06_2AH, 06_2DH, see Table 2-1. Additional MSRs specific to 06_2AH are listed in Table 2-21.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination" and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter" and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location" and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Count SMIs.
		63:32		Reserved.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 2-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 2-2.
C5H	197	IA32_PMC4	Core	Performance Counter Register (if core not shared by threads)
C6H	198	IA32_PMC5	Core	Performance Counter Register (if core not shared by threads)
C7H	199	IA32_PMC6	Core	Performance Counter Register (if core not shared by threads)
C8H	200	IA32_PMC7	Core	Performance Counter Register (if core not shared by threads)
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.
		28		Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.
		63:29		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org .
		15:0		LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		C-State Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.
		63:19		Reserved
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		0		RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSEL0	Thread	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 2-2.
18AH	394	IA32_PERFEVTSEL4	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 4.
18BH	395	IA32_PERFEVTSEL5	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 5.
18CH	396	IA32_PERFEVTSEL6	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 6.
18DH	397	IA32_PERFEVTSEL7	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 7.
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15:0		Current Performance State Value
		63:16		Reserved
198H	408	MSR_PERF_STATUS	Package	Performance Status
		47:32		Core Voltage (R/O) P-state core voltage can be computed by $MSR_PERF_STATUS[37:32] * (\text{float}) 1/(2^{13})$.
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		3:0		On demand Clock Modulation Duty Cycle (R/W) In 6.25% increment.
		4		On demand Clock Modulation Enable (R/W)
		63:5		Reserved
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal Threshold #1 Status (R/O) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.		

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		10		Power Limitation Status (R/O) See Table 2-2.
		11		Power Limitation Log (R/WCO) See Table 2-2.
		15:12		Reserved
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.
		31		Reading Valid (R/O) See Table 2-2.
		63:32		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 2-2
		6:1		Reserved
		7	Thread	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Thread	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12	Thread	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Thread	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
		22	Thread	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Thread	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
34	Thread	XD Bit Disable (R/W) See Table 2-2.		

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		37:35		Reserved
		38	Package	<p>Turbo Mode Disable (R/W)</p> <p>When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0).</p> <p>When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled.</p> <p>Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.</p>
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Unique	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.
		63:24		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1	Core	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.
		63:4		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1A7H	422	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
63:9		Reserved		
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.
		0		LBR: Last Branch Record
		1		BTF
		5:2		Reserved
		6		TR: Branch Trace
		7		BTS: Log Branch Trace Message to BTS buffer
		8		BTINT
		9		BTS_OFF_OS
		10		BTS_OFF_USER
		11		FREEZE_LBR_ON_PMI
		12		FREEZE_PERFMON_ON_PMI
		13		ENABLE_UNCORE_PMI
		14		FREEZE_WHILE_SMM
63:15		Reserved		
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org .
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MCO_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	Always 0 (CMCI not supported).
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format See Table 2-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs See Table 2-2.
		11:8		PEBS_REC_FORMAT See Table 2-2.
		12		SMM_FREEZE See Table 2-2.
	63:13		Reserved	
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0	Thread	Ovf_PMC0
		1	Thread	Ovf_PMC1
		2	Thread	Ovf_PMC2
		3	Thread	Ovf_PMC3
		4	Core	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5	Core	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Core	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)
		7	Core	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)
		31:8		Reserved
		32	Thread	Ovf_FixedCtr0
		33	Thread	Ovf_FixedCtr1
		34	Thread	Ovf_FixedCtr2
		60:35		Reserved
		61	Thread	Ovf_Uncore
		62	Thread	Ovf_BufDSSAVE
		63	Thread	CondChgd
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0	Thread	Set 1 to enable PMC0 to count.
		1	Thread	Set 1 to enable PMC1 to count.
		2	Thread	Set 1 to enable PMC2 to count.
		3	Thread	Set 1 to enable PMC3 to count.
		4	Core	Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4).
		5	Core	Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5).
		6	Core	Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6).
		7	Core	Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to enable FixedCtr0 to count.
		33	Thread	Set 1 to enable FixedCtr1 to count.
		34	Thread	Set 1 to enable FixedCtr2 to count.
		63:35		Reserved
390H	912	IA32_PERF_GLOBAL_OVF_CTRL		See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0	Thread	Set 1 to clear Ovf_PMC0.
		1	Thread	Set 1 to clear Ovf_PMC1.
		2	Thread	Set 1 to clear Ovf_PMC2.
		3	Thread	Set 1 to clear Ovf_PMC3.
		4	Core	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).
		5	Core	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).
6	Core	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).		

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7	Core	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to clear Ovf_FixedCtr0.
		33	Thread	Set 1 to clear Ovf_FixedCtr1.
		34	Thread	Set 1 to clear Ovf_FixedCtr2.
		60:35		Reserved
		61	Thread	Set 1 to clear Ovf_Uncore.
		62	Thread	Set 1 to clear Ovf_BufDSSAVE.
		63	Thread	Set 1 to clear CondChgd.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 19.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		62:36		Reserved
		63		Enable Precise Store (R/W)
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 19.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
3FEH	1022	MSR_CORE_C7_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C7 Residency Counter (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.
400H	1024	IA32_MCO_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
402H	1026	IA32_MCO_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
403H	1027	IA32_MCO_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
406H	1030	IA32_MC1_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
407H	1031	IA32_MC1_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
40BH	1035	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRS."
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRS."
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRS."
		0		PCU Hardware Error (R/W) When set, enables signaling of PCU hardware detected errors.
		1		PCU Controller Error (R/W) When set, enables signaling of PCU controller detected errors.
		2		PCU Firmware Error (R/W) When set, enables signaling of PCU firmware detected errors.
		63:2		Reserved
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	Thread	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTL	Thread	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTL	Thread	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLSS2	Thread	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Thread	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLSS	Thread	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLSS	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLSS	Thread	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLSS	Thread	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 2-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 2-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 2-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 2-2.
4C8H	1224	IA32_A_PMC7	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."
60AH	1546	MSR_PKGC3_IRTL	Package	Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60BH	1547	MSR_PKG_C6_IRTL	Package	Package C6 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C6 to a C0 state, where an interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W) See Section 14.10.3, "Package RAPL Domain."
638H	1592	MSR_PPO_POWER_LIMIT	Package	PPO RAPL Power Limit Control (R/W) See Section 14.10.4, "PPO/PP1 RAPL Domains."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.9.1 and record format in Section 17.4.8.1.
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table 2-2.
802H-83FH	2050-2111	X2APIC MSRs	Thread	See Table 2-2.
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 17.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

2.11.1 MSRs in the 2nd Generation Intel® Core™ Processor Family Based on Sandy Bridge Microarchitecture

Table 2-21 and Table 2-22 list model-specific registers (MSRs) that are specific to the 2nd generation Intel® Core™ processor family based on the Sandy Bridge microarchitecture. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2AH; see Table 2-1.

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60CH	1548	MSR_PKGC7_IRTL	Package	Package C7 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 14.10.4, "PPO/PP1 RAPL Domains."
63AH	1594	MSR_PPO_POLICY	Package	PPO Balance Policy (R/W) See Section 14.10.4, "PPO/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	PP1 RAPL Power Limit Control (R/W) See Section 14.10.4, "PPO/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 14.10.4, "PPO/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	PP1 Balance Policy (R/W) See Section 14.10.4, "PPO/PP1 RAPL Domains."
See Table 2-20, Table 2-21, and Table 2-22 for MSR definitions applicable to processors with CPUID signature 06_2AH.				

Table 2-22 lists the MSRs of uncore PMU for Intel processors with CPUID signature 06_2AH.

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4 select.
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
	63:32		Reserved	
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Report the number of C-Box units with performance counters, including processor cores and processor graphics.
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, Counter 1 Event Select MSR

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
702H	1794	MSR_UNC_CBO_0_PERFEVTSEL2	Package	Uncore C-Box 0, Counter 2 Event Select MSR
703H	1795	MSR_UNC_CBO_0_PERFEVTSEL3	Package	Uncore C-Box 0, Counter 3 Event Select MSR
705H	1797	MSR_UNC_CBO_0_UNIT_STATUS	Package	Uncore C-Box 0, Unit Status for Counter 0-3
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
708H	1800	MSR_UNC_CBO_0_PERFCTR2	Package	Uncore C-Box 0, Performance Counter 2
709H	1801	MSR_UNC_CBO_0_PERFCTR3	Package	Uncore C-Box 0, Performance Counter 3
710H	1808	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
712H	1810	MSR_UNC_CBO_1_PERFEVTSEL2	Package	Uncore C-Box 1, Counter 2 Event Select MSR
713H	1811	MSR_UNC_CBO_1_PERFEVTSEL3	Package	Uncore C-Box 1, Counter 3 Event Select MSR
715H	1813	MSR_UNC_CBO_1_UNIT_STATUS	Package	Uncore C-Box 1, Unit Status for Counter 0-3
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
718H	1816	MSR_UNC_CBO_1_PERFCTR2	Package	Uncore C-Box 1, Performance Counter 2
719H	1817	MSR_UNC_CBO_1_PERFCTR3	Package	Uncore C-Box 1, Performance Counter 3
720H	1824	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
722H	1826	MSR_UNC_CBO_2_PERFEVTSEL2	Package	Uncore C-Box 2, Counter 2 Event Select MSR
723H	1827	MSR_UNC_CBO_2_PERFEVTSEL3	Package	Uncore C-Box 2, Counter 3 Event Select MSR
725H	1829	MSR_UNC_CBO_2_UNIT_STATUS	Package	Uncore C-Box 2, Unit Status for Counter 0-3
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
728H	1832	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, Performance Counter 2
729H	1833	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, Performance Counter 3
730H	1840	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
732H	1842	MSR_UNC_CBO_3_PERFEVTSEL2	Package	Uncore C-Box 3, Counter 2 Event Select MSR
733H	1843	MSR_UNC_CBO_3_PERFEVTSEL3	Package	Uncore C-Box 3, counter 3 Event Select MSR
735H	1845	MSR_UNC_CBO_3_UNIT_STATUS	Package	Uncore C-Box 3, Unit Status for Counter 0-3
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
738H	1848	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, Performance Counter 2
739H	1849	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, Performance Counter 3
740H	1856	MSR_UNC_CBO_4_PERFEVTSEL0	Package	Uncore C-Box 4, Counter 0 Event Select MSR

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
741H	1857	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
742H	1858	MSR_UNC_CBO_4_PERFEVTSEL2	Package	Uncore C-Box 4, Counter 2 Event Select MSR
743H	1859	MSR_UNC_CBO_4_PERFEVTSEL3	Package	Uncore C-Box 4, Counter 3 Event Select MSR
745H	1861	MSR_UNC_CBO_4_UNIT_STATUS	Package	Uncore C-Box 4, Unit status for Counter 0-3
746H	1862	MSR_UNC_CBO_4_PERFCTRO	Package	Uncore C-Box 4, Performance Counter 0
747H	1863	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
748H	1864	MSR_UNC_CBO_4_PERFCTR2	Package	Uncore C-Box 4, Performance Counter 2
749H	1865	MSR_UNC_CBO_4_PERFCTR3	Package	Uncore C-Box 4, Performance Counter 3

2.11.2 MSRs in the Intel® Xeon® Processor E5 Family Based on Sandy Bridge Microarchitecture

Table 2-23 lists additional model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family based on the Sandy Bridge microarchitecture. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH, and also support MSRs listed in Table 2-20 and Table 2-24.

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, R/W if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 cores active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 cores active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 cores active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 cores active.

**Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family
Based on Sandy Bridge Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 cores active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 cores active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 cores active.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
39CH	924	MSR_PEBS_NUM_ALT	Package	ENABLE_PEBS_NUM_ALT (R/W)
		0		ENABLE_PEBS_NUM_ALT (R/W) Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see https://perfmon-events.intel.com/ .
		63:1		Reserved, must be zero.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	IA32_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

**Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family
Based on Sandy Bridge Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
41DH	1053	IA32_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41EH	1054	IA32_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	IA32_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
422H	1058	IA32_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	IA32_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
426H	1062	IA32_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	IA32_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	IA32_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42AH	1066	IA32_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	IA32_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	IA32_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42EH	1070	IA32_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	IA32_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	IA32_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
432H	1074	IA32_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	IA32_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	IA32_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
436H	1078	IA32_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	IA32_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	IA32_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
43AH	1082	IA32_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	IA32_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."

**Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family
Based on Sandy Bridge Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	IA32_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
43EH	1086	IA32_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	IA32_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	IA32_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
442H	1090	IA32_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	IA32_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
446H	1094	IA32_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44AH	1098	IA32_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44EH	1102	IA32_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
613H	1555	MSR_PKG_PERF_STATUS	Package	Package RAPL Perf Status (R/O)
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 14.10.4, "PPO/PP1 RAPL Domains."

See Table 2-20, Table 2-23, and Table 2-24 for MSR definitions applicable to processors with CPUID signature 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 detail of the uncore PMU, refer to Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C08H	3080	MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-box UCLK Fixed Counter Control
C09H	3081	MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-box UCLK Fixed Counter
C10H	3088	MSR_U_PMON_EVNTSELO	Package	Uncore U-box Perfmon Event Select for U-box Counter 0
C11H	3089	MSR_U_PMON_EVNTSEL1	Package	Uncore U-box Perfmon Event Select for U-box Counter 1
C16H	3094	MSR_U_PMON_CTR0	Package	Uncore U-box Perfmon Counter 0
C17H	3095	MSR_U_PMON_CTR1	Package	Uncore U-box Perfmon Counter 1
C24H	3108	MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU Perfmon for PCU-box-wide Control
C30H	3120	MSR_PCU_PMON_EVNTSELO	Package	Uncore PCU Perfmon Event Select for PCU Counter 0
C31H	3121	MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU Perfmon Event Select for PCU Counter 1
C32H	3122	MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU Perfmon Event Select for PCU Counter 2
C33H	3123	MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU Perfmon Event Select for PCU Counter 3
C34H	3124	MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU Perfmon box-wide Filter
C36H	3126	MSR_PCU_PMON_CTR0	Package	Uncore PCU Perfmon Counter 0
C37H	3127	MSR_PCU_PMON_CTR1	Package	Uncore PCU Perfmon Counter 1
C38H	3128	MSR_PCU_PMON_CTR2	Package	Uncore PCU Perfmon Counter 2
C39H	3129	MSR_PCU_PMON_CTR3	Package	Uncore PCU Perfmon Counter 3
D04H	3332	MSR_CO_PMON_BOX_CTL	Package	Uncore C-box 0 Perfmon Local Box Wide Control
D10H	3344	MSR_CO_PMON_EVNTSELO	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 0
D11H	3345	MSR_CO_PMON_EVNTSEL1	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 1
D12H	3346	MSR_CO_PMON_EVNTSEL2	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 2
D13H	3347	MSR_CO_PMON_EVNTSEL3	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 3
D14H	3348	MSR_CO_PMON_BOX_FILTER	Package	Uncore C-box 0 Perfmon Box Wide Filter
D16H	3350	MSR_CO_PMON_CTR0	Package	Uncore C-box 0 Perfmon Counter 0
D17H	3351	MSR_CO_PMON_CTR1	Package	Uncore C-box 0 Perfmon Counter 1
D18H	3352	MSR_CO_PMON_CTR2	Package	Uncore C-box 0 Perfmon Counter 2
D19H	3353	MSR_CO_PMON_CTR3	Package	Uncore C-box 0 Perfmon Counter 3
D24H	3364	MSR_C1_PMON_BOX_CTL	Package	Uncore C-box 1 Perfmon Local Box Wide Control

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D30H	3376	MSR_C1_PMON_EVNTSEL0	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 0
D31H	3377	MSR_C1_PMON_EVNTSEL1	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 1
D32H	3378	MSR_C1_PMON_EVNTSEL2	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 2
D33H	3379	MSR_C1_PMON_EVNTSEL3	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 3
D34H	3380	MSR_C1_PMON_BOX_FILTER	Package	Uncore C-box 1 Perfmon Box Wide Filter
D36H	3382	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 Perfmon Counter 0
D37H	3383	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 Perfmon Counter 1
D38H	3384	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 Perfmon Counter 2
D39H	3385	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 Perfmon Counter 3
D44H	3396	MSR_C2_PMON_BOX_CTL	Package	Uncore C-box 2 Perfmon Local Box Wide Control
D50H	3408	MSR_C2_PMON_EVNTSEL0	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 0
D51H	3409	MSR_C2_PMON_EVNTSEL1	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 1
D52H	3410	MSR_C2_PMON_EVNTSEL2	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 2
D53H	3411	MSR_C2_PMON_EVNTSEL3	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 3
D54H	3412	MSR_C2_PMON_BOX_FILTER	Package	Uncore C-box 2 Perfmon Box Wide Filter
D56H	3414	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 Perfmon Counter 0
D57H	3415	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 Perfmon Counter 1
D58H	3416	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 Perfmon Counter 2
D59H	3417	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 Perfmon Counter 3
D64H	3428	MSR_C3_PMON_BOX_CTL	Package	Uncore C-box 3 Perfmon Local Box Wide Control
D70H	3440	MSR_C3_PMON_EVNTSEL0	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 0
D71H	3441	MSR_C3_PMON_EVNTSEL1	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 1
D72H	3442	MSR_C3_PMON_EVNTSEL2	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 2
D73H	3443	MSR_C3_PMON_EVNTSEL3	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 3
D74H	3444	MSR_C3_PMON_BOX_FILTER	Package	Uncore C-box 3 Perfmon Box Wide Filter
D76H	3446	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 Perfmon Counter 0
D77H	3447	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 Perfmon Counter 1
D78H	3448	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 Perfmon Counter 2
D79H	3449	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 Perfmon Counter 3

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D84H	3460	MSR_C4_PMON_BOX_CTL	Package	Uncore C-box 4 Perfmon Local Box Wide Control
D90H	3472	MSR_C4_PMON_EVNTSEL0	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 0
D91H	3473	MSR_C4_PMON_EVNTSEL1	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 1
D92H	3474	MSR_C4_PMON_EVNTSEL2	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 2
D93H	3475	MSR_C4_PMON_EVNTSEL3	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 3
D94H	3476	MSR_C4_PMON_BOX_FILTER	Package	Uncore C-box 4 Perfmon Box Wide Filter
D96H	3478	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 Perfmon Counter 0
D97H	3479	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 Perfmon Counter 1
D98H	3480	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 Perfmon Counter 2
D99H	3481	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 Perfmon Counter 3
DA4H	3492	MSR_C5_PMON_BOX_CTL	Package	Uncore C-box 5 Perfmon Local Box Wide Control
DB0H	3504	MSR_C5_PMON_EVNTSEL0	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 0
DB1H	3505	MSR_C5_PMON_EVNTSEL1	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 1
DB2H	3506	MSR_C5_PMON_EVNTSEL2	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 2
DB3H	3507	MSR_C5_PMON_EVNTSEL3	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 3
DB4H	3508	MSR_C5_PMON_BOX_FILTER	Package	Uncore C-box 5 Perfmon Box Wide Filter
DB6H	3510	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 Perfmon Counter 0
DB7H	3511	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 Perfmon Counter 1
DB8H	3512	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 Perfmon Counter 2
DB9H	3513	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 Perfmon Counter 3
DC4H	3524	MSR_C6_PMON_BOX_CTL	Package	Uncore C-box 6 Perfmon Local Box Wide Control
DD0H	3536	MSR_C6_PMON_EVNTSEL0	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 0
DD1H	3537	MSR_C6_PMON_EVNTSEL1	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 1
DD2H	3538	MSR_C6_PMON_EVNTSEL2	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 2
DD3H	3539	MSR_C6_PMON_EVNTSEL3	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 3
DD4H	3540	MSR_C6_PMON_BOX_FILTER	Package	Uncore C-box 6 Perfmon Box Wide Filter
DD6H	3542	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 Perfmon Counter 0
DD7H	3543	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 Perfmon Counter 1
DD8H	3544	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 Perfmon Counter 2

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD9H	3545	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 Perfmon Counter 3
DE4H	3556	MSR_C7_PMON_BOX_CTL	Package	Uncore C-box 7 Perfmon Local Box Wide Control
DF0H	3568	MSR_C7_PMON_EVNTSEL0	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 0
DF1H	3569	MSR_C7_PMON_EVNTSEL1	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 1
DF2H	3570	MSR_C7_PMON_EVNTSEL2	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 2
DF3H	3571	MSR_C7_PMON_EVNTSEL3	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 3
DF4H	3572	MSR_C7_PMON_BOX_FILTER	Package	Uncore C-box 7 Perfmon Box Wide Filter
DF6H	3574	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 Perfmon Counter 0
DF7H	3575	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 Perfmon Counter 1
DF8H	3576	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 Perfmon Counter 2
DF9H	3577	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 Perfmon Counter 3

2.12 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY (BASED ON IVY BRIDGE MICROARCHITECTURE)

The 3rd generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v2 product family based on Ivy Bridge microarchitecture support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-25. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3AH.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (Based on Ivy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

**Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors
(Based on Ivy Bridge Microarchitecture) (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates that TDP Limit for Turbo mode is not programmable.
		31:30		Reserved
		32	Package	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.
		34:33	Package	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved
		39:35		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		55:48	Package	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.

**Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors
(Based on Ivy Bridge Microarchitecture) (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.
		28		Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.
		63:29		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O)
		7:0		Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O)
		14:0		PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.
		15		Reserved
		23:16		Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.

**Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors
(Based on Ivy Bridge Microarchitecture) (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		47		Reserved
		62:48		PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O)
		14:0		PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W)
		1:0		TDP_LEVEL (RW/L) System BIOS can program this field.
		30:2		Reserved.
		31		Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		7:0		MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.
		30:8		Reserved
		31		TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
See Table 2-20, Table 2-21 and Table 2-22 for other MSR definitions applicable to processors with CPUID signature 06_3AH.				

2.12.1 MSRs In 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 with DisplayFamily_DisplayModel 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 Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/W) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		22:16		Reserved
		23	Package	PPIN_CAP (R/O) When set to 1, indicates that Protected Processor Inventory Number (PPIN) capability can be enabled for a privileged system inventory agent to read PPIN from MSR_PPIN. When set to 0, PPIN capability is not supported. An attempt to access MSR_PPIN_CTL or MSR_PPIN will cause #GP.
		27:24		Reserved
	28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		30	Package	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved
		15		CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:16		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		Reserved
		26		MCG_ELOG_P
		63:27		Reserved
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R/O) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.
		27:24		TCC Activation Offset (R/W) Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO.[30] is set.
		63:28		Reserved
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		63:32		Reserved
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
296H	662	IA32_MC22_CTL2	Package	See Table 2-2.
297H	663	IA32_MC23_CTL2	Package	See Table 2-2.
298H	664	IA32_MC24_CTL2	Package	See Table 2-2.
299H	665	IA32_MC25_CTL2	Package	See Table 2-2.
29AH	666	IA32_MC26_CTL2	Package	See Table 2-2.
29BH	667	IA32_MC27_CTL2	Package	See Table 2-2.
29CH	668	IA32_MC28_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
458H	1112	IA32_MC22_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
459H	1113	IA32_MC22_STATUS	Package	
45AH	1114	IA32_MC22_ADDR	Package	
45BH	1115	IA32_MC22_MISC	Package	

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
45CH	1116	IA32_MC23_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
45DH	1117	IA32_MC23_STATUS	Package	
45EH	1118	IA32_MC23_ADDR	Package	
45FH	1119	IA32_MC23_MISC	Package	
460H	1120	IA32_MC24_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
461H	1121	IA32_MC24_STATUS	Package	
462H	1122	IA32_MC24_ADDR	Package	
463H	1123	IA32_MC24_MISC	Package	
464H	1124	IA32_MC25_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
465H	1125	IA32_MC25_STATUS	Package	
466H	1126	IA32_MC25_ADDR	Package	
467H	1127	IA32_MC25_MISC	Package	
468H	1128	IA32_MC26_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
469H	1129	IA32_MC26_STATUS	Package	
46AH	1130	IA32_MC26_ADDR	Package	
46BH	1131	IA32_MC26_MISC	Package	
46CH	1132	IA32_MC27_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
46DH	1133	IA32_MC27_STATUS	Package	
46EH	1134	IA32_MC27_ADDR	Package	
46FH	1135	IA32_MC27_MISC	Package	
470H	1136	IA32_MC28_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
471H	1137	IA32_MC28_STATUS	Package	
472H	1138	IA32_MC28_ADDR	Package	
473H	1139	IA32_MC28_MISC	Package	
613H	1555	MSR_PKG_PERF_STATUS	Package	Package RAPL Perf Status (R/O)
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 14.10.4, "PPO/PP1 RAPL Domains."

See Table 2-20, for other MSR definitions applicable to Intel Xeon processor E5 v2 with CPUID signature 06_3EH.

2.12.2 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family

Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with CPUID DisplayFamily_DisplayModel signature 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 Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)
		63:16		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		63:25		Reserved
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status (R/WO)
		0		RIPV
		1		EIPV
		2		MCIP
		3		LMCE Signaled
		63:4		Reserved
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10core active.

Table 2-27. Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		39:32	Package	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.
		47:40	Package	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.
		55:48	Package	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.
		62:56		Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
29DH	669	IA32_MC29_CTL2	Package	See Table 2-2.
29EH	670	IA32_MC30_CTL2	Package	See Table 2-2.
29FH	671	IA32_MC31_CTL2	Package	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 19.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		<i>n</i> :0		Enable PEBS on IA32_PMCx. (R/W)
		31: <i>n</i> +1		Reserved
		32+ <i>m</i> :32		Enable Load Latency on IA32_PMCx. (R/W)
		63:33+ <i>m</i>		Reserved
41BH	1051	IA32_MC6_MISC	Package	Misc MAC Information of Integrated I/O (R/O) See Section 15.3.2.4.
		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
474H	1140	IA32_MC29_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
475H	1141	IA32_MC29_STATUS	Package	
476H	1142	IA32_MC29_ADDR	Package	
477H	1143	IA32_MC29_MISC	Package	

Table 2-27. Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
478H	1144	IA32_MC30_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
479H	1145	IA32_MC30_STATUS	Package	
47AH	1146	IA32_MC30_ADDR	Package	
47BH	1147	IA32_MC30_MISC	Package	
47CH	1148	IA32_MC31_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
47DH	1149	IA32_MC31_STATUS	Package	
47EH	1150	IA32_MC31_ADDR	Package	
47FH	1147	IA32_MC31_MISC	Package	

See Table 2-20, Table 2-26 for other MSR definitions applicable to Intel Xeon processor E7 v2 with CPUID signature 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 with DisplayFamily_DisplayModel of 06_3EH.

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C00H	3072	MSR_PMON_GLOBAL_CTL	Package	Uncore Perfmon Per-Socket Global Control
C01H	3073	MSR_PMON_GLOBAL_STATUS	Package	Uncore Perfmon Per-Socket Global Status
C06H	3078	MSR_PMON_GLOBAL_CONFIG	Package	Uncore Perfmon Per-Socket Global Configuration
C15H	3093	MSR_U_PMON_BOX_STATUS	Package	Uncore U-box Perfmon U-Box Wide Status
C35H	3125	MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU Perfmon Box Wide Status
D1AH	3354	MSR_C0_PMON_BOX_FILTER1	Package	Uncore C-Box 0 Perfmon Box Wide Filter1
D3AH	3386	MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-Box 1 Perfmon Box Wide Filter1
D5AH	3418	MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-Box 2 Perfmon Box Wide Filter1
D7AH	3450	MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-Box 3 Perfmon Box Wide Filter1
D9AH	3482	MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-Box 4 Perfmon Box Wide Filter1
DBAH	3514	MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-Box 5 Perfmon Box Wide Filter1
DDAH	3546	MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-Box 6 Perfmon Box Wide Filter1
DFAH	3578	MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-Box 7 Perfmon Box Wide Filter1
E04H	3588	MSR_C8_PMON_BOX_CTL	Package	Uncore C-Box 8 Perfmon Local Box Wide Control

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E10H	3600	MSR_C8_PMON_EVNTSEL0	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0
E11H	3601	MSR_C8_PMON_EVNTSEL1	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1
E12H	3602	MSR_C8_PMON_EVNTSEL2	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2
E13H	3603	MSR_C8_PMON_EVNTSEL3	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3
E14H	3604	MSR_C8_PMON_BOX_FILTER	Package	Uncore C-Box 8 Perfmon Box Wide Filter
E16H	3606	MSR_C8_PMON_CTRL0	Package	Uncore C-Box 8 Perfmon Counter 0
E17H	3607	MSR_C8_PMON_CTRL1	Package	Uncore C-Box 8 Perfmon Counter 1
E18H	3608	MSR_C8_PMON_CTRL2	Package	Uncore C-Box 8 Perfmon Counter 2
E19H	3609	MSR_C8_PMON_CTRL3	Package	Uncore C-Box 8 Perfmon Counter 3
E1AH	3610	MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-Box 8 Perfmon Box Wide Filter1
E24H	3620	MSR_C9_PMON_BOX_CTL	Package	Uncore C-Box 9 Perfmon Local Box Wide Control
E30H	3632	MSR_C9_PMON_EVNTSEL0	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 0
E31H	3633	MSR_C9_PMON_EVNTSEL1	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 1
E32H	3634	MSR_C9_PMON_EVNTSEL2	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 2
E33H	3635	MSR_C9_PMON_EVNTSEL3	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 3
E34H	3636	MSR_C9_PMON_BOX_FILTER	Package	Uncore C-Box 9 Perfmon Box Wide Filter
E36H	3638	MSR_C9_PMON_CTRL0	Package	Uncore C-Box 9 Perfmon Counter 0
E37H	3639	MSR_C9_PMON_CTRL1	Package	Uncore C-Box 9 Perfmon Counter 1
E38H	3640	MSR_C9_PMON_CTRL2	Package	Uncore C-Box 9 Perfmon Counter 2
E39H	3641	MSR_C9_PMON_CTRL3	Package	Uncore C-Box 9 Perfmon Counter 3
E3AH	3642	MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-Box 9 Perfmon Box Wide Filter1
E44H	3652	MSR_C10_PMON_BOX_CTL	Package	Uncore C-Box 10 Perfmon Local Box Wide Control
E50H	3664	MSR_C10_PMON_EVNTSEL0	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0
E51H	3665	MSR_C10_PMON_EVNTSEL1	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1
E52H	3666	MSR_C10_PMON_EVNTSEL2	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2
E53H	3667	MSR_C10_PMON_EVNTSEL3	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3
E54H	3668	MSR_C10_PMON_BOX_FILTER	Package	Uncore C-Box 10 Perfmon Box Wide Filter
E56H	3670	MSR_C10_PMON_CTRL0	Package	Uncore C-Box 10 Perfmon Counter 0
E57H	3671	MSR_C10_PMON_CTRL1	Package	Uncore C-Box 10 Perfmon Counter 1

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E58H	3672	MSR_C10_PMON_CTR2	Package	Uncore C-Box 10 Perfmon Counter 2
E59H	3673	MSR_C10_PMON_CTR3	Package	Uncore C-Box 10 Perfmon Counter 3
E5AH	3674	MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-Box 10 Perfmon Box Wide Filter1
E64H	3684	MSR_C11_PMON_BOX_CTL	Package	Uncore C-Box 11 Perfmon Local Box Wide Control
E70H	3696	MSR_C11_PMON_EVNTSELO	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0
E71H	3697	MSR_C11_PMON_EVNTSEL1	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1
E72H	3698	MSR_C11_PMON_EVNTSEL2	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2
E73H	3699	MSR_C11_PMON_EVNTSEL3	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 3
E74H	3700	MSR_C11_PMON_BOX_FILTER	Package	Uncore C-Box 11 Perfmon Box Wide Filter
E76H	3702	MSR_C11_PMON_CTR0	Package	Uncore C-Box 11 Perfmon Counter 0
E77H	3703	MSR_C11_PMON_CTR1	Package	Uncore C-Box 11 Perfmon Counter 1
E78H	3704	MSR_C11_PMON_CTR2	Package	Uncore C-Box 11 Perfmon Counter 2
E79H	3705	MSR_C11_PMON_CTR3	Package	Uncore C-Box 11 Perfmon Counter 3
E7AH	3706	MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-Box 11 Perfmon Box Wide Filter1
E84H	3716	MSR_C12_PMON_BOX_CTL	Package	Uncore C-Box 12 Perfmon Local Box Wide Control
E90H	3728	MSR_C12_PMON_EVNTSELO	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0
E91H	3729	MSR_C12_PMON_EVNTSEL1	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1
E92H	3730	MSR_C12_PMON_EVNTSEL2	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2
E93H	3731	MSR_C12_PMON_EVNTSEL3	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3
E94H	3732	MSR_C12_PMON_BOX_FILTER	Package	Uncore C-Box 12 Perfmon Box Wide Filter
E96H	3734	MSR_C12_PMON_CTR0	Package	Uncore C-Box 12 Perfmon Counter 0
E97H	3735	MSR_C12_PMON_CTR1	Package	Uncore C-Box 12 Perfmon Counter 1
E98H	3736	MSR_C12_PMON_CTR2	Package	Uncore C-Box 12 Perfmon Counter 2
E99H	3737	MSR_C12_PMON_CTR3	Package	Uncore C-Box 12 Perfmon Counter 3
E9AH	3738	MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-Box 12 Perfmon Box Wide Filter1
EA4H	3748	MSR_C13_PMON_BOX_CTL	Package	Uncore C-Box 13 Perfmon Local Box Wide Control
EBOH	3760	MSR_C13_PMON_EVNTSELO	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0
EB1H	3761	MSR_C13_PMON_EVNTSEL1	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1
EB2H	3762	MSR_C13_PMON_EVNTSEL2	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EB3H	3763	MSR_C13_PMON_EVNTSEL3	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3
EB4H	3764	MSR_C13_PMON_BOX_FILTER	Package	Uncore C-Box 13 Perfmon Box Wide Filter
EB6H	3766	MSR_C13_PMON_CTR0	Package	Uncore C-Box 13 Perfmon Counter 0
EB7H	3767	MSR_C13_PMON_CTR1	Package	Uncore C-Box 13 Perfmon Counter 1
EB8H	3768	MSR_C13_PMON_CTR2	Package	Uncore C-Box 13 Perfmon Counter 2
EB9H	3769	MSR_C13_PMON_CTR3	Package	Uncore C-Box 13 Perfmon Counter 3
EBAH	3770	MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-Box 13 Perfmon Box Wide Filter1
EC4H	3780	MSR_C14_PMON_BOX_CTL	Package	Uncore C-Box 14 Perfmon Local Box Wide Control
ED0H	3792	MSR_C14_PMON_EVNTSELO	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0
ED1H	3793	MSR_C14_PMON_EVNTSEL1	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1
ED2H	3794	MSR_C14_PMON_EVNTSEL2	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2
ED3H	3795	MSR_C14_PMON_EVNTSEL3	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3
ED4H	3796	MSR_C14_PMON_BOX_FILTER	Package	Uncore C-Box 14 Perfmon Box Wide Filter
ED6H	3798	MSR_C14_PMON_CTR0	Package	Uncore C-Box 14 Perfmon Counter 0
ED7H	3799	MSR_C14_PMON_CTR1	Package	Uncore C-Box 14 Perfmon Counter 1
ED8H	3800	MSR_C14_PMON_CTR2	Package	Uncore C-Box 14 Perfmon Counter 2
ED9H	3801	MSR_C14_PMON_CTR3	Package	Uncore C-Box 14 Perfmon Counter 3
EDAH	3802	MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-Box 14 Perfmon Box Wide Filter1

2.13 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS (BASED ON HASWELL MICROARCHITECTURE)

The 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with CPUID DisplayFamily_DisplayModel signature 06_3CH/06_45H/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 or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3BH	59	IA32_TSC_ADJUST	Thread	Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		31:30		Reserved
		32	Package	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.
		34:33	Package	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved
		39:35		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		55:48	Package	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved
186H	390	IA32_PERFEVTSELO	Thread	Performance Event Select for Counter 0 (R/W) Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
187H	391	IA32_PERFEVTSEL1	Thread	Performance Event Select for Counter 1 (R/W) Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
188H	392	IA32_PERFEVTSEL2	Thread	Performance Event Select for Counter 2 (R/W) Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
		33		IN_TXCP: See Section 19.3.6.5.1. When IN_TXCP=1 & IN_TX=1 and in sampling, a spurious PMI may occur and transactions may continuously abort near overflow conditions. Software should favor using IN_TXCP for counting over sampling. If sampling, software should use large "sample-after" value after clearing the counter configured to use IN_TXCP and also always reset the counter even when no overflow condition was reported.
189H	393	IA32_PERFEVTSEL3	Thread	Performance Event Select for Counter 3 (R/W) Supports all fields described in Table 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W)
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
		9		EN_CALL_STACK
63:9	Reserved			
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		LBR: Last Branch Record
		1		BTF
		5:2		Reserved
		6		TR: Branch Trace
		7		BTS: Log Branch Trace Message to BTS Buffer
		8		BTINT
		9		BTS_OFF_OS
		10		BTS_OFF_USER
		11		FREEZE_LBR_ON_PMI
		12		FREEZE_PERFMON_ON_PMI
		13		ENABLE_UNCORE_PMI
		14		FREEZE_WHILE_SMM
		15		RTM_DEBUG
		63:15		Reserved
491H	1169	IA32_VMX_VMFUNC	Thread	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.
60BH	1548	MSR_PKG_C6_C7_INTERRUPT_RESPONSE_LIMIT_1	Package	Package C6/C7 Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the shorter-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60CH	1548	MSR_PKGC_IRTL2	Package	Package C6/C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the longer-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		Interrupt response time limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.
		12:10		Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.
		14:13		Reserved
		15		Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.
		63:16		Reserved
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 14.10.3, "Package RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O)
		7:0		Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 Ratio and Power Level (R/O)
		14:0		PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.
		15		Reserved
		23:16		Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.
		31:24		Reserved

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		46:32		PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.
		62:47		PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 Ratio and Power Level (R/O)
		14:0		PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.
		62:47		PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W)
		1:0		TDP_LEVEL (RW/L) System BIOS can program this field.
		30:2		Reserved
		31		Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		7:0		MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.
		30:8		Reserved
		31		TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved
C80H	3200	IA32_DEBUG_INTERFACE	Package	Silicon Debug Feature Control (R/W) See Table 2-2.

2.13.1 MSRs in 4th Generation Intel® Core™ Processor Family (based on Haswell Microarchitecture)

Table 2-30 lists model-specific registers (MSRs) that are specific to 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3CH/06_45H/06_46H, see Table 2-1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		3:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s Package C states C7 are not available to processors with a signature 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
		17DH	381	MSR_SMM_MCA_CAP
57:0				Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Core 0 select.
		1		Core 1 select.
		2		Core 2 select.
		3		Core 3 select.
		18:4		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Encoded number of C-Box, derive value by "-1".
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Core 0 select.
		1		Core 1 select.
		2		Core 2 select.
		3		Core 3 select.
		18:4		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.
		0		Lock (SMM-RW) When set to '1' locks this register from further changes.
		1		Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		2		SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL);EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.
		N-1:0		LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is OFFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL);EBX[15:0] can be updated.
		63:N		Reserved
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{ESU}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	PP1 RAPL Power Limit Control (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	PP1 Balance Policy (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved
		4		Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		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		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)
		0		PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved
		4		Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		9		Graphics Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		11		Package-Level PL2 Power Limiting Status (RO) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved
		20		Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		26		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)
		0		PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		5:2		Reserved
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9		Reserved
		10		Package-Level Power Limiting PL1 Status (RO) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		Package-Level PL2 Power Limiting Status (RO) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		Reserved
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		25		Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1824	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
736H	1846	MSR_UNC_CBO_3_PERFCTRO	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29 for other MSR definitions applicable to processors with CPUID signatures 063CH, 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 CPUID DisplayFamily_DisplayModel signature 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 DisplayFamily_DisplayModel Signature 06_45H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		3:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10
		9:4		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/W0)

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily_DisplayModel Signature 06_45H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		63:29		Reserved
630H	1584	MSR_PKG_C8_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C8 Residency Counter (R/O) Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.
		63:60		Reserved
631H	1585	MSR_PKG_C9_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C9 Residency Counter (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.
		63:60		Reserved
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C10 Residency Counter (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.
		63:60		Reserved
See Table 2-20, Table 2-21, Table 2-22, Table 2-29, Table 2-30 for other MSR definitions applicable to processors with CPUID signature 06_45H.				

2.14 MSRS IN INTEL® XEON® PROCESSOR E5 V3 AND E7 V3 PRODUCT FAMILY

Intel® Xeon® processor E5 v3 family and Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID DisplayFamily_DisplayModel = 06_3F). These processors supports the MSR interfaces listed in Table 2-20, Table 2-29, and Table 2-32.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
35H	53	MSR_CORE_THREAD_COUNT	Package	Configured State of Enabled Processor Core Count and Logical Processor Count (R/O) <ul style="list-style-type: none"> After a Power-On RESET, enumerates factory configuration of the number of processor cores and logical processors in the physical package. Following the sequence of (i) BIOS modified a Configuration Mask which selects a subset of processor cores to be active post RESET and (ii) a RESET event after the modification, enumerates the current configuration of enabled processor core count and logical processor count in the physical package.
		15:0		THREAD_COUNT (R/O) The number of logical processors that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		31:16		Core_COUNT (R/O) The number of processor cores that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		63:32		Reserved
53H	83	MSR_THREAD_ID_INFO	Thread	A Hardware Assigned ID for the Logical Processor (R/O)
		7:0		Logical_Processor_ID (R/O) An implementation-specific numerical value physically assigned to each logical processor. This ID is not related to Initial APIC ID or x2APIC ID, it is unique within a physical package.
		63:8		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/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		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		MCG_EM_P
		26		MCG_ELOG_P
		63:27		Reserved
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		39:32	Package	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.
		47:40	Package	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.
		55:48	Package	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:56	Package	Maximum Ratio Limit for 16C Maximum turbo ratio limit of 16 core active.
1AFH	431	MSR_TURBO_RATIO_LIMIT2	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 17C Maximum turbo ratio limit of 17 core active.
		15:8	Package	Maximum Ratio Limit for 18C Maximum turbo ratio limit of 18 core active.
		62:16	Package	Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1 and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy Consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
61EH	1566	MSR_PCIE_PLL_RATIO	Package	Configuration of PCIE PLL Relative to BCLK(R/W)

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1:0	Package	PCIE Ratio (R/W) 00b: Use 5:5 mapping for 100MHz operation (default). 01b: Use 5:4 mapping for 125MHz operation. 10b: Use 5:3 mapping for 166MHz operation. 11b: Use 5:2 mapping for 250MHz operation.
		2	Package	LPLL Select (R/W) If 1, use configured setting of PCIE Ratio.
		3	Package	LONG RESET (R/W) If 1, wait an additional time-out before re-locking Gen2/Gen3 PLLs.
		63:4		Reserved
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (RO) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (RO) When set, frequency is reduced below the operating system request due to a thermal event.
		2		Power Budget Management Status (RO) When set, frequency is reduced below the operating system request due to PBM limit
		3		Platform Configuration Services Status (RO) When set, frequency is reduced below the operating system request due to PCS limit
		4		Reserved

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		9		Reserved
		10		Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.
		12:11		Reserved
		13		Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.
		14		Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.
		15		Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		19		Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28:27		Reserved
		29		Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31		Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:32		Reserved
C8DH	3213	IA32_QM_EVTSEL	THREAD	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (R/W) Event encoding: 0x0: No monitoring. 0x1: L3 occupancy monitoring. All other encoding reserved.
		31:8		Reserved
		41:32		RMID (R/W)
		63:42		Reserved
C8EH	3214	IA32_QM_CTR	THREAD	Monitoring Counter Register (R/O) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		61:0		Resource Monitored Data
		62		Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.
		63		Error: If 1, indicates and unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		63: 10		Reserved
See Table 2-20, Table 2-29 for other MSR definitions applicable to processors with CPUID signature 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

Intel Xeon Processor E5 v3 and E7 v3 family are based on the Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-33. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3FH.

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
700H	1792	MSR_PMON_GLOBAL_CTL	Package	Uncore Perfmon Per-Socket Global Control
701H	1793	MSR_PMON_GLOBAL_STATUS	Package	Uncore Perfmon Per-Socket Global Status
702H	1794	MSR_PMON_GLOBAL_CONFIG	Package	Uncore Perfmon Per-Socket Global Configuration
703H	1795	MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-Box UCLK Fixed Counter Control
704H	1796	MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-Box UCLK Fixed Counter
705H	1797	MSR_U_PMON_EVNTSELO	Package	Uncore U-Box Perfmon Event Select for U-Box Counter 0
706H	1798	MSR_U_PMON_EVNTSEL1	Package	Uncore U-Box Perfmon Event Select for U-Box Counter 1
708H	1800	MSR_U_PMON_BOX_STATUS	Package	Uncore U-Box Perfmon U-Box Wide Status
709H	1801	MSR_U_PMON_CTR0	Package	Uncore U-Box Perfmon Counter 0
70AH	1802	MSR_U_PMON_CTR1	Package	Uncore U-Box Perfmon Counter 1
710H	1808	MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU Perfmon for PCU-Box-Wide Control
711H	1809	MSR_PCU_PMON_EVNTSELO	Package	Uncore PCU Perfmon Event Select for PCU Counter 0
712H	1810	MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU Perfmon Event Select for PCU Counter 1
713H	1811	MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU Perfmon Event Select for PCU Counter 2
714H	1812	MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU Perfmon Event Select for PCU Counter 3
715H	1813	MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU Perfmon Box-Wide Filter
716H	1814	MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU Perfmon Box Wide Status
717H	1815	MSR_PCU_PMON_CTR0	Package	Uncore PCU Perfmon Counter 0
718H	1816	MSR_PCU_PMON_CTR1	Package	Uncore PCU Perfmon Counter 1
719H	1817	MSR_PCU_PMON_CTR2	Package	Uncore PCU Perfmon Counter 2
71AH	1818	MSR_PCU_PMON_CTR3	Package	Uncore PCU Perfmon Counter 3
720H	1824	MSR_S0_PMON_BOX_CTL	Package	Uncore SBo 0 Perfmon for SBo 0 Box-Wide Control
721H	1825	MSR_S0_PMON_EVNTSELO	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 0
722H	1826	MSR_S0_PMON_EVNTSEL1	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 1
723H	1827	MSR_S0_PMON_EVNTSEL2	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 2
724H	1828	MSR_S0_PMON_EVNTSEL3	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 3
725H	1829	MSR_S0_PMON_BOX_FILTER	Package	Uncore SBo 0 Perfmon Box-Wide Filter
726H	1830	MSR_S0_PMON_CTR0	Package	Uncore SBo 0 Perfmon Counter 0
727H	1831	MSR_S0_PMON_CTR1	Package	Uncore SBo 0 Perfmon Counter 1
728H	1832	MSR_S0_PMON_CTR2	Package	Uncore SBo 0 Perfmon Counter 2
729H	1833	MSR_S0_PMON_CTR3	Package	Uncore SBo 0 Perfmon Counter 3
72AH	1834	MSR_S1_PMON_BOX_CTL	Package	Uncore SBo 1 Perfmon for SBo 1 Box-Wide Control

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
72BH	1835	MSR_S1_PMON_EVNTSELO	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 0
72CH	1836	MSR_S1_PMON_EVNTSEL1	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 1
72DH	1837	MSR_S1_PMON_EVNTSEL2	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 2
72EH	1838	MSR_S1_PMON_EVNTSEL3	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 3
72FH	1839	MSR_S1_PMON_BOX_FILTER	Package	Uncore SBo 1 Perfmon Box-Wide Filter
730H	1840	MSR_S1_PMON_CTR0	Package	Uncore SBo 1 Perfmon Counter 0
731H	1841	MSR_S1_PMON_CTR1	Package	Uncore SBo 1 Perfmon Counter 1
732H	1842	MSR_S1_PMON_CTR2	Package	Uncore SBo 1 Perfmon Counter 2
733H	1843	MSR_S1_PMON_CTR3	Package	Uncore SBo 1 Perfmon Counter 3
734H	1844	MSR_S2_PMON_BOX_CTL	Package	Uncore SBo 2 Perfmon for SBo 2 Box-Wide Control
735H	1845	MSR_S2_PMON_EVNTSELO	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 0
736H	1846	MSR_S2_PMON_EVNTSEL1	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 1
737H	1847	MSR_S2_PMON_EVNTSEL2	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 2
738H	1848	MSR_S2_PMON_EVNTSEL3	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 3
739H	1849	MSR_S2_PMON_BOX_FILTER	Package	Uncore SBo 2 Perfmon Box-Wide Filter
73AH	1850	MSR_S2_PMON_CTR0	Package	Uncore SBo 2 Perfmon Counter 0
73BH	1851	MSR_S2_PMON_CTR1	Package	Uncore SBo 2 Perfmon Counter 1
73CH	1852	MSR_S2_PMON_CTR2	Package	Uncore SBo 2 Perfmon Counter 2
73DH	1853	MSR_S2_PMON_CTR3	Package	Uncore SBo 2 Perfmon Counter 3
73EH	1854	MSR_S3_PMON_BOX_CTL	Package	Uncore SBo 3 Perfmon for SBo 3 Box-Wide Control
73FH	1855	MSR_S3_PMON_EVNTSELO	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 0
740H	1856	MSR_S3_PMON_EVNTSEL1	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 1
741H	1857	MSR_S3_PMON_EVNTSEL2	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 2
742H	1858	MSR_S3_PMON_EVNTSEL3	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 3
743H	1859	MSR_S3_PMON_BOX_FILTER	Package	Uncore SBo 3 Perfmon Box-Wide Filter
744H	1860	MSR_S3_PMON_CTR0	Package	Uncore SBo 3 Perfmon Counter 0
745H	1861	MSR_S3_PMON_CTR1	Package	Uncore SBo 3 Perfmon Counter 1
746H	1862	MSR_S3_PMON_CTR2	Package	Uncore SBo 3 Perfmon Counter 2
747H	1863	MSR_S3_PMON_CTR3	Package	Uncore SBo 3 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E00H	3584	MSR_CO_PMON_BOX_CTL	Package	Uncore C-Box 0 Perfmon for Box-Wide Control
E01H	3585	MSR_CO_PMON_EVNTSELO	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 0
E02H	3586	MSR_CO_PMON_EVNTSEL1	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 1
E03H	3587	MSR_CO_PMON_EVNTSEL2	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 2
E04H	3588	MSR_CO_PMON_EVNTSEL3	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 3
E05H	3589	MSR_CO_PMON_BOX_FILTER0	Package	Uncore C-Box 0 Perfmon Box Wide Filter 0
E06H	3590	MSR_CO_PMON_BOX_FILTER1	Package	Uncore C-Box 0 Perfmon Box Wide Filter 1
E07H	3591	MSR_CO_PMON_BOX_STATUS	Package	Uncore C-Box 0 Perfmon Box Wide Status
E08H	3592	MSR_CO_PMON_CTR0	Package	Uncore C-Box 0 Perfmon Counter 0
E09H	3593	MSR_CO_PMON_CTR1	Package	Uncore C-Box 0 Perfmon Counter 1
E0AH	3594	MSR_CO_PMON_CTR2	Package	Uncore C-Box 0 Perfmon Counter 2
E0BH	3595	MSR_CO_PMON_CTR3	Package	Uncore C-Box 0 Perfmon Counter 3
E10H	3600	MSR_C1_PMON_BOX_CTL	Package	Uncore C-Box 1 Perfmon for Box-Wide Control
E11H	3601	MSR_C1_PMON_EVNTSELO	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 0
E12H	3602	MSR_C1_PMON_EVNTSEL1	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 1
E13H	3603	MSR_C1_PMON_EVNTSEL2	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 2
E14H	3604	MSR_C1_PMON_EVNTSEL3	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 3
E15H	3605	MSR_C1_PMON_BOX_FILTER0	Package	Uncore C-Box 1 Perfmon Box Wide Filter 0
E16H	3606	MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-Box 1 Perfmon Box Wide Filter 1
E17H	3607	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-Box 1 Perfmon Box Wide Status
E18H	3608	MSR_C1_PMON_CTR0	Package	Uncore C-Box 1 Perfmon Counter 0
E19H	3609	MSR_C1_PMON_CTR1	Package	Uncore C-Box 1 Perfmon Counter 1
E1AH	3610	MSR_C1_PMON_CTR2	Package	Uncore C-Box 1 Perfmon Counter 2
E1BH	3611	MSR_C1_PMON_CTR3	Package	Uncore C-Box 1 Perfmon Counter 3
E20H	3616	MSR_C2_PMON_BOX_CTL	Package	Uncore C-Box 2 Perfmon for Box-Wide Control
E21H	3617	MSR_C2_PMON_EVNTSELO	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 0
E22H	3618	MSR_C2_PMON_EVNTSEL1	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 1
E23H	3619	MSR_C2_PMON_EVNTSEL2	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 2
E24H	3620	MSR_C2_PMON_EVNTSEL3	Package	Uncore C-Box 2 Perfmon Event select for C-Box 2 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E25H	3621	MSR_C2_PMON_BOX_FILTER0	Package	Uncore C-Box 2 Perfmon Box Wide Filter 0
E26H	3622	MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-Box 2 Perfmon Box Wide Filter 1
E27H	3623	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-Box 2 Perfmon Box Wide Status
E28H	3624	MSR_C2_PMON_CTR0	Package	Uncore C-Box 2 Perfmon Counter 0
E29H	3625	MSR_C2_PMON_CTR1	Package	Uncore C-Box 2 Perfmon Counter 1
E2AH	3626	MSR_C2_PMON_CTR2	Package	Uncore C-Box 2 Perfmon Counter 2
E2BH	3627	MSR_C2_PMON_CTR3	Package	Uncore C-Box 2 Perfmon Counter 3
E30H	3632	MSR_C3_PMON_BOX_CTL	Package	Uncore C-Box 3 Perfmon for Box-Wide Control
E31H	3633	MSR_C3_PMON_EVNTSELO	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 0
E32H	3634	MSR_C3_PMON_EVNTSEL1	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 1
E33H	3635	MSR_C3_PMON_EVNTSEL2	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 2
E34H	3636	MSR_C3_PMON_EVNTSEL3	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 3
E35H	3637	MSR_C3_PMON_BOX_FILTER0	Package	Uncore C-Box 3 Perfmon Box Wide Filter 0
E36H	3638	MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-Box 3 Perfmon Box Wide Filter 1
E37H	3639	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-Box 3 Perfmon Box Wide Status
E38H	3640	MSR_C3_PMON_CTR0	Package	Uncore C-Box 3 Perfmon Counter 0
E39H	3641	MSR_C3_PMON_CTR1	Package	Uncore C-Box 3 Perfmon Counter 1
E3AH	3642	MSR_C3_PMON_CTR2	Package	Uncore C-Box 3 Perfmon Counter 2
E3BH	3643	MSR_C3_PMON_CTR3	Package	Uncore C-Box 3 Perfmon Counter 3
E40H	3648	MSR_C4_PMON_BOX_CTL	Package	Uncore C-Box 4 Perfmon for Box-Wide Control
E41H	3649	MSR_C4_PMON_EVNTSELO	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 0
E42H	3650	MSR_C4_PMON_EVNTSEL1	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 1
E43H	3651	MSR_C4_PMON_EVNTSEL2	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 2
E44H	3652	MSR_C4_PMON_EVNTSEL3	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 3
E45H	3653	MSR_C4_PMON_BOX_FILTER0	Package	Uncore C-Box 4 Perfmon Box Wide Filter 0
E46H	3654	MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-Box 4 Perfmon Box Wide Filter 1
E47H	3655	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-Box 4 Perfmon Box Wide Status
E48H	3656	MSR_C4_PMON_CTR0	Package	Uncore C-Box 4 Perfmon Counter 0
E49H	3657	MSR_C4_PMON_CTR1	Package	Uncore C-Box 4 Perfmon Counter 1
E4AH	3658	MSR_C4_PMON_CTR2	Package	Uncore C-Box 4 Perfmon Counter 2
E4BH	3659	MSR_C4_PMON_CTR3	Package	Uncore C-Box 4 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E50H	3664	MSR_C5_PMON_BOX_CTL	Package	Uncore C-Box 5 Perfmon for Box-Wide Control
E51H	3665	MSR_C5_PMON_EVNTSELO	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 0
E52H	3666	MSR_C5_PMON_EVNTSEL1	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 1
E53H	3667	MSR_C5_PMON_EVNTSEL2	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 2
E54H	3668	MSR_C5_PMON_EVNTSEL3	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 3
E55H	3669	MSR_C5_PMON_BOX_FILTER0	Package	Uncore C-Box 5 Perfmon Box Wide Filter 0
E56H	3670	MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-Box 5 Perfmon Box Wide Filter 1
E57H	3671	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-Box 5 Perfmon Box Wide Status
E58H	3672	MSR_C5_PMON_CTR0	Package	Uncore C-Box 5 Perfmon Counter 0
E59H	3673	MSR_C5_PMON_CTR1	Package	Uncore C-Box 5 Perfmon Counter 1
E5AH	3674	MSR_C5_PMON_CTR2	Package	Uncore C-Box 5 Perfmon Counter 2
E5BH	3675	MSR_C5_PMON_CTR3	Package	Uncore C-Box 5 Perfmon Counter 3
E60H	3680	MSR_C6_PMON_BOX_CTL	Package	Uncore C-Box 6 Perfmon for Box-Wide Control
E61H	3681	MSR_C6_PMON_EVNTSELO	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 0
E62H	3682	MSR_C6_PMON_EVNTSEL1	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 1
E63H	3683	MSR_C6_PMON_EVNTSEL2	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 2
E64H	3684	MSR_C6_PMON_EVNTSEL3	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 3
E65H	3685	MSR_C6_PMON_BOX_FILTER0	Package	Uncore C-Box 6 Perfmon Box Wide Filter 0
E66H	3686	MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-Box 6 Perfmon Box Wide Filter 1
E67H	3687	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-Box 6 Perfmon Box Wide Status
E68H	3688	MSR_C6_PMON_CTR0	Package	Uncore C-Box 6 Perfmon Counter 0
E69H	3689	MSR_C6_PMON_CTR1	Package	Uncore C-Box 6 Perfmon Counter 1
E6AH	3690	MSR_C6_PMON_CTR2	Package	Uncore C-Box 6 Perfmon Counter 2
E6BH	3691	MSR_C6_PMON_CTR3	Package	Uncore C-Box 6 Perfmon Counter 3
E70H	3696	MSR_C7_PMON_BOX_CTL	Package	Uncore C-Box 7 Perfmon for Box-Wide Control
E71H	3697	MSR_C7_PMON_EVNTSELO	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 0
E72H	3698	MSR_C7_PMON_EVNTSEL1	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 1
E73H	3699	MSR_C7_PMON_EVNTSEL2	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 2
E74H	3700	MSR_C7_PMON_EVNTSEL3	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E75H	3701	MSR_C7_PMON_BOX_FILTER0	Package	Uncore C-Box 7 Perfmon Box Wide Filter 0
E76H	3702	MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-Box 7 Perfmon Box Wide Filter 1
E77H	3703	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-Box 7 Perfmon Box Wide Status
E78H	3704	MSR_C7_PMON_CTR0	Package	Uncore C-Box 7 Perfmon Counter 0
E79H	3705	MSR_C7_PMON_CTR1	Package	Uncore C-Box 7 Perfmon Counter 1
E7AH	3706	MSR_C7_PMON_CTR2	Package	Uncore C-Box 7 Perfmon Counter 2
E7BH	3707	MSR_C7_PMON_CTR3	Package	Uncore C-Box 7 Perfmon Counter 3
E80H	3712	MSR_C8_PMON_BOX_CTL	Package	Uncore C-Box 8 Perfmon Local Box Wide Control
E81H	3713	MSR_C8_PMON_EVNTSELO	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0
E82H	3714	MSR_C8_PMON_EVNTSEL1	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1
E83H	3715	MSR_C8_PMON_EVNTSEL2	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2
E84H	3716	MSR_C8_PMON_EVNTSEL3	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3
E85H	3717	MSR_C8_PMON_BOX_FILTER0	Package	Uncore C-Box 8 Perfmon Box Wide Filter 0
E86H	3718	MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-Box 8 Perfmon Box Wide Filter 1
E87H	3719	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-Box 8 Perfmon Box Wide Status
E88H	3720	MSR_C8_PMON_CTR0	Package	Uncore C-Box 8 Perfmon Counter 0
E89H	3721	MSR_C8_PMON_CTR1	Package	Uncore C-Box 8 Perfmon Counter 1
E8AH	3722	MSR_C8_PMON_CTR2	Package	Uncore C-Box 8 Perfmon Counter 2
E8BH	3723	MSR_C8_PMON_CTR3	Package	Uncore C-Box 8 Perfmon Counter 3
E90H	3728	MSR_C9_PMON_BOX_CTL	Package	Uncore C-Box 9 Perfmon Local Box Wide Control
E91H	3729	MSR_C9_PMON_EVNTSELO	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 0
E92H	3730	MSR_C9_PMON_EVNTSEL1	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 1
E93H	3731	MSR_C9_PMON_EVNTSEL2	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 2
E94H	3732	MSR_C9_PMON_EVNTSEL3	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 3
E95H	3733	MSR_C9_PMON_BOX_FILTER0	Package	Uncore C-Box 9 Perfmon Box Wide Filter 0
E96H	3734	MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-Box 9 Perfmon Box Wide Filter 1
E97H	3735	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-Box 9 Perfmon Box Wide Status
E98H	3736	MSR_C9_PMON_CTR0	Package	Uncore C-Box 9 Perfmon Counter 0
E99H	3737	MSR_C9_PMON_CTR1	Package	Uncore C-Box 9 Perfmon Counter 1
E9AH	3738	MSR_C9_PMON_CTR2	Package	Uncore C-Box 9 Perfmon Counter 2
E9BH	3739	MSR_C9_PMON_CTR3	Package	Uncore C-Box 9 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EA0H	3744	MSR_C10_PMON_BOX_CTL	Package	Uncore C-Box 10 Perfmon Local Box Wide Control
EA1H	3745	MSR_C10_PMON_EVNTSELO	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0
EA2H	3746	MSR_C10_PMON_EVNTSEL1	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1
EA3H	3747	MSR_C10_PMON_EVNTSEL2	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2
EA4H	3748	MSR_C10_PMON_EVNTSEL3	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3
EA5H	3749	MSR_C10_PMON_BOX_FILTER0	Package	Uncore C-Box 10 Perfmon Box Wide Filter 0
EA6H	3750	MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-Box 10 Perfmon Box Wide Filter 1
EA7H	3751	MSR_C10_PMON_BOX_STATUS	Package	Uncore C-Box 10 Perfmon Box Wide Status
EA8H	3752	MSR_C10_PMON_CTR0	Package	Uncore C-Box 10 Perfmon Counter 0
EA9H	3753	MSR_C10_PMON_CTR1	Package	Uncore C-Box 10 perfmon Counter 1
EAAH	3754	MSR_C10_PMON_CTR2	Package	Uncore C-Box 10 Perfmon Counter 2
EABH	3755	MSR_C10_PMON_CTR3	Package	Uncore C-Box 10 Perfmon Counter 3
EBOH	3760	MSR_C11_PMON_BOX_CTL	Package	Uncore C-Box 11 Perfmon Local Box Wide Control
EB1H	3761	MSR_C11_PMON_EVNTSELO	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0
EB2H	3762	MSR_C11_PMON_EVNTSEL1	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1
EB3H	3763	MSR_C11_PMON_EVNTSEL2	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2
EB4H	3764	MSR_C11_PMON_EVNTSEL3	Package	Uncore C-box 11 Perfmon Event Select for C-Box 11 Counter 3
EB5H	3765	MSR_C11_PMON_BOX_FILTER0	Package	Uncore C-Box 11 Perfmon Box Wide Filter 0
EB6H	3766	MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-Box 11 Perfmon Box Wide Filter 1
EB7H	3767	MSR_C11_PMON_BOX_STATUS	Package	Uncore C-Box 11 Perfmon Box Wide Status
EB8H	3768	MSR_C11_PMON_CTR0	Package	Uncore C-Box 11 Perfmon Counter 0
EB9H	3769	MSR_C11_PMON_CTR1	Package	Uncore C-Box 11 Perfmon Counter 1
EBAH	3770	MSR_C11_PMON_CTR2	Package	Uncore C-Box 11 Perfmon Counter 2
EBBH	3771	MSR_C11_PMON_CTR3	Package	Uncore C-Box 11 Perfmon Counter 3
EC0H	3776	MSR_C12_PMON_BOX_CTL	Package	Uncore C-Box 12 Perfmon Local Box Wide Control
EC1H	3777	MSR_C12_PMON_EVNTSELO	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0
EC2H	3778	MSR_C12_PMON_EVNTSEL1	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1
EC3H	3779	MSR_C12_PMON_EVNTSEL2	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2
EC4H	3780	MSR_C12_PMON_EVNTSEL3	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EC5H	3781	MSR_C12_PMON_BOX_FILTER0	Package	Uncore C-Box 12 Perfmon Box Wide Filter 0
EC6H	3782	MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-Box 12 Perfmon Box Wide Filter 1
EC7H	3783	MSR_C12_PMON_BOX_STATUS	Package	Uncore C-Box 12 Perfmon Box Wide Status
EC8H	3784	MSR_C12_PMON_CTR0	Package	Uncore C-Box 12 Perfmon Counter 0
EC9H	3785	MSR_C12_PMON_CTR1	Package	Uncore C-Box 12 Perfmon Counter 1
ECAH	3786	MSR_C12_PMON_CTR2	Package	Uncore C-Box 12 Perfmon Counter 2
ECBH	3787	MSR_C12_PMON_CTR3	Package	Uncore C-Box 12 Perfmon Counter 3
ED0H	3792	MSR_C13_PMON_BOX_CTL	Package	Uncore C-Box 13 Perfmon local box wide control.
ED1H	3793	MSR_C13_PMON_EVNTSELO	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0
ED2H	3794	MSR_C13_PMON_EVNTSEL1	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1
ED3H	3795	MSR_C13_PMON_EVNTSEL2	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2
ED4H	3796	MSR_C13_PMON_EVNTSEL3	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3
ED5H	3797	MSR_C13_PMON_BOX_FILTER0	Package	Uncore C-Box 13 Perfmon Box Wide Filter 0
ED6H	3798	MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-Box 13 Perfmon Box Wide Filter 1
ED7H	3799	MSR_C13_PMON_BOX_STATUS	Package	Uncore C-Box 13 Perfmon Box Wide Status
ED8H	3800	MSR_C13_PMON_CTR0	Package	Uncore C-Box 13 Perfmon Counter 0
ED9H	3801	MSR_C13_PMON_CTR1	Package	Uncore C-Box 13 Perfmon Counter 1
EDAH	3802	MSR_C13_PMON_CTR2	Package	Uncore C-Box 13 Perfmon Counter 2
EDBH	3803	MSR_C13_PMON_CTR3	Package	Uncore C-Box 13 Perfmon Counter 3
EE0H	3808	MSR_C14_PMON_BOX_CTL	Package	Uncore C-Box 14 Perfmon Local Box Wide Control
EE1H	3809	MSR_C14_PMON_EVNTSELO	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0
EE2H	3810	MSR_C14_PMON_EVNTSEL1	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1
EE3H	3811	MSR_C14_PMON_EVNTSEL2	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2
EE4H	3812	MSR_C14_PMON_EVNTSEL3	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3
EE5H	3813	MSR_C14_PMON_BOX_FILTER	Package	Uncore C-Box 14 Perfmon Box Wide Filter 0
EE6H	3814	MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-Box 14 Perfmon Box Wide Filter 1
EE7H	3815	MSR_C14_PMON_BOX_STATUS	Package	Uncore C-Box 14 Perfmon Box Wide Status
EE8H	3816	MSR_C14_PMON_CTR0	Package	Uncore C-Box 14 Perfmon Counter 0
EE9H	3817	MSR_C14_PMON_CTR1	Package	Uncore C-Box 14 Perfmon Counter 1
EEAH	3818	MSR_C14_PMON_CTR2	Package	Uncore C-Box 14 Perfmon Counter 2
EEBH	3819	MSR_C14_PMON_CTR3	Package	Uncore C-Box 14 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EF0H	3824	MSR_C15_PMON_BOX_CTL	Package	Uncore C-Box 15 Perfmon Local Box Wide Control
EF1H	3825	MSR_C15_PMON_EVNTSELO	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 0
EF2H	3826	MSR_C15_PMON_EVNTSEL1	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 1
EF3H	3827	MSR_C15_PMON_EVNTSEL2	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 2
EF4H	3828	MSR_C15_PMON_EVNTSEL3	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 3
EF5H	3829	MSR_C15_PMON_BOX_FILTER0	Package	Uncore C-Box 15 Perfmon Box Wide Filter 0
EF6H	3830	MSR_C15_PMON_BOX_FILTER1	Package	Uncore C-Box 15 Perfmon Box Wide Filter 1
EF7H	3831	MSR_C15_PMON_BOX_STATUS	Package	Uncore C-Box 15 Perfmon Box Wide Status
EF8H	3832	MSR_C15_PMON_CTR0	Package	Uncore C-Box 15 Perfmon Counter 0
EF9H	3833	MSR_C15_PMON_CTR1	Package	Uncore C-Box 15 Perfmon Counter 1
EFAH	3834	MSR_C15_PMON_CTR2	Package	Uncore C-Box 15 Perfmon Counter 2
EFBH	3835	MSR_C15_PMON_CTR3	Package	Uncore C-Box 15 Perfmon Counter 3
F00H	3840	MSR_C16_PMON_BOX_CTL	Package	Uncore C-Box 16 Perfmon for Box-Wide Control
F01H	3841	MSR_C16_PMON_EVNTSELO	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 0
F02H	3842	MSR_C16_PMON_EVNTSEL1	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 1
F03H	3843	MSR_C16_PMON_EVNTSEL2	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 2
F04H	3844	MSR_C16_PMON_EVNTSEL3	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 3
F05H	3845	MSR_C16_PMON_BOX_FILTER0	Package	Uncore C-Box 16 Perfmon Box Wide Filter 0
F06H	3846	MSR_C16_PMON_BOX_FILTER1	Package	Uncore C-Box 16 Perfmon Box Wide Filter 1
F07H	3847	MSR_C16_PMON_BOX_STATUS	Package	Uncore C-Box 16 Perfmon Box Wide Status
F08H	3848	MSR_C16_PMON_CTR0	Package	Uncore C-Box 16 Perfmon Counter 0
F09H	3849	MSR_C16_PMON_CTR1	Package	Uncore C-Box 16 Perfmon Counter 1
F0AH	3850	MSR_C16_PMON_CTR2	Package	Uncore C-Box 16 Perfmon Counter 2
FOBH	3851	MSR_C16_PMON_CTR3	Package	Uncore C-Box 16 Perfmon Counter 3
F10H	3856	MSR_C17_PMON_BOX_CTL	Package	Uncore C-Box 17 Perfmon for Box-Wide Control
F11H	3857	MSR_C17_PMON_EVNTSELO	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 0
F12H	3858	MSR_C17_PMON_EVNTSEL1	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 1
F13H	3859	MSR_C17_PMON_EVNTSEL2	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 2
F14H	3860	MSR_C17_PMON_EVNTSEL3	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
F15H	3861	MSR_C17_PMON_BOX_FILTER0	Package	Uncore C-Box 17 Perfmon Box Wide Filter 0
F16H	3862	MSR_C17_PMON_BOX_FILTER1	Package	Uncore C-Box 17 Perfmon Box Wide Filter 1
F17H	3863	MSR_C17_PMON_BOX_STATUS	Package	Uncore C-Box 17 Perfmon Box Wide Status
F18H	3864	MSR_C17_PMON_CTR0	Package	Uncore C-Box 17 Perfmon Counter 0
F19H	3865	MSR_C17_PMON_CTR1	Package	Uncore C-Box 17 Perfmon Counter 1
F1AH	3866	MSR_C17_PMON_CTR2	Package	Uncore C-Box 17 Perfmon Counter 2
F1BH	3867	MSR_C17_PMON_CTR3	Package	Uncore C-Box 17 Perfmon Counter 3

2.15 MSRS IN INTEL® CORE™ M PROCESSORS AND 5TH GENERATION INTEL CORE PROCESSORS

The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors, and Intel® Xeon® Processor E3-1200 v4 family are based on the Broadwell microarchitecture. The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors have CPUID DisplayFamily_DisplayModel signature 06_3DH. Intel® Xeon® Processor E3-1200 v4 family and the 5th generation Intel® Core™ Processors have CPUID DisplayFamily_DisplayModel signature 06_47H. Processors with signatures 06_3DH and 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 signatures 06_3DH, 06_47H, 06_4FH, and 06_56H).

Table 2-34. Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0		Ovf_PMC0
		1		Ovf_PMC1
		2		Ovf_PMC2
		3		Ovf_PMC3
		31:4		Reserved
		32		Ovf_FixedCtr0
		33		Ovf_FixedCtr1
		34		Ovf_FixedCtr2
		54:35		Reserved
		55		Trace_ToPA_PMI See Section 32.2.7.2, "Table of Physical Addresses (ToPA)."
		60:56		Reserved

Table 2-34. Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		61		Ovf_Uncore
		62		Ovf_BufDSSAVE
		63		CondChgd
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0		Set 1 to clear Ovf_PMC0
		1		Set 1 to clear Ovf_PMC1
		2		Set 1 to clear Ovf_PMC2
		3		Set 1 to clear Ovf_PMC3
		31:4		Reserved
		32		Set 1 to clear Ovf_FixedCtr0
		33		Set 1 to clear Ovf_FixedCtr1
		34		Set 1 to clear Ovf_FixedCtr2
		54:35		Reserved.
		55		Set 1 to clear Trace_ToPA_PMI. See Section 32.2.7.2, "Table of Physical Addresses (ToPA)."
		60:56		Reserved
		61		Set 1 to clear Ovf_Uncore
		62		Set 1 to clear Ovf_BufDSSAVE
		63		Set 1 to clear CondChgd
560H	1376	IA32_RTIT_OUTPUT_BASE	THREAD	Trace Output Base Register (R/W)
		6:0		Reserved
		MAXPHYADDR ¹ -1:7		Base physical address.
		63:MAXPHYADDR		Reserved
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	THREAD	Trace Output Mask Pointers Register (R/W)
		6:0		Reserved
		31:7		MaskOrTableOffset
		63:32		Output Offset.
570H	1392	IA32_RTIT_CTL	Thread	Trace Control Register (R/W)
		0		TraceEn
		1		Reserved, must be zero.
		2		OS
		3		User
		6:4		Reserved, must be zero.
		7		CR3 filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.

Table 2-34. Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		9		Reserved, must be zero.
		10		TSCEn
		11		DisRETC
		12		Reserved, must be zero.
		13		Reserved; writing 0 will #GP if also setting TraceEn.
		63:14		Reserved, must be zero.
571H	1393	IA32_RTIT_STATUS	Thread	Tracing Status Register (R/W)
		0		Reserved, writes ignored.
		1		ContexEn, writes ignored.
		2		TriggerEn, writes ignored.
		3		Reserved
		4		Error (R/W)
		5		Stopped
		63:6		Reserved, must be zero.
572H	1394	IA32_RTIT_CR3_MATCH	THREAD	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match.
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.

NOTES:

1. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

Table 2-35 lists MSRs that are specific to Intel Core M processors and 5th Generation Intel Core Processors.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		3:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10
		9:4		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/WO)
		24:16		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Enable Package C-State Auto-Demotion (R/W)
		30		Enable Package C-State Undemotion (R/W)
		63:31		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6core active.
		63:48		Reserved
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 14.10.4, "PPO/PP1 RAPL Domains."

See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, Table 2-34 for other MSR definitions applicable to processors with CPUID signature 06_3DH.

2.16 MSRS IN INTEL® XEON® PROCESSORS E5 V4 FAMILY

The MSRs listed in Table 2-36 are available and common to Intel® Xeon® Processor D product Family (CPUID DisplayFamily_DisplayModel = 06_56H) and to Intel Xeon processors E5 v4, E7 v4 families (CPUID DisplayFamily_DisplayModel = 06_4FH). They are based on the Broadwell microarchitecture.

See Section 2.16.1 for lists of tables of MSRs that are supported by Intel® Xeon® Processor D Family.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/W0) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) See Table 2-26.
		22:16		Reserved.
		23	Package	PPIN_CAP (R/O) See Table 2-26.
		27:24		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.
		30	Package	Programmable TJ OFFSET (R/O) See Table 2-26.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) See Table 2-26.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/WO)

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		16		Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).
		24:17		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCI_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		MCG_EM_P
		26		MCG_ELOG_P
		63:27		Reserved
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal Threshold #1 Status (R/O) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.
		10		Power Limitation Status (R/O) See Table 2-2.
		11		Power Limitation Log (R/WCO) See Table 2-2.
		12		Current Limit Status (R/O) See Table 2-2.
		13		Current Limit Log (R/WCO) See Table 2-2.
		14		Cross Domain Limit Status (R/O) See Table 2-2.
		15		Cross Domain Limit Log (R/WCO) See Table 2-2.
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31		Reading Valid (R/O) See Table 2-2.
		63:32		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R/O) See Table 2-26.
		27:24		TCC Activation Offset (R/W) See Table 2-26.
		63:28		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C
		15:8	Package	Maximum Ratio Limit for 2C
		23:16	Package	Maximum Ratio Limit for 3C
		31:24	Package	Maximum Ratio Limit for 4C
		39:32	Package	Maximum Ratio Limit for 5C
		47:40	Package	Maximum Ratio Limit for 6C
		55:48	Package	Maximum Ratio Limit for 7C
63:56	Package	Maximum Ratio Limit for 8C		
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C
		15:8	Package	Maximum Ratio Limit for 10C
		23:16	Package	Maximum Ratio Limit for 11C
		31:24	Package	Maximum Ratio Limit for 12C
		39:32	Package	Maximum Ratio Limit for 13C
		47:40	Package	Maximum Ratio Limit for 14C
		55:48	Package	Maximum Ratio Limit for 15C
63:56	Package	Maximum Ratio Limit for 16C		
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.
		2		Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit.
		3		Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit.
		4		Reserved
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved
		8		Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).
		9		Reserved
		10		Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.
		12:11		Reserved
		13		Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.
		14		Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.
		15		Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved
		21		Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved
		24		Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		28:27		Reserved
		29		Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		31		Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:32		Reserved
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, "Enabling HWP".
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, "Managing HWP".
		7:0		Minimum Performance (R/W)
		15:8		Maximum Performance (R/W)
		23:16		Desired Performance (R/W)
		63:24		Reserved
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, "HWP Feedback".
		1:0		Reserved
		2		Excursion to Minimum (R/O)
		63:3		Reserved
C8DH	3213	IA32_QM_EVTSEL	Thread	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31:8		Reserved
		41:32		RMID (R/W)
		63:42		Reserved
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		31:10		Reserved
		51:32		COS (R/W)
		63: 52		Reserved
C90H	3216	IA32_L3_QOS_MASK_0	Package	L3 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:19		CBM: Bit vector of available L3 ways for COS 0 enforcement.
		63:20		Reserved
C91H	3217	IA32_L3_QOS_MASK_1	Package	L3 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:19		CBM: Bit vector of available L3 ways for COS 1 enforcement.
		63:20		Reserved
C92H	3218	IA32_L3_QOS_MASK_2	Package	L3 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:19		CBM: Bit vector of available L3 ways for COS 2 enforcement.
		63:20		Reserved
C93H	3219	IA32_L3_QOS_MASK_3	Package	L3 Class Of Service Mask - COS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L3 ways for COS 3 enforcement.
		63:20		Reserved
C94H	3220	IA32_L3_QOS_MASK_4	Package	L3 Class Of Service Mask - COS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.
		0:19		CBM: Bit vector of available L3 ways for COS 4 enforcement.
		63:20		Reserved
C95H	3221	IA32_L3_QOS_MASK_5	Package	L3 Class Of Service Mask - COS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.
		0:19		CBM: Bit vector of available L3 ways for COS 5 enforcement.
		63:20		Reserved

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C96H	3222	IA32_L3_QOS_MASK_6	Package	L3 Class Of Service Mask - COS 6 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=6.
		0:19		CBM: Bit vector of available L3 ways for COS 6 enforcement.
		63:20		Reserved
C97H	3223	IA32_L3_QOS_MASK_7	Package	L3 Class Of Service Mask - COS 7 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=7.
		0:19		CBM: Bit vector of available L3 ways for COS 7 enforcement.
		63:20		Reserved
C98H	3224	IA32_L3_QOS_MASK_8	Package	L3 Class Of Service Mask - COS 8 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=8.
		0:19		CBM: Bit vector of available L3 ways for COS 8 enforcement.
		63:20		Reserved
C99H	3225	IA32_L3_QOS_MASK_9	Package	L3 Class Of Service Mask - COS 9 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=9.
		0:19		CBM: Bit vector of available L3 ways for COS 9 enforcement.
		63:20		Reserved
C9AH	3226	IA32_L3_QOS_MASK_10	Package	L3 Class Of Service Mask - COS 10 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=10.
		0:19		CBM: Bit vector of available L3 ways for COS 10 enforcement.
		63:20		Reserved
C9BH	3227	IA32_L3_QOS_MASK_11	Package	L3 Class Of Service Mask - COS 11 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=11.
		0:19		CBM: Bit vector of available L3 ways for COS 11 enforcement.
		63:20		Reserved
C9CH	3228	IA32_L3_QOS_MASK_12	Package	L3 Class Of Service Mask - COS 12 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=12.
		0:19		CBM: Bit vector of available L3 ways for COS 12 enforcement.
		63:20		Reserved
C9DH	3229	IA32_L3_QOS_MASK_13	Package	L3 Class Of Service Mask - COS 13 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=13.
		0:19		CBM: Bit vector of available L3 ways for COS 13 enforcement.
		63:20		Reserved

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C9EH	3230	IA32_L3_QOS_MASK_14	Package	L3 Class Of Service Mask - COS 14 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14.
		0:19		CBM: Bit vector of available L3 ways for COS 14 enforcement.
		63:20		Reserved
C9FH	3231	IA32_L3_QOS_MASK_15	Package	L3 Class Of Service Mask - COS 15 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15.
		0:19		CBM: Bit vector of available L3 ways for COS 15 enforcement.
		63:20		Reserved

2.16.1 Additional MSRs Supported in the Intel® Xeon® Processor D Product Family

The MSRs listed in Table 2-37 are available to Intel® Xeon® Processor D Product Family (CPUID DisplayFamily_DisplayModel = 06_56H). The Intel® Xeon® processor D product family is based on the 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 DisplayFamily_DisplayModel 06_56H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		62:0	Package	Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with DisplayFamily_DisplayModel 06_56H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
See Table 2-20, Table 2-29, Table 2-34, and Table 2-36 for other MSR definitions applicable to processors with CPUID signature 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 Intel® Xeon® Processor E5 v4 and E7 v4 Families (CPUID DisplayFamily_DisplayModel = 06_4FH). The Intel® Xeon® processor E5 v4 family is based on the Broadwell micro-

architecture 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 DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.
		62:0	Package	Reserved
		63	Package	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1 and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
C81H	3201	IA32_L3_QOS_CFG	Package	Cache Allocation Technology Configuration (R/W)
		0		CAT Enable. Set 1 to enable Cache Allocation Technology.
		63:1		Reserved

See Table 2-20, Table 2-21, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with CPUID signature 06_45H.

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.17 MSRS IN THE 6TH GENERATION, 7TH GENERATION, 8TH GENERATION, 9TH GENERATION, 10TH GENERATION, 11TH GENERATION, AND 12TH GENERATION INTEL® CORE™ PROCESSORS, INTEL® XEON® PROCESSOR SCALABLE FAMILY, 2ND AND 3RD GENERATION INTEL® XEON® PROCESSOR SCALABLE FAMILY, 8TH GENERATION INTEL® CORE™ I3 PROCESSORS, AND INTEL® XEON® E PROCESSORS

6th generation Intel® Core™ processors are based on the Skylake microarchitecture and have CPUID DisplayFamily_DisplayModel signatures of 06_4EH and 06_5EH.

The Intel® Xeon® Processor Scalable Family based on the Skylake microarchitecture, the 2nd generation Intel® Xeon® Processor Scalable Family based on the Cascade Lake product, and the 3rd generation Intel® Xeon® Processor Scalable Family based on the Cooper Lake product all have a CPUID DisplayFamily_DisplayModel signature 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 the Coffee Lake microarchitecture; these processors have CPUID DisplayFamily_DisplayModel signatures of 06_8EH and 06_9EH.

8th generation Intel® Core™ i3 processors are based on Cannon Lake microarchitecture and have a CPUID DisplayFamily_DisplayModel signature of 06_66H.

10th generation Intel® Core™ processors are based on Comet Lake microarchitecture (with CPUID DisplayFamily_DisplayModel signatures of 06_A5H, 06_A6H) and Ice Lake microarchitecture (with CPUID DisplayFamily_DisplayModel signatures of 06_7DH and 06_7EH).

11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture and have CPUID DisplayFamily_DisplayModel signatures of 06_8CH and 06_8DH.

The 3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture have CPUID DisplayFamily_DisplayModel signatures of 06_6AH and 06_6CH.

The 12th generation Intel® Core™ processors supporting the Alder Lake performance hybrid architecture have CPUID DisplayFamily_DisplayModel signatures of 06_97H, 06_9AH, and 06_BFH.

These processors support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-25, Table 2-29, Table 2-35, and Table 2-39¹. For an MSR listed in Table 2-39 that also appears in the model-specific tables of prior generations, Table 2-39 supersedes 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.

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 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	MTRR Capability (R/O, Architectural) See Table 2-2
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal threshold #1 Status (R/O) See Table 2-2.
		7		Thermal threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.
		10		Power Limitation Status (R/O) See Table 2-2.
		11		Power Limitation Log (R/WCO) See Table 2-2.
		12		Current Limit Status (R/O) See Table 2-2.
		13		Current Limit Log (R/WCO) See Table 2-2.
14		Cross Domain Limit Status (R/O) See Table 2-2.		

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15		Cross Domain Limit Log (R/WC0) See Table 2-2.
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.
		31		Reading Valid (R/O) See Table 2-2.
		63:32		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, R/W if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register See http://biosbits.org .
		0		Reserved
		1	Package	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		18:2		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		19		Disable Energy Efficiency Optimization (R/W) Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.
		20		Disable Race to Halt Optimization (R/W) Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.
		63:21		Reserved
300H	768	MSR_SGXOWNEREPOCH0	Package	Lower 64 Bit CR_SGXOWNEREPOCH (w) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	768	MSR_SGXOWNEREPOCH1	Package	Upper 64 Bit CR_SGXOWNEREPOCH (w) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Ovf_PMC0
		1	Thread	Ovf_PMC1
		2	Thread	Ovf_PMC2
		3	Thread	Ovf_PMC3
		4	Thread	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)
		5	Thread	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Thread	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)
		7	Thread	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)
		31:8		Reserved
	32	Thread	Ovf_FixedCtr0	

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		33	Thread	Ovf_FixedCtr1
		34	Thread	Ovf_FixedCtr2
		54:35		Reserved
		55	Thread	Trace_ToPA_PMI
		57:56		Reserved
		58	Thread	LBR_Frz
		59	Thread	CTR_Frz
		60	Thread	ASCI
		61	Thread	Ovf_Uncore
		62	Thread	Ovf_BufDSSAVE
		63	Thread	CondChgd
390H	912	IA32_PERF_GLOBAL_STATUS_RESET		See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Set 1 to clear Ovf_PMC0.
		1	Thread	Set 1 to clear Ovf_PMC1.
		2	Thread	Set 1 to clear Ovf_PMC2.
		3	Thread	Set 1 to clear Ovf_PMC3.
		4	Thread	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).
		5	Thread	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).
		6	Thread	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).
		7	Thread	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to clear Ovf_FixedCtr0.
		33	Thread	Set 1 to clear Ovf_FixedCtr1.
		34	Thread	Set 1 to clear Ovf_FixedCtr2.
		54:35		Reserved
		55	Thread	Set 1 to clear Trace_ToPA_PMI.
		57:56		Reserved
		58	Thread	Set 1 to clear LBR_Frz.
		59	Thread	Set 1 to clear CTR_Frz.
		60	Thread	Set 1 to clear ASCI.
		61	Thread	Set 1 to clear Ovf_Uncore.
62	Thread	Set 1 to clear Ovf_BufDSSAVE.		
63	Thread	Set 1 to clear CondChgd.		

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
391H	913	IA32_PERF_GLOBAL_STATUS_SET		See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Set 1 to cause Ovf_PMC0 = 1.
		1	Thread	Set 1 to cause Ovf_PMC1 = 1.
		2	Thread	Set 1 to cause Ovf_PMC2 = 1.
		3	Thread	Set 1 to cause Ovf_PMC3 = 1.
		4	Thread	Set 1 to cause Ovf_PMC4=1 (if CPUID.0AH:EAX[15:8] > 4).
		5	Thread	Set 1 to cause Ovf_PMC5=1 (if CPUID.0AH:EAX[15:8] > 5).
		6	Thread	Set 1 to cause Ovf_PMC6=1 (if CPUID.0AH:EAX[15:8] > 6).
		7	Thread	Set 1 to cause Ovf_PMC7=1 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to cause Ovf_FixedCtr0 = 1.
		33	Thread	Set 1 to cause Ovf_FixedCtr1 = 1.
		34	Thread	Set 1 to cause Ovf_FixedCtr2 = 1.
		54:35		Reserved
		55	Thread	Set 1 to cause Trace_ToPA_PMI = 1.
		57:56		Reserved
		58	Thread	Set 1 to cause LBR_Frz = 1.
		59	Thread	Set 1 to cause CTR_Frz = 1.
		60	Thread	Set 1 to cause ASCI = 1.
		61	Thread	Set 1 to cause Ovf_Uncore.
62	Thread	Set 1 to cause Ovf_BufDSSAVE.		
63		Reserved		
392H	913	IA32_PERF_GLOBAL_INUSE	Thread	See Table 2-2.
3F7H	1015	MSR_PEBS_FRONTEND	Thread	FrontEnd Precise Event Condition Select (R/W)
		2:0		Event Code Select
		3		Reserved
		4		Event Code Select High
		7:5		Reserved
		19:8		IDQ_Bubble_Length Specifier
		22:20		IDQ_Bubble_Width Specifier
		63:23		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
500H	1280	IA32_SGX_SVN_STATUS	Thread	Status and SVN Threshold of SGX Support for ACM (R/O)
		0		Lock See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1		Reserved
		23:16		SGX_SVN_SINIT See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		63:24		Reserved
560H	1376	IA32_RTIT_OUTPUT_BASE	Thread	Trace Output Base Register (R/W) See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Thread	Trace Output Mask Pointers Register (R/W) See Table 2-2.
570H	1392	IA32_RTIT_CTL	Thread	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		6:4		Reserved, must be zero.
		7		CR3 filter
		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		

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:40		Reserved, must be zero.
571H	1393	IA32_RTIT_STATUS	Thread	Tracing Status Register (R/W)
		0		FilterEn, writes ignored.
		1		ContexEn, writes ignored.
		2		TriggerEn, writes ignored.
		3		Reserved
		4		Error (R/W)
		5		Stopped
		31:6		Reserved, must be zero.
		48:32		PacketByteCnt
		63:49		Reserved, must be zero.
572H	1394	IA32_RTIT_CR3_MATCH	Thread	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match
580H	1408	IA32_RTIT_ADDRO_A	Thread	Region 0 Start Address (R/W)
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDRO_B	Thread	Region 0 End Address (R/W)
		63:0		See Table 2-2.
582H	1410	IA32_RTIT_ADDR1_A	Thread	Region 1 Start Address (R/W)
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Thread	Region 1 End Address (R/W)
		63:0		See Table 2-2.
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 14.10.4, "PPO/PP1 RAPL Domains."
64DH	1613	MSR_PLATFORM_ENERGY_COUNTER	Platform*	Platform Energy Counter (R/O) This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it. This MSR will read 0 if not valid.
		31:0		Total energy consumed by all devices in the platform that receive power from integrated power delivery mechanism, included platform devices are processor cores, SOC, memory, add-on or peripheral devices that get powered directly from the platform power delivery means. The energy units are specified in the MSR_RAPL_POWER_UNIT.Energy_Status_Unit.
		63:32		Reserved
64EH	1614	MSR_PPERF	Thread	Productive Performance Count (R/O)

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:0		Hardware's view of workload scalability. See Section 14.4.5.1.
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (RO) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		Thermal Status (RO) When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved
		4		Residency State Regulation Status (RO) When set, frequency is reduced below the operating system request due to residency state regulation limit.
		5		Running Average Thermal Limit Status (RO) When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).
		6		VR Therm Alert Status (RO) 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 (RO) When set, frequency is reduced below the operating system request due to VR thermal design current limit.
		8		Other Status (RO) 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 (RO) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.
		11		Package/Platform-Level PL2 Power Limiting Status (RO) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		24		Other Log When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved
652H	1618	MSR_PKG_HDC_CONFIG	Package	HDC Configuration (R/W)
		2:0		PKG_Cx_Monitor Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY.
		63: 3		Reserved
653H	1619	MSR_CORE_HDC_RESIDENCY	Core	Core HDC Idle Residency (R/O)
		63:0		Core_Cx_Duty_Cycle_Cnt
655H	1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	Package	Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle (R/O)
		63:0		Pkg_C2_Duty_Cycle_Cnt

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
656H	1622	MSR_PKG_HDC_DEEP_RESIDENCY	Package	Package Cx HDC Idle Residency (R/O)
		63:0		Pkg_Cx_Duty_Cycle_Cnt
658H	1624	MSR_WEIGHTED_CORE_CO	Package	Core-count Weighted C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.
659H	1625	MSR_ANY_CORE_CO	Package	Any Core C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor core in the package is in C0.
65AH	1626	MSR_ANY_GFXE_CO	Package	Any Graphics Engine C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.
65BH	1627	MSR_CORE_GFXE_OVERLAP_CO	Package	Core and Graphics Engine Overlapped C0 Residency (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.
65CH	1628	MSR_PLATFORM_POWER_LIMIT	Platform*	Platform Power Limit Control (R/W-L) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows.
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15		Enable Platform Power Limit #1 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.
		16		Platform Clamping Limitation #1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID (EAX=6):EAX[4] is set.
		23:17		Time Window for Platform Power Limit #1 Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH. The unit is specified in MSR_RAPLPOWER_UNIT[Time Unit].
		31:24		Reserved
		46:32		Platform Power Limit #2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit # 1).
		47		Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Thread	Last Branch Record 16 From IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.12.
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Thread	Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Thread	Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Thread	Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Thread	Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Thread	Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Thread	Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Thread	Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Thread	Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Thread	Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Thread	Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Thread	Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Thread	Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Thread	Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Thread	Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Thread	Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)
		0		PROCHOT Status (RO) When set, frequency is reduced due to assertion of external PROCHOT.
		1		Thermal Status (RO) When set, frequency is reduced due to a thermal event.
		4:2		Reserved.
		5		Running Average Thermal Limit Status (RO) When set, frequency is reduced due to running average thermal limit.
		6		VR Therm Alert Status (RO) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.
		7		VR Thermal Design Current Status (RO) When set, frequency is reduced due to VR TDC limit.
		8		Other Status (RO) When set, frequency is reduced due to electrical or other constraints.
		9		Reserved
		10		Package/Platform-Level Power Limiting PL1 Status (RO) When set, frequency is reduced due to package/platform-level power limiting PL1.
		11		Package/Platform-Level PL2 Power Limiting Status (RO) When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.
		12		Inefficient Operation Status (RO) When set, processor graphics frequency is operating below target frequency.
		15:13		Reserved
		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 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20:18		Reserved.
		21		Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24		Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		28		Inefficient Operation Log When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:29		Reserved
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)
		0		PROCHOT Status (RO) When set, frequency is reduced due to assertion of external PROCHOT.
		1		Thermal Status (RO) When set, frequency is reduced due to a thermal event.
		4:2		Reserved
		5		Running Average Thermal Limit Status (RO) When set, frequency is reduced due to running average thermal limit.
		6		VR Therm Alert Status (RO) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.
		7		VR Thermal Design Current Status (RO) When set, frequency is reduced due to VR TDC limit.
		8		Other Status (RO) When set, frequency is reduced due to electrical or other constraints.
		9		Reserved
		10		Package/Platform-Level Power Limiting PL1 Status (RO) When set, frequency is reduced due to package/Platform-level power limiting PL1.
		11		Package/Platform-Level PL2 Power Limiting Status (RO) When set, frequency is reduced due to package/Platform-level power limiting PL2/PL3.
		15:12		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		16		PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20:18		Reserved
		21		Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24		Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved
		26		Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		27		Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:28		Reserved
6D0H	1744	MSR_LASTBRANCH_16_TO_IP	Thread	Last Branch Record 16 To IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.12.
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Thread	Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Thread	Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Thread	Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Thread	Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Thread	Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Thread	Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Thread	Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Thread	Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Thread	Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Thread	Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Thread	Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Thread	Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Thread	Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Thread	Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Thread	Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, "Enabling HWP".
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Section 14.4.4, "Managing HWP".
773H	1907	IA32_HWP_INTERRUPT	Thread	See Section 14.4.6, "HWP Notifications".
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, "Managing HWP".
		7:0		Minimum Performance (R/W)
		15:8		Maximum Performance (R/W)
		23:16		Desired Performance (R/W)
		31:24		Energy/Performance Preference (R/W)
		41:32		Activity Window (R/W)
		42		Package Control (R/W)
		63:43		Reserved
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, "HWP Feedback".
D90H	3472	IA32_BNDCFGS	Thread	See Table 2-2.
DA0H	3488	IA32_XSS	Thread	See Table 2-2.
DB0H	3504	IA32_PKG_HDC_CTL	Package	See Section 14.5.2, "Package level Enabling HDC".
DB1H	3505	IA32_PM_CTL1	Thread	See Section 14.5.3, "Logical-Processor Level HDC Control".
DB2H	3506	IA32_THREAD_STALL	Thread	See Section 14.5.4.1, "IA32_THREAD_STALL".
DC0H	3520	MSR_LBR_INFO_0	Thread	Last Branch Record 0 Additional Information (R/W) One of 32 triplet of last branch record registers on the last branch record stack. This part of the stack contains flag, TSX-related and elapsed cycle information. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.9.1, "LBR Stack."
DC1H	3521	MSR_LBR_INFO_1	Thread	Last Branch Record 1 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC2H	3522	MSR_LBR_INFO_2	Thread	Last Branch Record 2 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC3H	3523	MSR_LBR_INFO_3	Thread	Last Branch Record 3 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DC4H	3524	MSR_LBR_INFO_4	Thread	Last Branch Record 4 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC5H	3525	MSR_LBR_INFO_5	Thread	Last Branch Record 5 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC6H	3526	MSR_LBR_INFO_6	Thread	Last Branch Record 6 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC7H	3527	MSR_LBR_INFO_7	Thread	Last Branch Record 7 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC8H	3528	MSR_LBR_INFO_8	Thread	Last Branch Record 8 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC9H	3529	MSR_LBR_INFO_9	Thread	Last Branch Record 9 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCAH	3530	MSR_LBR_INFO_10	Thread	Last Branch Record 10 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCBH	3531	MSR_LBR_INFO_11	Thread	Last Branch Record 11 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCCH	3532	MSR_LBR_INFO_12	Thread	Last Branch Record 12 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCDH	3533	MSR_LBR_INFO_13	Thread	Last Branch Record 13 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCEH	3534	MSR_LBR_INFO_14	Thread	Last Branch Record 14 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCFH	3535	MSR_LBR_INFO_15	Thread	Last Branch Record 15 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD0H	3536	MSR_LBR_INFO_16	Thread	Last Branch Record 16 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD1H	3537	MSR_LBR_INFO_17	Thread	Last Branch Record 17 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD2H	3538	MSR_LBR_INFO_18	Thread	Last Branch Record 18 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD3H	3539	MSR_LBR_INFO_19	Thread	Last Branch Record 19 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD4H	3520	MSR_LBR_INFO_20	Thread	Last Branch Record 20 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD5H	3521	MSR_LBR_INFO_21	Thread	Last Branch Record 21 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD6H	3522	MSR_LBR_INFO_22	Thread	Last Branch Record 22 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD7H	3523	MSR_LBR_INFO_23	Thread	Last Branch Record 23 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD8H	3524	MSR_LBR_INFO_24	Thread	Last Branch Record 24 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD9H	3525	MSR_LBR_INFO_25	Thread	Last Branch Record 25 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDAH	3526	MSR_LBR_INFO_26	Thread	Last Branch Record 26 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDBH	3527	MSR_LBR_INFO_27	Thread	Last Branch Record 27 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDCH	3528	MSR_LBR_INFO_28	Thread	Last Branch Record 28 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDDH	3529	MSR_LBR_INFO_29	Thread	Last Branch Record 29 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDEH	3530	MSR_LBR_INFO_30	Thread	Last Branch Record 30 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDFH	3531	MSR_LBR_INFO_31	Thread	Last Branch Record 31 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-40 lists the MSRs of uncore PMU for Intel processors with CPUID DisplayFamily_DisplayModel signatures of 06_4EH, 06_5EH, 06_8EH, 06_9EH, and 06_66H.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		43:0		Current count.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:44		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTRO	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTRO	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTRO	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTRO	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
736H	1846	MSR_UNC_CBO_3_PERFCTRO	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
E01H	3585	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4select.
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E02H	3586	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved

2.17.1 MSRs Introduced in 7th Generation and 8th Generation Intel® Core™ Processors based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Table 2-41 lists additional MSRs for 7th generation and 8th generation Intel Core processors with a CPUID DisplayFamily_DisplayModel signatures of 06_8EH and 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 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
80H	128	MSR_TRACE_HUB_STH ACPIBAR_BASE	Package	NPK Address Used by AET Messages (R/W)
		0		Lock Bit If set, this MSR cannot be re-written anymore. Lock bit has to be set in order for the AET packets to be directed to NPK MMIO.
		17:1		Reserved
		63:18		ACPIBAR_BASE_ADDRESS AET target address in NPK MMIO space.
1F4H	500	MSR_PRMRM_PHYS_BASE	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MemType PRMRM BASE MemType.
		11:3		Reserved
		45:12		Base PRMRM Base Address.
		63:46		Reserved
1F5H	501	MSR_PRMRM_PHYS_MASK	Core	Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)
		9:0		Reserved
		10		Lock Lock bit for the PRMRM.

Table 2-41. Additional MSRs Supported by 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		11		VLD Enable bit for the PRMRR.
		45:12		Mask PRMRR MASK bits.
		63:46		Reserved
1FBH	507	MSR_PRMRR_VALID_CONFIG	Core	Valid PRMRR Configurations (R/W)
		0		1M supported MEE size.
		4:1		Reserved
		5		32M supported MEE size.
		6		64M supported MEE size.
		7		128M supported MEE size.
		31:8		Reserved
2F4H	756	MSR_UNCORE_PRMRR_PHYS_BASE ¹	Package	(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.
		11:0		Reserved
		PAWIDTH-1:12		Range Base This field corresponds to bits PAWIDTH-1:12 of the base address memory range which is allocated to PRMRR memory.
		63:PAWIDTH		Reserved
2F5H	757	MSR_UNCORE_PRMRR_PHYS_MASK ¹	Package	(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.
		9:0		Reserved
		10		Lock Setting this bit locks all writeable settings in this register, including itself.
		11		Range_En Indicates whether the PRMRR range is enabled and valid.
		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

Table 2-41. Additional MSRs Supported by 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		6:0		MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.
		7		Reserved
		14:8		MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.
		63:15		Reserved

NOTES:

1. This MSR is specific to 7th generation and 8th generation Intel® Core™ processors.

2.17.2 MSRs Specific to 8th Generation Intel® Core™ i3 Processors

Table 2-42 lists additional MSRs for 8th generation Intel Core i3 processors with a CPUID DisplayFamily_DisplayModel signature of 06_66H. For an MSR listed in Table 2-42 that also appears in the model-specific tables of prior generations, Table 2-42 supersedes prior generation tables.

Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)
		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		
350H	848	MSR_BR_DETECT_CTRL		Branch Monitoring Global Control (R/W)

**Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		EnMonitoring Global enable for branch monitoring.
		1		EnExcept Enable branch monitoring event signaling on threshold trip. The branch monitoring event handler is signaled via the existing PMI signaling mechanism as programmed from the corresponding local APIC LVT entry.
		2		EnLBRFrz Enable LBR freeze on threshold trip. This will cause the LBR frozen bit 58 to be set in IA32_PERF_GLOBAL_STATUS when a triggering condition occurs and this bit is enabled.
		3		DisableInGuest When set to '1', branch monitoring, event triggering and LBR freeze actions are disabled when operating at VMX non-root operation.
		7:4		Reserved
		17:8		WindowSize Window size defined by WindowCntSel. Values 0 - 1023 are supported. Once the Window counter reaches the WindowSize count both the Window Counter and all Branch Monitoring Counters are cleared.
		23:18		Reserved
		25:24		WindowCntSel Window event count select: '00 = Instructions retired. '01 = Branch instructions retired '10 = Return instructions retired. '11 = Indirect branch instructions retired.
		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
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.

**Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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

**Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
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.
		15		MispredEventCnt Mispredict events counting behavior: '0 = Mispredict events are counted in a window. '1 = Mispredict events are counted based on a consecutive occurrence. CntrThreshold is treated as # of consecutive mispredicts. This control bit only applies to events specified by CntrEvSel that involve a prediction (0000010, 0000011, 0000100). Setting this bit for other events is ignored.
	63:16			Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Package C3 Residency Counter (R/O)
		63:0		Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		6:0		MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.
		7		Reserved
		14:8		MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.
		63:15		Reserved

Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Core C1 Residency Counter (R/O)
		63:0		Value since last reset for the Core C1 residency. Counter rate is the Max Non-Turbo frequency (same as TSC). This counter counts in case both of the core's threads are in an idle state and at least one of the core's thread residency is in a C1 state or in one of its sub states. The counter is updated only after a core C state exit. Note: Always reads 0 if core C1 is unsupported. A value of zero indicates that this processor does not support core C1 or never entered core C1 level state.
662H	1634	MSR_CORE_C3_RESIDENCY	Core	Core C3 Residency Counter (R/O)
		63:0		Will always return 0.

Table 2-43 lists the MSRs of uncore PMU for Intel processors with CPUID signature 06_66H.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Report the number of C-Box units with performance counters, including processor cores and processor graphics.
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSELO	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, Counter 1 Event Select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSELO	Package	Uncore C-Box 0, Counter 0 Event Select MSR

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
702H	1794	MSR_UNC_CBO_0_PERFCTRO	Package	Uncore C-Box 0, Performance Counter 0
703H	1795	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
708H	1800	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
709H	1801	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
70AH	1802	MSR_UNC_CBO_1_PERFCTRO	Package	Uncore C-Box 1, Performance Counter 0
70BH	1803	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
710H	1808	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
712H	1810	MSR_UNC_CBO_2_PERFCTRO	Package	Uncore C-Box 2, Performance Counter 0
713H	1811	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
718H	1816	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
719H	1817	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
71AH	1818	MSR_UNC_CBO_3_PERFCTRO	Package	Uncore C-Box 3, Performance Counter 0
71BH	1819	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
720H	1824	MSR_UNC_CBO_4_PERFEVTSELO	Package	Uncore C-Box 4, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
722H	1826	MSR_UNC_CBO_4_PERFCTRO	Package	Uncore C-Box 4, Performance Counter 0
723H	1827	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
728H	1832	MSR_UNC_CBO_5_PERFEVTSELO	Package	Uncore C-Box 5, Counter 0 Event Select MSR
729H	1833	MSR_UNC_CBO_5_PERFEVTSEL1	Package	Uncore C-Box 5, Counter 1 Event Select MSR
72AH	1834	MSR_UNC_CBO_5_PERFCTRO	Package	Uncore C-Box 5, Performance Counter 0
72BH	1835	MSR_UNC_CBO_5_PERFCTR1	Package	Uncore C-Box 5, Performance Counter 1
730H	1840	MSR_UNC_CBO_6_PERFEVTSELO	Package	Uncore C-Box 6, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_6_PERFEVTSEL1	Package	Uncore C-Box 6, Counter 1 Event Select MSR
732H	1842	MSR_UNC_CBO_6_PERFCTRO	Package	Uncore C-Box 6, Performance Counter 0
733H	1843	MSR_UNC_CBO_6_PERFCTR1	Package	Uncore C-Box 6, Performance Counter 1
738H	1848	MSR_UNC_CBO_7_PERFEVTSELO	Package	Uncore C-Box 7, Counter 0 Event Select MSR
739H	1849	MSR_UNC_CBO_7_PERFEVTSEL1	Package	Uncore C-Box 7, Counter 1 Event Select MSR
73AH	1850	MSR_UNC_CBO_7_PERFCTRO	Package	Uncore C-Box 7, Performance Counter 0
73BH	1851	MSR_UNC_CBO_7_PERFCTR1	Package	Uncore C-Box 7, Performance Counter 1
E01H	3585	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
	4			Slice 4select.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved
E02H	3586	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved

2.17.3 MSRs Introduced in 10th Generation Intel® Core™ Processors

Table 2-44 lists additional MSRs for 10th generation Intel Core processors with a CPUID DisplayFamily_DisplayModel signature values of 06_7DH and 06_7EH. For an MSR listed in Table 2-44 that also appears in the model-specific tables of prior generations, Table 2-44 supersedes prior generation tables.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		28:0		Reserved.
		29		Enable #AC(0) exception for split locked accesses: Cause #AC(0) exception for split locked access at all CPL irrespective of CR0.AM or EFLAGS.AC. If bits 29 and 31 are both set, bit 29 takes precedence.
		30		Reserved.
		31		Reserved.
48H	72	IA32_SPEC_CTRL	Core	See Table 2-2.
49H	73	IA32_PREDICT_CMD	Thread	See Table 2-2.
8CH	140	IA32_SGXLEPUBKEYHASH0	Thread	See Table 2-2.
8DH	141	IA32_SGXLEPUBKEYHASH1	Thread	See Table 2-2.
8EH	142	IA32_SGXLEPUBKEYHASH2	Thread	See Table 2-2.
8FH	143	IA32_SGXLEPUBKEYHASH3	Thread	See Table 2-2.
A0H	160	MSR_BIOS_MCU_ERRORCODE	Package	BIOS MCU ERRORCODE (R/O) This MSR indicates if WRMSR 0x79 failed to configure PRM memory and gives a hint to debug BIOS.
		15:0	Package	Error Codes (R/O)

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		30:16		Reserved.
		31	Thread	MCU Partial Success (R/O) When set to 1, WRMSR 0x79 skipped part of the functionality during BIOS.
A5H	165	MSR_FIT_BIOS_ERROR	Thread	FIT BIOS ERROR (R/W) Report error codes for debug in case the processor failed to parse the Firmware Table in BIOS. Can also be used to log BIOS information.
		7:0		Error Codes (R/W) Error codes for debug.
		15:8		Entry Type (R/W) Failed FIT entry type.
		16		FIT MCU Entry (R/W) FIT contains MCU entry.
		62:17		Reserved.
		63		LOCK (R/W) When set to 1, writes to this MSR will be skipped.
10BH	267	IA32_FLUSH_CMD	Thread	See Table 2-2.
151H	337	MSR_BIOS_DONE	Thread	BIOS Done (R/WO)
		0	Thread	BIOS Done Indication (R/WO) Set by BIOS when it finishes programming the processor and wants to lock the memory configuration from changes by software that is running on this thread. Writes to the bit will be ignored if EAX[0] is 0.
		1	Package	Package BIOS Done Indication (R/O) When set to 1, all threads in the package have bit 0 of this MSR set.
		31:2		Reserved.
1F1H	497	MSR_CRASHLOG_CONTROL	Thread	Write Data to a Crash Log Configuration
		0		CDDIS: CrashDump_Disable If set, indicates that Crash Dump is disabled.
		63:1		Reserved.
2A0H	672	MSR_PRMRR_BASE_0	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MEMTYPE: PRMRR BASE Memory Type.
		3		CONFIGURED: PRMRR BASE Configured.
		11:4		Reserved.
		51:12		BASE: PRMRR Base Address.
		63:52		Reserved.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
30CH	780	IA32_FIXED_CTR3	Thread	Fixed-Function Performance Counter Register 3 (R/W) Bit definitions are the same as found in IA32_FIXED_CTR0, offset 309H. See Table 2-2.
329H	809	MSR_PERF_METRICS	Thread	Performance Metrics (R/W) Reports metrics directly. Software can check (and/or expose to its guests) the availability of PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15).
		7:0		Retiring. Percent of utilized slots by uops that eventually retire (commit).
		15:8		Bad Speculation. Percent of wasted slots due to incorrect speculation, covering utilized by uops that do not retire, or recovery bubbles (unutilized slots).
		23:16		Frontend Bound. Percent of unutilized slots where front-end did not deliver a uop while back-end is ready.
		31:24		Backend Bound. Percent of unutilized slots where a uop was not delivered to back-end due to lack of back-end resources.
		63:25		Reserved.
3F2H	1010	MSR_PEBS_DATA_CFG	Thread	PEBS Data Configuration (R/W) Provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.
		0		Memory Info. Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.
		1		GPRs. Setting this bit will capture the contents of the General Purpose registers in the PEBS record.
		2		XMMs. Setting this bit will capture the contents of the XMM registers in the PEBS record.
		3		LBRs. Setting this bit will capture LBR TO, FROM and INFO in the PEBS record.
		23:4		Reserved.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31:24		LBR Entries. Set the field to the desired number of entries - 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, software can use LBR_entries that is multiple of 3.
541H	1345	MSR_CORE_UARCH_CTL	Core	Core Microarchitecture Control MSR (R/W)
		0		L1 Scrubbing Enable When set to 1, enable L1 scrubbing.
		31:1		Reserved.
657H	1623	MSR_FAST_UNCORE_MSRS_CTL	Thread	Fast WRMSR/RDMSR Control MSR (R/W)
		3:0		FAST_ACCESS_ENABLE: Bit 0: When set to '1', provides a hint for the hardware to enable fast access mode for the IA32_HWP_REQUEST MSR. This bit is sticky and is cleaned by the hardware only during reset time. This bit is valid only if FAST_UNCORE_MSRS_CAPABILITY[0] is set. Setting this bit will cause CPUID[6].EAX[18] to be set.
		31:4		Reserved.
65EH	1630	MSR_FAST_UNCORE_MSRS_STATUS	Thread	Indication of Uncore MSRs, Post Write Activates
		0		Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor. Software can use the status of this bit to avoid overwriting IA32_HWP_REQUEST.
		31:1		Reserved.
65FH	1631	MSR_FAST_UNCORE_MSRS_CAPABILITY	Thread	Fast WRMSR/RDMSR Enumeration MSR (R/O)
		3:0		MSRS_CAPABILITY: Bit 0: If set to '1', hardware supports the fast access mode for the IA32_HWP_REQUEST MSR.
		31:4		Reserved.
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Table 2-2.
775H	1909	IA32_PECI_HWP_REQUEST_INFO	Thread	See Table 2-2.
777H	1911	IA32_HWP_STATUS	Thread	See Table 2-2.

2.17.4 MSRs Introduced in 11th Generation Intel® Core™ Processors based on Tiger Lake Microarchitecture

Table 2-45 lists additional MSRs for 11th generation Intel Core processors with CPUID DisplayFamily_DisplayModel signatures of 06_8CH and 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 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
A0H	160	MSR_BIOS_MCU_ERRORCODE	Package	BIOS MCU ERRORCODE (R/O)
		15:0		Error Codes
		31:16		Reserved
A7H	167	MSR_BIOS_DEBUG	Thread	BIOS DEBUG (R/O) This MSR indicates if WRMSR 79H failed to configure PRM memory and gives a hint to debug BIOS.
		30:0		Reserved
		31		MCU Partial Success When set to 1, WRMSR 79H skipped part of the functionality during BIOS.
		63:32		Reserved
CFH	207	IA32_CORE_CAPABILITIES	Package	IA32 Core Capabilities Register (R/O) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		1:0		Reserved
		2		FUSA_SUPPORTED
		3		RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.
		4		Reserved
		5		SPLIT_LOCK_DISABLE_SUPPORTED When set to 1, the ability to set MEMORY_CONTROL (MSR 33H) bit 29 enables an #AC to be created when a split lock is detected.
		31:6		Reserved

**Table 2-45. Additional MSRs Supported by 11th Generation Intel® Core™ Processors
Based on Tiger Lake Microarchitecture**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
492H	1170	IA32_VMX_PROCBASED_CTL3	Core	IA32_VMX_PROCBASED_CTL3 This MSR enumerates the allowed 1-settings of the third set of processor-based controls. Specifically, VM entry allows bit X of the tertiary processor-based VM-execution controls to be 1 if and only if bit X of the MSR is set to 1. If bit X of the MSR is cleared to 0, VM entry fails if control X and the “activate tertiary controls” primary processor-based VM-execution control are both 1.
		0		LOADIWKEY This control determines whether executions of LOADIWKEY cause VM exits.
		63:1		Reserved
601H	1537	MSR_VR_CURRENT_CONFIG	Package	Power Limit 4 (PL4) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.
		12:0		PL4 Value PL4 value in 0.125 A increments. This field is locked by VR_CURRENT_CONFIG[LOCK]. When the LOCK bit is set to 1b, this field becomes Read Only.
		30:13		Reserved
		31		Lock Indication (LOCK) This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1b, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.
		62:32		Not in use.
		63		Reserved
6A0H	1696	IA32_U_CET		Configure User Mode CET (R/W) See Table 2-2.
6A2H	1698	IA32_S_CET		Configure Supervisor Mode CET (R/W) See Table 2-2.
6A4H	1700	IA32_PL0_SSP		Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.
6A5H	1701	IA32_PL1_SSP		Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.
6A6H	1702	IA32_PL2_SSP		Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.

**Table 2-45. Additional MSRs Supported by 11th Generation Intel® Core™ Processors
Based on Tiger Lake Microarchitecture**

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6A7H	1703	IA32_PL3_SSP		Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.
6A8H	1704	IA32_INTERRUPT_SSP_TABLE_ADDR		Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.
981H	2433	IA32_TME_CAPABILITY		See Table 2-2.
982H	2434	IA32_TME_ACTIVATE		See Table 2-2.
983H	2435	IA32_TME_EXCLUDE_MASK		See Table 2-2.
984H	2436	IA32_TME_EXCLUDE_BASE		See Table 2-2.
990H	2448	IA32_COPY_STATUS ¹	Thread	See Table 2-2.
991H	2449	IA32_IWKEYBACKUP_STATUS ¹	Platform	See Table 2-2.
C82H	3202	IA32_L2_QOS_CFG	Core	IA32_CR_L2_QOS_CFG This MSR provides software an enumeration of the parameters that L2 QoS (Intel RDT) support in any particular implementation.
		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 17, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features” for further details on CDP.
		31:1		Reserved
D10H - D17H	3220 - 3351	IA32_L2_QOS_MASK [0-7]	Package	IA32_CR_L2_QOS_MASK [0-7] Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 17, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features”.
		19:0		WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.4.2:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.
		31:20		Reserved
D91H	3473	IA32_COPY_LOCAL_TO_PLATFORM ¹	Thread	See Table 2-2.
D92H	3474	IA32_COPY_PLATFORM_TO_LOCAL ¹	Thread	See Table 2-2.

NOTES:

1. Further details on Key Locker and usage of this MSR can be found here:

<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.17.5 MSRs Introduced in 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Table 2-47 lists additional MSRs for 12th generation Intel Core processors with CPUID DisplayFamily_DisplayModel signatures of 06_97H, 06_9AH, and 06_BFH. Table 2-48 lists the MSRs unique to the processor P-core. Table 2-48 lists the MSRs unique to the processor E-core.

The MSRs listed in Table 2-44¹ and Table 2-45 are also supported by these processors. For an MSR listed in Table 2-46, Table 2-47, or Table 2-48 that also appears in the model-specific tables of prior generations, Table 2-48, Table 2-47 and Table 2-48 supersede prior generation tables.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		26:0		Reserved.
		27		UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.
		28		UC_LOCK_DISABLE If set to 1, a UC load lock will trigger a #GP fault.
		29		Enable #AC(0) exception for split locked accesses: Cause #AC(0) exception for split locked access at all CPL irrespective of CRO.AM or EFLAGS.AC. If bits 29 and 31 are both set, bit 29 takes precedence.
		30		Reserved.
		31		Reserved.
BCH	188	IA32_MISC_PACKAGE_CTL5	Package	Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.
C7H	199	IA32_PMC6	Core	General Performance Counter 6 (R/W) See Table 2-2.
C8H	200	IA32_PMC7	Core	General Performance Counter 7 (R/W) See Table 2-2.

1. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core: 30CH, 329H, 541H and 657H. The MSR at address 657H is not supported in the 12th generation Intel Core processor P-core.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CFH	207	IA32_CORE_CAPABILITIES	Package	IA32 Core Capabilities Register (R/O) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		0		STLB_QOS_SUPPORTED When set to 1, the STLB QoS feature is supported and the STLB QoS MSRs (1A8FH - 1A97H) are accessible. When set to 0, access to these MSRs will #GP.
		1		Reserved
		2		FUSA_SUPPORTED
		3		RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.
		4		UC_LOCK_DISABLE_SUPPORTED When set to 1, processor supports UC load lock disable feature.
		5		SPLIT_LOCK_DISABLE_SUPPORTED When set to 1, the ability to set MEMORY_CONTROL (MSR 33H) bit 29 enables an #AC to be created when a split lock is detected.
		6		SNOOP_FILTER_QOS_SUPPORTED When set to 1, the Snoop Filter QoS Mask MSRs are supported. When set to 0, access to these MSRs will #GP.
		7		UC_STORE_THROTTLING_SUPPORTED When set 1, UC Store throttle capability exist through MSR_MEMORY_CTRL (33H) bit 27.
	31:8		Reserved	
E1H	225	IA32_UMWAIT_CONTROL		UMWAIT Control (R/W) See Table 2-2.
10AH	266	IA32_ARCH_CAPABILITIES		Enumeration of Architectural Features (R/O) See Table 2-2.
18CH	396	IA32_PERFEVTSEL6	Core	See Table 2-20.
18DH	397	IA32_PERFEVTSEL7	Core	See Table 2-20.
195H	405	IA32_OVERCLOCKING_STATUS	Package	Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR. See Table 2-2.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_PRIMARY_TURBO_RATIO_LIMIT	Package	Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.
		7:0		MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.
		15:8		MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.
		23:16		MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.
		31:24		MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.
		39:32		MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.
		47:40		MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.
		55:48		MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.
		63:56		MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.
493H	1171	IA32_VMX_EXIT_CTL2		See Table 2-2.
4C7H	1223	IA32_A_PMC6		Full Width Writable IA32_PMC6 Alias (R/W) See Table 2-2.
4C8H	1224	IA32_A_PMC7		Full Width Writable IA32_PMC7 Alias (R/W) See Table 2-2.
650H	1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	Package	Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.
		7:0		MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.
		15:8		MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.
		23:16		MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31:24		MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.
		39:32		MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.
		47:40		MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.
		55:48		MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.
		63:56		MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.
6E1H	1761	IA32_PKRS		Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.
776H	1910	IA32_HWP_CTL		See Table 2-2.
981H	2433	IA32_TME_CAPABILITY		Memory Encryption Capability MSR See Table 2-2.
1200H - 121FH	4608 - 4639	IA32_LBR_x_INFO		Last Branch Record Entry X Info Register (R/W) See Table 2-2.
14CEH	5326	IA32_LBR_CTL		Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.
14CFH	5327	IA32_LBR_DEPTH		Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.
1500H - 151FH	5376 - 5407	IA32_LBR_x_FROM_IP		Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.
1600H - 161FH	5632 - 5663	IA32_LBR_x_TO_IP		Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.
17D2H	6098	IA32_THREAD_FEEDBACK_CHAR		Thread Feedback Characteristics (R/O) See Table 2-2.
17D4H	6100	IA32_HW_FEEDBACK_THREAD_CONFIG		Hardware Feedback Thread Configuration (R/W) See Table 2-2.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
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 generation Intel Core processor P-core. These MSRs are not supported on the processor E-core.

Table 2-47. MSRs Supported by 12th Generation Intel® Core™ Processor P-core

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3F7H	1015	MSR_PEBS_FRONTEND	Thread	FrontEnd Precise Event Condition Select (R/W) See Table 2-39.
540H	1344	MSR_THREAD_UARCH_CTL	Thread	Thread Microarchitectural Control (R/W)
		0		WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).
		63:1		Reserved
541H	1345	MSR_CORE_UARCH_CTL	Core	Core Microarchitecture Control MSR (R/W) See Table 2-44.
D10H - D17H	3220 - 3351	IA32_L2_QOS_MASK_[0-7]	Core	IA32_CR_L2_QOS_MASK_[0-7] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 17, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features”.
		19:0		WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.4.2:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.
		31:20		Reserved

The MSRs listed in Table 2-48 are unique to the 12th generation Intel Core processor E-core. These MSRs are not supported on the processor P-core.

Table 2-48. MSRs Supported by 12th Generation Intel® Core™ Processor E-core

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D10H - D1FH	3220 - 3359	IA32_L2_QOS_MASK_[0-15]	Module	IA32_CR_L2_QOS_MASK_[0-15] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 17, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features”.
		19:0		WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.4.2:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.
		31:20		Reserved
1309H - 130BH	4873 - 4875	MSR_RELOAD_FIXED_CTRx		Reload value for IA32_FIXED_CTRx (R/W)
		47:0		Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.
		63:48		Reserved
14C1H - 14C6H	5313 - 5318	MSR_RELOAD_PMCx	Core	Reload value for IA32_PMCx (R/W)
		47:0		Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.
		63:48		Reserved

Table 2-49 lists the MSRs of uncore PMU for Intel processors with CPUID DisplayFamily_DisplayModel signatures of 06_97H, 06_9AH, and 06_BFH.

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).
		63:4		Reserved
2000H	8192	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, Counter 0 Event Select MSR
2001H	8193	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
2002H	8194	MSR_UNC_CBO_0_PERFCTRO	Package	Uncore C-Box 0, Performance Counter 0
2003H	8195	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
2008H	8200	MSR_UNC_CBO_1_PERFEVTSELO	Package	Uncore C-Box 1, Counter 0 Event Select MSR
2009H	8201	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
200AH	8202	MSR_UNC_CBO_1_PERFCTRO	Package	Uncore C-Box 1, Performance Counter 0
200BH	8203	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
2010H	8208	MSR_UNC_CBO_2_PERFEVTSELO	Package	Uncore C-Box 2, Counter 0 Event Select MSR
2011H	8209	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
2012H	8210	MSR_UNC_CBO_2_PERFCTRO	Package	Uncore C-Box 2, Performance Counter 0
2013H	8211	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
2018H	8216	MSR_UNC_CBO_3_PERFEVTSELO	Package	Uncore C-Box 3, Counter 0 Event Select MSR
2019H	8217	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
201AH	8218	MSR_UNC_CBO_3_PERFCTRO	Package	Uncore C-Box 3, Performance Counter 0
201BH	8219	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
2020H	8224	MSR_UNC_CBO_4_PERFEVTSELO	Package	Uncore C-Box 4, Counter 0 Event Select MSR
2021H	8225	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
2022H	8226	MSR_UNC_CBO_4_PERFCTRO	Package	Uncore C-Box 4, Performance Counter 0
2023H	8227	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
2028H	8232	MSR_UNC_CBO_5_PERFEVTSELO	Package	Uncore C-Box 5, Counter 0 Event Select MSR
2029H	8233	MSR_UNC_CBO_5_PERFEVTSEL1	Package	Uncore C-Box 5, Counter 1 Event Select MSR
202AH	8234	MSR_UNC_CBO_5_PERFCTRO	Package	Uncore C-Box 5, Performance Counter 0
202BH	8235	MSR_UNC_CBO_5_PERFCTR1	Package	Uncore C-Box 5, Performance Counter 1
2030H	8240	MSR_UNC_CBO_6_PERFEVTSELO	Package	Uncore C-Box 6, Counter 0 Event Select MSR
2031H	8241	MSR_UNC_CBO_6_PERFEVTSEL1	Package	Uncore C-Box 6, Counter 1 Event Select MSR
2032H	8242	MSR_UNC_CBO_6_PERFCTRO	Package	Uncore C-Box 6, Performance Counter 0
2033H	8243	MSR_UNC_CBO_6_PERFCTR1	Package	Uncore C-Box 6, Performance Counter 1
2038H	8248	MSR_UNC_CBO_7_PERFEVTSELO	Package	Uncore C-Box 7, Counter 0 Event Select MSR
2039H	8249	MSR_UNC_CBO_7_PERFEVTSEL1	Package	Uncore C-Box 7, Counter 1 Event Select MSR
203AH	8250	MSR_UNC_CBO_7_PERFCTRO	Package	Uncore C-Box 7, Performance Counter 0
203BH	8251	MSR_UNC_CBO_7_PERFCTR1	Package	Uncore C-Box 7, Performance Counter 1
2040H	8256	MSR_UNC_CBO_8_PERFEVTSELO	Package	Uncore C-Box 8, Counter 0 Event Select MSR
2041H	8257	MSR_UNC_CBO_8_PERFEVTSEL1	Package	Uncore C-Box 8, Counter 1 Event Select MSR
2042H	8258	MSR_UNC_CBO_8_PERFCTRO	Package	Uncore C-Box 8, Performance Counter 0
2043H	8259	MSR_UNC_CBO_8_PERFCTR1	Package	Uncore C-Box 8, Performance Counter 1
2048H	8264	MSR_UNC_CBO_9_PERFEVTSELO	Package	Uncore C-Box 9, Counter 0 Event Select MSR
2049H	8265	MSR_UNC_CBO_9_PERFEVTSEL1	Package	Uncore C-Box 9, Counter 1 Event Select MSR
204AH	8266	MSR_UNC_CBO_9_PERFCTRO	Package	Uncore C-Box 9, Performance Counter 0

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
204BH	8267	MSR_UNC_CBO_9_PERFCTR1	Package	Uncore C-Box 9, Performance Counter 1
2FD0H	12240	MSR_UNC_ARB_0_PERFEVTSEL0	Package	Uncore Arb Unit 0, Counter 0 Event Select MSR
2FD1H	12241	MSR_UNC_ARB_0_PERFEVTSEL1	Package	Uncore Arb Unit 0, Counter 1 Event Select MSR
2FD2H	12242	MSR_UNC_ARB_0_PERFCTRO	Package	Uncore Arb Unit 0, Performance Counter 0
2FD3H	12243	MSR_UNC_ARB_0_PERFCTR1	Package	Uncore Arb Unit 0, Performance Counter 1
2FD4H	12244	MSR_UNC_ARB_0_PERF_STATUS	Package	Uncore Arb Unit 0, Performance Status
2FD5H	12245	MSR_UNC_ARB_0_PERF_CTRL	Package	Uncore Arb Unit 0, Performance Control
2FD8H	12248	MSR_UNC_ARB_1_PERFEVTSEL0	Package	Uncore Arb Unit 1, Counter 0 Event Select MSR
2FD9H	12249	MSR_UNC_ARB_1_PERFEVTSEL1	Package	Uncore Arb Unit 1, Counter 1 Event Select MSR
2FDAH	12250	MSR_UNC_ARB_1_PERFCTRO	Package	Uncore Arb Unit 1, Performance Counter 0
2FDBH	12251	MSR_UNC_ARB_1_PERFCTR1	Package	Uncore Arb Unit 1, Performance Counter 1
2FDCH	12252	MSR_UNC_ARB_1_PERF_STATUS	Package	Uncore Arb Unit 1, Performance Status
2FDDH	12253	MSR_UNC_ARB_1_PERF_CTRL	Package	Uncore Arb Unit 1, Performance Control
2FDEH	12254	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
2FDFH	12255	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		43:0		Current count.
		63:44		Reserved
2FF0H	12272	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4 select.
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved
2FF2H	12274	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved

2.17.6 MSRs Introduced in Intel® Xeon® Processor Scalable Family

Intel® Xeon® Processor Scalable Family (CUID DisplayFamily_DisplayModel = 06_55H) support the MSRs listed in Table 2-50.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Enable VMX Inside SMX Operation (R/WL)
		2		Enable VMX Outside SMX Operation (R/WL)
		14:8		SENTER Local Functions Enables (R/WL)
		15		SENTER Global Functions Enable (R/WL)
		18		SGX Global Functions Enable (R/WL)
		20		LMCE_ENABLED (R/WL)
		63:21		Reserved
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) See Table 2-26.
		22:16		Reserved.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		23	Package	PPIN_CAP (R/O) See Table 2-26.
		27:24		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.
		30	Package	Programmable TJ OFFSET (R/O) See Table 2-26.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) See Table 2-26.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .
		2:0		Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		I/O MWAIT Redirection Enable (R/W)
		14:11		Reserved
		15		CFG Lock (R/W0)
		16		Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).
		24:17		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		27		Enable C3 Undemotion (R/W)
		28		Enable C1 Undemotion (R/W)
		29		Package C State Demotion Enable (R/W)
		30		Package C State UnDemotion Enable (R/W)
		63:31		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		7:0		Count
		8		MCG_CTL_P
		9		MCG_EXT_P
		10		MCP_CMCL_P
		11		MCG_TES_P
		15:12		Reserved
		23:16		MCG_EXT_CNT
		24		MCG_SER_P
		25		MCG_EM_P
		26		MCG_ELOG_P
		63:27		Reserved
17DH	381	MSR_SMM_MCA_CAP	THREAD	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and a host-space interface is available to SMM handler.
		59		Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface is available to SMM handler.
		63:60		Reserved
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O) See Table 2-2.
		1		Thermal Status Log (R/WCO) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		3		PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.
		4		Critical Temperature Status (R/O) See Table 2-2.
		5		Critical Temperature Status Log (R/WCO) See Table 2-2.
		6		Thermal Threshold #1 Status (R/O) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WCO) See Table 2-2.
		8		Thermal Threshold #2 Status (R/O) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WCO) See Table 2-2.
		10		Power Limitation Status (R/O) See Table 2-2.
		11		Power Limitation Log (R/WCO) See Table 2-2.
		12		Current Limit Status (R/O) See Table 2-2.
		13		Current Limit Log (R/WCO) See Table 2-2.
		14		Cross Domain Limit Status (R/O) See Table 2-2.
		15		Cross Domain Limit Log (R/WCO) See Table 2-2.
		22:16		Digital Readout (R/O) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O) See Table 2-2.
		31		Reading Valid (R/O) See Table 2-2.
63:32		Reserved		
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R/O) See Table 2-26.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		27:24		TCC Activation Offset (R/W) See Table 2-26.
		63:28		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	This register defines the ratio limits. RATIO[0:7] must be populated in ascending order. RATIO[i+1] must be less than or equal to RATIO[i]. Entries with RATIO[i] will be ignored. If any of the rules above are broken, the configuration is silently rejected. If the programmed ratio is: <ul style="list-style-type: none"> ▪ Above the fused ratio for that core count, it will be clipped to the fuse limits (assuming !OC). ▪ Below the min supported ratio, it will be clipped.
		7:0		RATIO_0 Defines ratio limits.
		15:8		RATIO_1 Defines ratio limits.
		23:16		RATIO_2 Defines ratio limits.
		31:24		RATIO_3 Defines ratio limits.
		39:32		RATIO_4 Defines ratio limits.
		47:40		RATIO_5 Defines ratio limits.
		55:48		RATIO_6 Defines ratio limits.
		63:56		RATIO_7 Defines ratio limits.
1AEH	430	MSR_TURBO_RATIO_LIMIT_CORES	Package	This register defines the active core ranges for each frequency point. NUMCORE[0:7] must be populated in ascending order. NUMCORE[i+1] must be greater than NUMCORE[i]. Entries with NUMCORE[i] == 0 will be ignored. The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, the configuration is silently rejected.
		7:0		NUMCORE_0 Defines the active core ranges for each frequency point.
		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.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		31:24		NUMCORE_3 Defines the active core ranges for each frequency point.
		39:32		NUMCORE_4 Defines the active core ranges for each frequency point.
		47:40		NUMCORE_5 Defines the active core ranges for each frequency point.
		55:48		NUMCORE_6 Defines the active core ranges for each frequency point.
		63:56		NUMCORE_7 Defines the active core ranges for each frequency point.
280H	640	IA32_MC0_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.
401H	1025	IA32_MC0_STATUS	Core	
402H	1026	IA32_MC0_ADDR	Core	
403H	1027	IA32_MC0_MISC	Core	

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.
405H	1029	IA32_MC1_STATUS	Core	
406H	1030	IA32_MC1_ADDR	Core	
407H	1031	IA32_MC1_MISC	Core	
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.
409H	1033	IA32_MC2_STATUS	Core	
40AH	1034	IA32_MC2_ADDR	Core	
40BH	1035	IA32_MC2_MISC	Core	
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.
40DH	1037	IA32_MC3_STATUS	Core	
40EH	1038	IA32_MC3_ADDR	Core	
40FH	1039	IA32_MC3_MISC	Core	
410H	1040	IA32_MC4_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.
411H	1041	IA32_MC4_STATUS	Package	
412H	1042	IA32_MC4_ADDR	Package	
413H	1043	IA32_MC4_MISC	Package	
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs". Bank MC19 reports MC errors from a link interconnect module.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\wedge}ESU$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
620H	1568	MSR_UNCORE_RATIO_LIMIT	Package	Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
C8DH	3213	IA32_QM_EVTSEL	THREAD	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.
		31:8		Reserved
		41:32		RMID (R/W)
		63:42		Reserved
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		31:10		Reserved
		51:32		COS (R/W)
		63: 52		Reserved
C90H	3216	IA32_L3_QOS_MASK_0	Package	L3 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:19		CBM: Bit vector of available L3 ways for COS 0 enforcement.
		63:20		Reserved
C91H	3217	IA32_L3_QOS_MASK_1	Package	L3 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:19		CBM: Bit vector of available L3 ways for COS 1 enforcement.
		63:20		Reserved
C92H	3218	IA32_L3_QOS_MASK_2	Package	L3 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:19		CBM: Bit vector of available L3 ways for COS 2 enforcement.
		63:20		Reserved
C93H	3219	IA32_L3_QOS_MASK_3	Package	L3 Class Of Service Mask - COS 3 (R/W). If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L3 ways for COS 3 enforcement.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:20		Reserved
C94H	3220	IA32_L3_QOS_MASK_4	Package	L3 Class Of Service Mask - COS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.
		0:19		CBM: Bit vector of available L3 ways for COS 4 enforcement.
		63:20		Reserved
C95H	3221	IA32_L3_QOS_MASK_5	Package	L3 Class Of Service Mask - COS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.
		0:19		CBM: Bit vector of available L3 ways for COS 5 enforcement.
		63:20		Reserved
C96H	3222	IA32_L3_QOS_MASK_6	Package	L3 Class Of Service Mask - COS 6 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6.
		0:19		CBM: Bit vector of available L3 ways for COS 6 enforcement.
		63:20		Reserved
C97H	3223	IA32_L3_QOS_MASK_7	Package	L3 Class Of Service Mask - COS 7 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7.
		0:19		CBM: Bit vector of available L3 ways for COS 7 enforcement.
		63:20		Reserved
C98H	3224	IA32_L3_QOS_MASK_8	Package	L3 Class Of Service Mask - COS 8 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8.
		0:19		CBM: Bit vector of available L3 ways for COS 8 enforcement.
		63:20		Reserved
C99H	3225	IA32_L3_QOS_MASK_9	Package	L3 Class Of Service Mask - COS 9 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9.
		0:19		CBM: Bit vector of available L3 ways for COS 9 enforcement.
		63:20		Reserved
C9AH	3226	IA32_L3_QOS_MASK_10	Package	L3 Class Of Service Mask - COS 10 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10.
		0:19		CBM: Bit vector of available L3 ways for COS 10 enforcement.
		63:20		Reserved
C9BH	3227	IA32_L3_QOS_MASK_11	Package	L3 Class Of Service Mask - COS 11 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		0:19		CBM: Bit vector of available L3 ways for COS 11 enforcement.
		63:20		Reserved
C9CH	3228	IA32_L3_QOS_MASK_12	Package	L3 Class Of Service Mask - COS 12 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=12.
		0:19		CBM: Bit vector of available L3 ways for COS 12 enforcement.
		63:20		Reserved
C9DH	3229	IA32_L3_QOS_MASK_13	Package	L3 Class Of Service Mask - COS 13 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=13.
		0:19		CBM: Bit vector of available L3 ways for COS 13 enforcement.
		63:20		Reserved
C9EH	3230	IA32_L3_QOS_MASK_14	Package	L3 Class Of Service Mask - COS 14 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=14.
		0:19		CBM: Bit vector of available L3 ways for COS 14 enforcement.
		63:20		Reserved
C9FH	3231	IA32_L3_QOS_MASK_15	Package	L3 Class Of Service Mask - COS 15 (R/W) If CPUID.(EAX=10H, ECX=1);EDX.COS_MAX[15:0] >=15.
		0:19		CBM: Bit vector of available L3 ways for COS 15 enforcement.
		63:20		Reserved

2.17.7 MSRs Specific to 3rd Generation Intel® Xeon® Processor Scalable Family based on Ice Lake Microarchitecture

The 3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture (CPUID DisplayFamily_DisplayModel signatures of 06_6AH and 06_6CH) support the MSRs listed in Table 2-51.

Table 2-51. MSRs Supported by 3rd Generation Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel Signatures of 06_6AH and 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
612H	1554	MSR_PACKAGE_ENERGY_TIME_STATUS	Package	Package energy consumed by the entire CPU (R/W)
		31:0		Total amount of energy consumed since last reset.

Table 2-51. MSRs Supported by 3rd Generation Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel Signatures of 06_6AH and 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:32		Total time elapsed when the energy was last updated. This is a monotonic increment counter with auto wrap back to zero after overflow. Unit is 10ns.
618H	1560	MSR_DRAM_POWER_LIMIT	Package	Allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.
		14:0		DRAM_PP_PWR_LIM: Power Limit[0] for DDR domain. Units = Watts, Format = 11.3, Resolution = 0.125W, Range = 0-2047.875W.
		15		PWR_LIM_CTRL_EN: Power Limit[0] enable bit for DDR domain.
		16		Reserved
		23:17		CTRL_TIME_WIN: Power Limit[0] time window Y value, for DDR domain. Actual time_window for RAPL is: (1/1024 seconds) * (1+(x/4)) * (2^y)
		62:24		Reserved
		63		PP_PWR_LIM_LOCK: When set, this entire register becomes read-only. This bit will typically be set by BIOS during boot.
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM Power Parameters (R/W)
		14:0		Spec DRAM Power (DRAM_TDP): The Spec power allowed for DRAM. The TDP setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].
		15		Reserved
		30:16		Minimal DRAM Power (DRAM_MIN_PWR): The minimal power setting allowed for DRAM. Lower values will be clamped to this value. The minimum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].

Table 2-51. MSRs Supported by 3rd Generation Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel Signatures of 06_6AH and 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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 Floating Point number given by 1.x *power(2,y). The unit of measurement is defined in MSR_DRAM_POWER_INFO_UNIT[TIME_UNIT].
		62:55		Reserved
		63		LOCK: Lock bit to lock the register.
981H	2433	IA32_TME_CAPABILITY		See Table 2-2.
982H	2434	IA32_TME_ACTIVATE		See Table 2-2.
983H	2435	IA32_TME_EXCLUDE_MASK		See Table 2-2.
984H	2436	IA32_TME_EXCLUDE_BASE		See Table 2-2.

2.17.8 MSRs Introduced in Future Intel® Xeon® Processors

Table 2-52 lists additional MSRs for future Intel Xeon processors with a CPUID DisplayFamily_DisplayModel signature of 06_8FH. 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 Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CFH	207	IA32_CORE_CAPABILITY	Core	IA32 Core Capabilities Register (R/W) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		0		Reserved: returns zero.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1		Reserved: returns zero.
		2		INTEGRITY_CAPABILITIES When set to 1, the processor supports MSR_INTEGRITY_CAPABILITIES.
		3		RSM_IN_CPLO_ONLY Indicates that RSM will only be allowed in CPLO and will #GP for all non-CPLO privilege levels.
		4		UC_LOCK_DISABLE_SUPPORTED When set to 1, processor supports UC load lock disable feature.
		5		SPLIT_LOCK_DISABLE_SUPPORTED Indicates that there is core support for #AC on split lock detection feature.
		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.
2C2H	706	MSR_COPY_SCAN_HASHES	Die	COPY_SCAN_HASHES (W)
		63:0		SCAN_HASH_ADDR Contains the linear address of the SCAN Test HASH Binary loaded into memory.
2C3H	707	MSR_SCAN_HASHES_STATUS		SCAN_HASHES_STATUS (R/O)
		15:0	Die	CHUNK_SIZE Chunk size of the test in KB.
		23:16	Die	NUM_CHUNKS Total number of chunks.
		31:24		Reserved: all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32	Thread	<p>ERROR_CODE</p> <p>The error-code refers to the LP that runs WRMSR(2C2H).</p> <p>0x0: No error reported.</p> <p>0x1: Attempt to copy scan-hashes when copy already in progress.</p> <p>0x2: Secure Memory not set up correctly.</p> <p>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.</p> <p>0x4: Reserved</p> <p>0x5: Integrity check failed.</p> <p>0x6: Re-install of scan test image attempted when current scan test image is in use by other LPs.</p>
		50:40		Reserved: set to all zeros.
		62:51	Die	<p>MAX_CORE_LIMIT</p> <p>Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1.</p> <p>0 means 1 core at a time.</p>
		63	Die	<p>Valid</p> <p>Valid bit is set when COPY_SCAN_HASHES has completed successfully.</p>
2C4H	708	MSR_AUTHENTICATE_AND_COPY_CHUNK	Die	AUTHENTICATE_AND_COPY_CHUNK (w)
		7:0		<p>CHUNK_INDEX</p> <p>Chunk Index, should be less than the total number of chunks defined by NUM_CHUNKS (MSR_SCAN_HASHES_STATUS[23:16]).</p>
		63:8		<p>CHUNK_ADDR</p> <p>Bits 63:8 of 256B aligned Linear address of scan chunk in memory.</p>
2C5H	709	MSR_CHUNKS_AUTHENTICATION_STATUS		CHUNKS_AUTHENTICATION_STATUS (R/O)
		7:0	Die	<p>VALID_CHUNKS</p> <p>Total number of Valid (authenticated) chunks.</p>
		15:8	Die	<p>TOTAL_CHUNKS</p> <p>Total number of chunks.</p>
		31:16		Reserved: all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32	Thread	<p>ERROR_CODE</p> <p>The error code refers to the LP that runs WRMSR(2C4H).</p> <p>0x0: No error reported.</p> <p>0x1: Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core.</p> <p>0x2: CHUNK authentication error. HASH of chunk did not match expected value.</p>
		63:40		Reserved: set to all zeros.
2C6H	710	MSR_ACTIVATE_SCAN	Thread	ACTIVATE_SCAN (w)
		7:0		<p>CHUNK_START_INDEX</p> <p>Indicates chunk index to start from.</p>
		15:8		<p>CHUNK_STOP_INDEX</p> <p>Indicates what chunk index to stop at (inclusive).</p>
		31:16		Reserved: all zeros.
		62:32		<p>THREAD_WAIT_DELAY</p> <p>TSC based delay to allow threads to rendezvous.</p>
		63		<p>SIGNAL_MCE</p> <p>If 1, then on scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0].</p> <p>If 0, then no logging/no signaling.</p>
2C7H	711	MSR_SCAN_STATUS		SCAN_STATUS (R/O)
		7:0	Core	<p>CHUNK_NUM</p> <p>SCAN Chunk that was reached.</p>
		15:8	Core	<p>CHUNK_STOP_INDEX</p> <p>Indicates what chunk index to stop at (inclusive). Maps to same field in WRMSR(ACTIVATE_SCAN).</p>
		31:16		Reserved: return all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32	Thread	<p>ERROR_CODE</p> <p>0x0: No error.</p> <p>0x1: SCAN operation did not start. Other thread did not join in time.</p> <p>0x2: SCAN operation did not start. Interrupt occurred prior to threads rendezvous.</p> <p>0x3: SCAN operation did not start. Power Management conditions are inadequate to run Intel In-field Scan.</p> <p>0x4: SCAN operation did not start. Non-valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX.</p> <p>0x5: SCAN operation did not start. Mismatch in arguments between threads TO/T1.</p> <p>0x6: SCAN operation did not start. Core not capable of performing SCAN currently.</p> <p>0x8: SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Intel In-field Scan concurrently. MAX_CORE_LIMIT exceeded.</p> <p>0x9: Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed.</p>
		61:40		Reserved: return all zeros.
		62	Core	SCAN_CONTROL_ERROR Scan-System-Controller malfunction.
		63	Core	SCAN_SIGNATURE_ERROR Core failed SCAN-SIGNATURE checking for this chunk.
2C8H	712	MSR_SCAN_MODULE_ID	Module	SCAN_MODULE_ID (R/O)
		31:0		RevID of the currently installed scan test image. Maps to Revision field in external header (offset 4).
		63:32		Reserved: return all zeros.
2C9H	713	MSR_LAST_SAF_WP	Core	LAST_SAF_WP (R/O)
		31:0		<p>LAST_WP</p> <p>Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. Available only if enumerated in MSR_INTEGRITY_CAPABILITIES[10:9].</p>
		63:32		Reserved: return all zeros.
2D9H	729	MSR_INTEGRITY_CAPABILITIES	Module	INTEGRITY_CAPABILITIES (R/O)
		0		<p>STARTUP_SCAN_BIST</p> <p>When set, supports Intel In-field Scan.</p>
		3:1		Reserved: return all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		4		PERIODIC_SCAN_BIST When set, supports Intel In-field Scan.
		23:5		Reserved: return all zeros.
		31:24		ID of the scan programs supported for this part. WRMSR(2C2H) verifies this value against the corresponding value in the scan-image header, i.e., Image_info.
410H	1040	IA32_MC4_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.
411H	1041	IA32_MC4_STATUS	Package	
412H	1042	IA32_MC4_ADDR	Package	
413H	1043	IA32_MC4_MISC	Package	

2.18 MSRS IN INTEL® XEON PHI™ PROCESSOR 3200/5200/7200 SERIES AND INTEL® XEON PHI™ PROCESSOR 7215/7285/7295 SERIES

Intel® Xeon Phi™ processor 3200, 5200, 7200 series, with CPUID DisplayFamily_DisplayModel signature 06_57H, supports the MSR interfaces listed in Table 2-53. These processors are based on the Knights Landing microarchitecture. Intel® Xeon Phi™ processor 7215, 7285, 7295 series, with CPUID DisplayFamily_DisplayModel signature 06_85H, supports the MSR interfaces listed in Table 2-53 and Table 2-54. These processors are based on the Knights Mill microarchitecture. Some MSRs are shared between a pair of processor cores, the scope is marked as module.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination." See Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O)
		63:32		Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Reserved
		2		Enable VMX outside SMX operation (R/WL)
3BH	59	IA32_TSC_ADJUST	THREAD	Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	THREAD	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	THREAD	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	THREAD	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Package	C-State Configuration Control (R/W)
		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		

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		26		C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		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
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Tile	Power Management IO Capture Base (R/W)
		15:0		LVL_2 Base Address (R/W) Microcode will compare IO-read zone to this base address to determine if an MWAIT(C2/3/4) needs to be issued instead of the IO-read. Should be programmed to the chipset Plevel_2 IO address.
		22:16		C-State Range (R/W) The IO-port block size in which IO-redirection will be executed (0-127). Should be programmed based on the number of LVLx registers existing in the chipset.
		63:23		Reserved
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Core	Memory Type Range Register (R) See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		1:0		AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, the AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved
140H	320	MISC_FEATURE_ENABLES	Thread	MISC_FEATURE_ENABLES

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		0		Reserved
		1		User Mode MONITOR and MWAIT (R/W) If set to 1, the MONITOR and MWAIT instructions do not cause invalid-opcode exceptions when executed with CPL > 0 or in virtual-8086 mode. If MWAIT is executed when CPL > 0 or in virtual-8086 mode, and if EAX indicates a C-state other than C0 or C1, the instruction operates as if EAX indicated the C-state C1.
		63:2		Reserved
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	See Table 2-2.
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		31:0		Bank Support (SMM-RO) One bit per MCA bank. If the bit is set, that bank supports Enhanced MCA (Default all 0; does not support EMCA).
		55:32		Reserved
		56		Targeted SMI (SMM-RO) Set if targeted SMI is supported.
		57		SMM_CPU_SVRSTR (SMM-RO) Set if SMM SRAM save/restore feature is supported.
		58		SMM_CODE_ACCESS_CHK (SMM-RO) Set if SMM code access check feature is supported.
		59		Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
186H	390	IA32_PERFEVTSELO	Thread	Performance Monitoring Event Select Register (R/W) See Table 2-2.
		7:0		Event Select
		15:8		UMask
		16		USR
		17		OS
		18		Edge

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		19		PC
		20		INT
		21		AnyThread
		22		EN
		23		INV
		31:24		CMASK
		63:32		Reserved
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Module	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Module	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (R/O)
		1		Thermal Status Log (R/WCO)
		2		PROTCHOT # or FORCEPR# Status (R/O)
		3		PROTCHOT # or FORCEPR# Log (R/WCO)
		4		Critical Temperature Status (R/O)
		5		Critical Temperature Status Log (R/WCO)
		6		Thermal Threshold #1 Status (R/O)
		7		Thermal Threshold #1 Log (R/WCO)
		8		Thermal Threshold #2 Status (R/O)
		9		Thermal Threshold #2 Log (R/WCO)
		10		Power Limitation Status (R/O)
		11		Power Limitation Log (R/WCO)
		15:12		Reserved
		22:16		Digital Readout (R/O)
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (R/O)
31		Reading Valid (R/O)		
63:32		Reserved		
1A0H	416	IA32_MISC_ENABLE	Thread	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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)
		37:35		Reserved
		38		Turbo Mode Disable (R/W)
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R)
		29:24		Target Offset (R/W)
		63:30		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher.
		1	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher.
		63:2		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Shared	Offcore Response Event Select Register (R/W)
1A7H	423	MSR_OFFCORE_RSP_1	Shared	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode for Groups of Cores (R/W)
		0		Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		7:1	Package	Maximum Number of Cores in Group 0 Number active processor cores which operates under the maximum ratio limit for group 0.
		15:8	Package	Maximum Ratio Limit for Group 0 Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.
		20:16	Package	Number of Incremental Cores Added to Group 1 Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".
		23:21	Package	Group Ratio Delta for Group 1 An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.
		28:24	Package	Number of Incremental Cores Added to Group 2 Group 2, which includes the specified number of additional cores plus all the cores in group 1, operates under the group 2 turbo max ratio limit = "group 1 Max ratio limit" - "group ratio delta for group 2".
		31:29	Package	Group Ratio Delta for Group 2 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.
		36:32	Package	Number of Incremental Cores Added to Group 3 Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".
		39:37	Package	Group Ratio Delta for Group 3 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.
		44:40	Package	Number of Incremental Cores Added to Group 4 Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".
		47:45	Package	Group Ratio Delta for Group 4 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.
		52:48	Package	Number of Incremental Cores Added to Group 5 Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		55:53	Package	Group Ratio Delta for Group 5 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.
		60:56	Package	Number of Incremental Cores Added to Group 6 Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".
		63:61	Package	Group Ratio Delta for Group 6 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.
1B0H	432	IA32_ENERGY_PERF_BIAS	Thread	See Table 2-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		0		CPL_EQ_0
		1		CPL_NEQ_0
		2		JCC
		3		NEAR_REL_CALL
		4		NEAR_IND_CALL
		5		NEAR_RET
		6		NEAR_IND_JMP
		7		NEAR_REL_JMP
		8		FAR_BRANCH
	63:9			Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W)
		0		LBR Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.
		1		BTF Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.
	5:2			Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record from Linear IP (R)
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record to Linear IP (R)
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Package	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2.
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Table 2-2.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C3 Residency Counter (R/O)

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Package C6 Residency Counter (R/O)
		63:0		
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Package C7 Residency Counter (R/O)
		63:0		
3FCH	1020	MSR_MC0_RESIDENCY	Module	Note: C-state values are processor specific C-state code names, unrelated to MWait extension C-state parameters or ACPI C-states. Module C0 Residency Counter (R/O)
		63:0		
3FDH	1021	MSR_MC6_RESIDENCY	Module	Module C6 Residency Counter (R/O)
		63:0		
3FFH	1023	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWait extension C-state parameters or ACPI C-states. CORE C6 Residency Counter (R/O)
		63:0		
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C2 Residency Counter (R/O)
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 14.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W) See Section 14.10.3, "Package RAPL Domain."
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
638H	1592	MSR_PPO_POWER_LIMIT	Package	PPO RAPL Power Limit Control (R/W) See Section 14.10.4, "PPO/PP1 RAPL Domains."
639H	1593	MSR_PPO_ENERGY_STATUS	Package	PPO Energy Status (R/O) See Section 14.10.4, "PPO/PP1 RAPL Domains."

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O) See Table 2-25.
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O) See Table 2-25.
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O) See Table 2-25.
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W) See Table 2-25.
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W) See Table 2-25.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		0		PROCHOT Status (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
6E0H	1760	IA32_TSC_DEADLINE	Core	TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID Register (R/O)
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version Register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority Register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority Register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI Register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination Register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector Register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service Register Bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service Register Bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service Register Bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service Register Bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service Register Bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service Register Bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service Register Bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service Register Bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMRO	Thread	x2APIC Trigger Mode Register Bits [31:0] (R/O)

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode Register Bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode Register Bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode Register Bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode Register Bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode Register Bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode Register Bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode Register Bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request Register Bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request Register Bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request Register Bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request Register Bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request Register Bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request Register Bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request Register Bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request Register Bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status Register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command Register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt Register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt Register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor Register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 Register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 Register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error Register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count Register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count Register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration Register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI Register (W/O)
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2

Table 2-54 lists model-specific registers that are supported by Intel® Xeon Phi™ processor 7215, 7285, 7295 series based on the Knights Mill microarchitecture.

Table 2-54. Additional MSRs Supported by Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with DisplayFamily_DisplayModel Signature 06_85H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
9BH	155	IA32_SMM_MONITOR_CTL	Core	SMM Monitor Configuration (R/W) This MSR is readable only if VMX is enabled, and writeable only if VMX is enabled and in SMM mode, and is used to configure the VMX MSEG base address. See Table 2-2.
480H	1152	IA32_VMX_BASIC	Core	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.
481H	1153	IA32_VMX_PINBASED_CTL	Core	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.
482H	1154	IA32_VMX_PROCBASED_CTL	Core	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)
483H	1155	IA32_VMX_EXIT_CTL	Core	Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.
484H	1156	IA32_VMX_ENTRY_CTL	Core	Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.
485H	1157	IA32_VMX_MISC	Core	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.
486H	1158	IA32_VMX_CRO_FIXED0	Core	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Table 2-2.
487H	1159	IA32_VMX_CRO_FIXED1	Core	Capability Reporting Register of CRO Bits Fixed to 1 (R/O) See Table 2-2.

Table 2-54. Additional MSRs Supported by Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with DisplayFamily_DisplayModel Signature 06_85H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
488H	1160	IA32_VMX_CR4_FIXED0	Core	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	Core	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.
48AH	1162	IA32_VMX_VMCS_ENUM	Core	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.
48BH	1163	IA32_VMX_PROCBASED_CTL2	Core	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Table 2-2.
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.
48DH	1165	IA32_VMX_TRUE_PINBASED_CTL2	Core	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTL2	Core	Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2.
48FH	1167	IA32_VMX_TRUE_EXIT_CTL2	Core	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.
490H	1168	IA32_VMX_TRUE_ENTRY_CTL2	Core	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.
491H	1169	IA32_VMX_FMFUNC	Core	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.

2.19 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-55 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table 2-1.

- MSRs with an "IA32_" prefix are designated as "architectural." This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an "MSR_" prefix are model specific with respect to address functionalities. The column "Model Availability" lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See "CPUID—CPU Identification" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_LINE_SIZE	3, 4, 6	Shared	See Section 8.10.5, "Monitor/Mwait Address Range Determination."
10H	16	IA32_TIME_STAMP_COUNTER	0, 1, 2, 3, 4, 6	Unique	Time Stamp Counter See Table 2-2.
					On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3, 4, 6	Shared	Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	0, 1, 2, 3, 4, 6	Unique	APIC Location and Status (R/W) See Table 2-2. See Section 10.4.4, "Local APIC Status and Location."
2AH	42	MSR_EBC_HARD_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.
		0			Output Tri-state Enabled (R) Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		1			Execute BIST (R) Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		2			In Order Queue Depth (R) Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		3			MCERR# Observation Disabled (R) Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		4			BINIT# Observation Enabled (R) Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		6:5			APIC Cluster ID (R) Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		7			Bus Park Disable (R) Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		11:8			Reserved
		13:12			Agent ID (R) Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		63:14			Reserved
2BH	43	MSR_EBC_SOFT_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Soft Power-On Configuration (R/W) Enables and disables processor features.
		0			RCNT/SCNT On Request Encoding Enable (R/W) Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).
		1			Data Error Checking Disable (R/W) Set to disable system data bus parity checking; clear to enable parity checking.
		2			Response Error Checking Disable (R/W) Set to disable (default); clear to enable.
		3			Address/Request Error Checking Disable (R/W) Set to disable (default); clear to enable.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		4			Initiator MCERR# Disable (R/W) Set to disable MCERR# driving for initiator bus requests (default); clear to enable.
		5			Internal MCERR# Disable (R/W) Set to disable MCERR# driving for initiator internal errors (default); clear to enable.
		6			BINIT# Driver Disable (R/W) Set to disable BINIT# driver (default); clear to enable driver.
		63:7			Reserved
2CH	44	MSR_EBC_FREQUENCY_ID	2,3, 4, 6	Shared	Processor Frequency Configuration The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2. (R) The field Indicates the current processor frequency configuration.
		15:0			Reserved
		18:16			Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6)
					133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
					266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.
		23:19			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
		31:24			Core Clock Frequency to System Bus Frequency Ratio (R) The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.
		63:25			Reserved
2CH	44	MSR_EBC_FREQUENCY_ID	0, 1	Shared	Processor Frequency Configuration (R) The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.
		20:0			Reserved
		23:21			Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz All others values reserved.
		63:24			Reserved
3AH	58	IA32_FEATURE_CONTROL	3, 4, 6	Unique	Control Features in IA-32 Processor (R/W) See Table 2-2. (If CPUID.01H:ECX.[bit 5])
79H	121	IA32_BIOS_UPDT_TRIG	0, 1, 2, 3, 4, 6	Shared	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
9BH	155	IA32_SMM_MONITOR_CTL	3, 4, 6	Unique	SMM Monitor Configuration (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	0, 1, 2, 3, 4, 6	Unique	MTRR Information See Section 11.11.1, "MTRR Feature Identification."
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3, 4, 6	Unique	CS Register Target for CPL 0 Code (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3, 4, 6	Unique	Stack Pointer for CPL 0 Stack (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	CPL 0 Code Entry Point (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
179H	377	IA32_MCG_CAP	0, 1, 2, 3, 4, 6	Unique	Machine Check Capabilities (R) See Table 2-2. See Section 15.3.1.1, "IA32_MCG_CAP MSR."
17AH	378	IA32_MCG_STATUS	0, 1, 2, 3, 4, 6	Unique	Machine Check Status (R) See Table 2-2. See Section 15.3.1.2, "IA32_MCG_STATUS MSR."
17BH	379	IA32_MCG_CTL			Machine Check Feature Enable (R/W) See Table 2-2. See Section 15.3.1.3, "IA32_MCG_CTL MSR."
180H	384	MSR_MCG_RAX	0, 1, 2, 3, 4, 6	Unique	Machine Check EAX/RAX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
181H	385	MSR_MCG_RBX	0, 1, 2, 3, 4, 6	Unique	Machine Check EBX/RBX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
182H	386	MSR_MCG_RCX	0, 1, 2, 3, 4, 6	Unique	Machine Check ECX/RCX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
183H	387	MSR_MCG_RDX	0, 1, 2, 3, 4, 6	Unique	Machine Check EDX/RDX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
184H	388	MSR_MCG_RSI	0, 1, 2, 3, 4, 6	Unique	Machine Check ESI/RSI Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
185H	389	MSR_MCG_RDI	0, 1, 2, 3, 4, 6	Unique	Machine Check EDI/RDI Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
186H	390	MSR_MCG_RBP	0, 1, 2, 3, 4, 6	Unique	Machine Check EBP/RBP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
187H	391	MSR_MCG_RSP	0, 1, 2, 3, 4, 6	Unique	Machine Check ESP/RSP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
188H	392	MSR_MCG_RFLAGS	0, 1, 2, 3, 4, 6	Unique	Machine Check EFLAGS/RFLAG Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
189H	393	MSR_MCG_RIP	0, 1, 2, 3, 4, 6	Unique	Machine Check EIP/RIP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
18AH	394	MSR_MCG_MISC	0, 1, 2, 3, 4, 6	Unique	Machine Check Miscellaneous See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		0			DS When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.
		63:1			Reserved
18BH - 18FH	395 - 399	MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
190H	400	MSR_MCG_R8	0, 1, 2, 3, 4, 6	Unique	Machine Check R8 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
191H	401	MSR_MCG_R9	0, 1, 2, 3, 4, 6	Unique	Machine Check R9D/R9 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
192H	402	MSR_MCG_R10	0, 1, 2, 3, 4, 6	Unique	Machine Check R10 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
193H	403	MSR_MCG_R11	0, 1, 2, 3, 4, 6	Unique	Machine Check R11 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
194H	404	MSR_MCG_R12	0, 1, 2, 3, 4, 6	Unique	Machine Check R12 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
195H	405	MSR_MCG_R13	0, 1, 2, 3, 4, 6	Unique	Machine Check R13 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
196H	406	MSR_MCG_R14	0, 1, 2, 3, 4, 6	Unique	Machine Check R14 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
197H	407	MSR_MCG_R15	0, 1, 2, 3, 4, 6	Unique	Machine Check R15 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	See Table 2-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	See Table 2-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
19AH	410	IA32_CLOCK_MODULATION	0, 1, 2, 3, 4, 6	Unique	Thermal Monitor Control (R/W) See Table 2-2. See Section 14.8.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3, 4, 6	Unique	Thermal Interrupt Control (R/W) See Section 14.8.2, "Thermal Monitor," and see Table 2-2.
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	Thermal Monitor Status (R/W) See Section 14.8.2, "Thermal Monitor," and see Table 2-2.
19DH	413	MSR_THERM2_CTL			Thermal Monitor 2 Control
			3,	Shared	For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.
			4, 6	Shared	For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3, 4, 6	Shared	Enable Miscellaneous Processor Features (R/W)

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		0			Fast-Strings Enable. See Table 2-2.
		1			Reserved
		2			x87 FPU Fopcode Compatibility Mode Enable
		3			Thermal Monitor 1 Enable See Section 14.8.2, "Thermal Monitor," and see 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 11.5.4, "Disabling and Enabling the L3 Cache."
		7			Performance Monitoring Available (R) See Table 2-2.
		8			Suppress Lock Enable When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.
		9			Prefetch Queue Disable When set, disables the prefetch queue. When clear (default), enables the prefetch queue.
		10			FERR# Interrupt Reporting Enable (R/W) When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
					When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt.
					This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted.
		11			Branch Trace Storage Unavailable (BTS_UNAVILABLE) (R) See Table 2-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.
		12			PEBS_UNAVILABLE: Processor Event Based Sampling Unavailable (R) See Table 2-2. When set, the processor does not support processor event-based sampling (PEBS); when clear, PEBS is supported.
		13	3		TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		17:14			Reserved
		18	3, 4, 6		ENABLE MONITOR FSM (R/W) See Table 2-2.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/Unique ¹	Bit Description
Hex	Dec				
		19			Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line of the 128-byte sector containing currently required data. When set to 0, the processor fetches both cache lines in the sector.
					Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.
		21:20			Reserved
		22	3, 4, 6		Limit CPUID MAXVAL (R/W) See Table 2-2. Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.
		23		Shared	xTPR Message Disable (R/W) See Table 2-2.
		24			L1 Data Cache Context Mode (R/W) When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 11.5.6, "L1 Data Cache Context Mode." When L1 is running in adaptive mode and CR3s are identical, data in L1 is shared across logical processors. Otherwise, L1 is not shared and cache use is competitive. If the Context ID feature flag (ECX[10]) is set to 0 after executing CPUID with EAX = 1, the ability to switch modes is not supported. BIOS must not alter the contents of IA32_MISC_ENABLE[24].
		33:25			Reserved
		34		Unique	XD Bit Disable (R/W) See Table 2-2.
		63:35			Reserved
		1A1H	417	MSR_PLATFORM_BRV	3, 4, 6
		17:0			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		18			PLATFORM Requirements When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.
		63:19			Reserved
1D7H	471	MSR_LER_FROM_LIP	0, 1, 2, 3, 4, 6	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.13.3, "Last Exception Records."
		31:0			From Linear IP Linear address of the last branch instruction.
		63:32			Reserved
1D7H	471	63:0		Unique	From Linear IP Linear address of the last branch instruction (If IA-32e mode is active).
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3, 4, 6	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.13.3, "Last Exception Records."
		31:0			From Linear IP Linear address of the target of the last branch instruction.
		63:32			Reserved
1D8H	472	63:0		Unique	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).
1D9H	473	MSR_DEBUGCTLA	0, 1, 2, 3, 4, 6	Unique	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.13.1, "MSR_DEBUGCTLA MSR."
1DAH	474	MSR_LASTBRANCH_TOS	0, 1, 2, 3, 4, 6	Unique	Last Branch Record Stack TOS (R/O) Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 17.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture"; and addresses 1DBH-1DEH and 680H-68FH.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2	Unique	Last Branch Record 0 (R/O) One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH.
					See Section 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
1DCH	477	MSR_LASTBRANCH_1	0, 1, 2	Unique	Last Branch Record 1 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2	Unique	Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2	Unique	Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
200H	512	IA32_MTRR_PHYSBASE0	0, 1, 2, 3, 4, 6	Shared	Variable Range Base MTRR See Section 11.11.2.3, "Variable Range MTRRs."
201H	513	IA32_MTRR_PHYSMASK0	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
202H	514	IA32_MTRR_PHYSBASE1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
203H	515	IA32_MTRR_PHYSMASK1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
204H	516	IA32_MTRR_PHYSBASE2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
205H	517	IA32_MTRR_PHYSMASK2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
206H	518	IA32_MTRR_PHYSBASE3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
207H	519	IA32_MTRR_PHYSMASK3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
208H	520	IA32_MTRR_PHYSBASE4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
209H	521	IA32_MTRR_PHYSMASK4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
20AH	522	IA32_MTRR_PHYSBASE5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20BH	523	IA32_MTRR_PHYSMASK5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20CH	524	IA32_MTRR_PHYSBASE6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20DH	525	IA32_MTRR_PHYSMASK6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20EH	526	IA32_MTRR_PHYSBASE7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20FH	527	IA32_MTRR_PHYSMASK7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
250H	592	IA32_MTRR_FIX64K_00000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
258H	600	IA32_MTRR_FIX16K_80000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
259H	601	IA32_MTRR_FIX16K_A0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
268H	616	IA32_MTRR_FIX4K_C0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
269H	617	IA32_MTRR_FIX4K_C8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26AH	618	IA32_MTRR_FIX4K_D0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26BH	619	IA32_MTRR_FIX4K_D8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26CH	620	IA32_MTRR_FIX4K_E0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26DH	621	IA32_MTRR_FIX4K_E8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26EH	622	IA32_MTRR_FIX4K_F0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26FH	623	IA32_MTRR_FIX4K_F8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
277H	631	IA32_PAT	0, 1, 2, 3, 4, 6	Unique	Page Attribute Table See Section 11.11.2.2, "Fixed Range MTRRs."
2FFH	767	IA32_MTRR_DEF_TYPE	0, 1, 2, 3, 4, 6	Shared	Default Memory Types (R/W) See Table 2-2. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
308H	776	MSR_FLAME_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
309H	777	MSR_FLAME_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30AH	778	MSR_FLAME_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30BH	779	MSR_FLAME_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
368H	872	MSR_FLAME_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3AEH	942	MSR_SAA_T_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3AFH	943	MSR_SAA_T_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 19.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	Shared	See Section 19.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3COH	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3F0H	1008	MSR_TC_PRECISE_EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	0, 1, 2, 3, 4, 6	Shared	Processor Event Based Sampling (PEBS) (R/W) Controls the enabling of processor event sampling and replay tagging.
		12:0			See https://perfmon-events.intel.com/ .
		23:13			Reserved
		24			UOP Tag Enables replay tagging when set.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
		25			ENABLE_PEBS_MY_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 19.6.4.3, "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 19.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Intel Hyper-Threading Technology.
		63:27			Reserved
3F2H	1010	MSR_PEBS_MATRIX_VERT	0, 1, 2, 3, 4, 6	Shared	See https://perfmon-events.intel.com/ .
400H	1024	IA32_MCO_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MCO_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MCO_MISC MSR is either not implemented or does not contain additional information if the MISC_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
406H	1030	IA32_MC1_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	IA32_MC1_MISC		Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISC_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40BH	1035	IA32_MC2_MISC			See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISC_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
40FH	1039	IA32_MC3_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRIV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC4_MISC			See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	3, 4, 6	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTL	3, 4, 6	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTL	3, 4, 6	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and see Table 2-2.
483H	1155	IA32_VMX_EXIT_CTL	3, 4, 6	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and see Table 2-2.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique ¹	Bit Description
Hex	Dec				
484H	1156	IA32_VMX_ENTRY_CTL5	3, 4, 6	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and see Table 2-2.
485H	1157	IA32_VMX_MISC	3, 4, 6	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and see Table 2-2.
486H	1158	IA32_VMX_CR0_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 2-2.
487H	1159	IA32_VMX_CR0_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 2-2.
488H	1160	IA32_VMX_CR4_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 2-2.
48AH	1162	IA32_VMX_VMCS_ENUM	3, 4, 6	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration," and see Table 2-2.
48BH	1163	IA32_VMX_PROCBASED_CTL52	3, 4, 6	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and see Table 2-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	3, 4, 6	Unique	Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
					The MSRs at 680H-68FH, 6C0H-6CfH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH, which performed the same function for early releases. See Section 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
681H	1665	MSR_LASTBRANCH_1_FROM_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	3, 4, 6	Unique	Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6C0H.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6C0H.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6C0H.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6C0H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique ¹	Bit Description
Hex	Dec				
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6COH.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6COH.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6COH.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6COH.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6COH.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6COH.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6COH.
C000_ 0080H		IA32_EFER	3, 4, 6	Unique	Extended Feature Enables See Table 2-2.
C000_ 0081H		IA32_STAR	3, 4, 6	Unique	System Call Target Address (R/W) See Table 2-2.
C000_ 0082H		IA32_LSTAR	3, 4, 6	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_ 0084H		IA32_FMASK	3, 4, 6	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_ 0100H		IA32_FS_BASE	3, 4, 6	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_ 0101H		IA32_GS_BASE	3, 4, 6	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_ 0102H		IA32_KERNEL_GS_BASE	3, 4, 6	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

NOTES

1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.

2.19.1 MSRs Unique to Intel® Xeon® Processor MP with L3 Cache

The MSRs listed in Table 2-56 apply to Intel® Xeon® Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (see CPUID instruction for more details).

Table 2-56. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache

Register Address	Register Name Fields and Flags	Model Availability	Shared/Unique	Bit Description
107CCH	MSR_IFSB_BUSQ0	3, 4	Shared	IFSB BUSQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CDH	MSR_IFSB_BUSQ1	3, 4	Shared	IFSB BUSQ Event Control and Counter Register (R/W)
107CEH	MSR_IFSB_SNPQ0	3, 4	Shared	IFSB SNPQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CFH	MSR_IFSB_SNPQ1	3, 4	Shared	IFSB SNPQ Event Control and Counter Register (R/W)
107D0H	MSR_EFSB_DRDY0	3, 4	Shared	EFSB DRDY Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D1H	MSR_EFSB_DRDY1	3, 4	Shared	EFSB DRDY Event Control and Counter Register (R/W)
107D2H	MSR_IFSB_CTL6	3, 4	Shared	IFSB Latency Event Control Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D3H	MSR_IFSB_CNTR7	3, 4	Shared	IFSB Latency Event Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."

The MSRs listed in Table 2-57 apply to Intel® Xeon® Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table 2-57 are shared between logical processors in the same core, but are replicated for each core.

Table 2-57. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address	Register Name Fields and Flags	Model Availability	Shared/Unique	Bit Description
107CCH	MSR_EMON_L3_CTR_CTL0	6	Shared	GBUSQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."

Table 2-57. MSRs Unique to Intel® Xeon® Processor 7100 Series (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique	Bit Description
107CDH		MSR_EMON_L3_CTR_CTL1	6	Shared	GBUSQ Event Control and Counter Register (R/W)
107CEH		MSR_EMON_L3_CTR_CTL2	6	Shared	GSNPQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CFH		MSR_EMON_L3_CTR_CTL3	6	Shared	GSNPQ Event Control and Counter Register (R/W)
107D0H		MSR_EMON_L3_CTR_CTL4	6	Shared	FSB Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D1H		MSR_EMON_L3_CTR_CTL5	6	Shared	FSB Event Control and Counter Register (R/W)
107D2H		MSR_EMON_L3_CTR_CTL6	6	Shared	FSB Event Control and Counter Register (R/W)
107D3H		MSR_EMON_L3_CTR_CTL7	6	Shared	FSB Event Control and Counter Register (R/W)

2.20 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table 2-58. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	P5_MC_ADDR	Unique	See Section 2.23, "MSRs in Pentium Processors," and see Table 2-2.
1H	1	P5_MC_TYPE	Unique	See Section 2.23, "MSRs in Pentium Processors," and see Table 2-2.
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and see Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location," and see Table 2-2.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
2AH	42	MSR_EBL_CR_POWERON	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		6: 5		Reserved
		7		BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled
		9		Execute BIST (R/O) 1 = Enabled; 0 = Disabled
		10		MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled
		11		Reserved
		12		BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled
		13		Reserved
		14		1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		APIC Cluster ID (R/O)
		18		System Bus Frequency (R/O) 0 = 100 MHz 1 = Reserved
		19		Reserved
		21: 20		Symmetric Arbitration ID (R/O)
26:22	Clock Frequency Ratio (R/O)			

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in IA-32 Processor (R/W) See Table 2-2.
40H	64	MSR_LASTBRANCH_0	Unique	Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: <ul style="list-style-type: none"> Last Branch Record Stack TOS at 1C9H Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_LASTBRANCH_1	Unique	Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	Unique	Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	Unique	Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	Unique	Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	Unique	Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	Unique	Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	Unique	Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
C1H	193	IA32_PMC0	Unique	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (R/O) This field indicates the scaleable bus clock speed:
		2:0		<ul style="list-style-type: none"> 101B: 100 MHz (FSB 400) 001B: 133 MHz (FSB 533) 011B: 167 MHz (FSB 667) 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

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (R/W) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved
		8		L2 Enabled (R/W) 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved
		23		L2 Not Present (R/O) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		0		RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1		EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2		MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved
186H	390	IA32_PERFEVTSELO	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor".
19DH	413	MSR_THERM2_CTL	Unique	Thermal Monitor 2 Control
		15:0		Reserved
		16		TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		2:0		Reserved
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		9:8		Reserved
		10	Shared	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	Branch Trace Storage Unavailable (R/O) See Table 2-2.
		12		Reserved
		13	Shared	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19		Reserved
		22	Shared	Limit CPUID Maxval (R/W) See Table 2-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 2.
		33:23		Reserved
		34	Shared	XD Bit Disable (R/W) See Table 2-2.
		63:35		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	MTRRphysBase0	Unique	Memory Type Range Registers
201H	513	MTRRphysMask0	Unique	Memory Type Range Registers
202H	514	MTRRphysBase1	Unique	Memory Type Range Registers
203H	515	MTRRphysMask1	Unique	Memory Type Range Registers
204H	516	MTRRphysBase2	Unique	Memory Type Range Registers
205H	517	MTRRphysMask2	Unique	Memory Type Range Registers
206H	518	MTRRphysBase3	Unique	Memory Type Range Registers
207H	519	MTRRphysMask3	Unique	Memory Type Range Registers
208H	520	MTRRphysBase4	Unique	Memory Type Range Registers
209H	521	MTRRphysMask4	Unique	Memory Type Range Registers
20AH	522	MTRRphysBase5	Unique	Memory Type Range Registers
20BH	523	MTRRphysMask5	Unique	Memory Type Range Registers
20CH	524	MTRRphysBase6	Unique	Memory Type Range Registers
20DH	525	MTRRphysMask6	Unique	Memory Type Range Registers
20EH	526	MTRRphysBase7	Unique	Memory Type Range Registers
20FH	527	MTRRphysMask7	Unique	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Unique	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Unique	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Unique	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Unique	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Unique	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Unique	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Unique	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Unique	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Unique	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Unique	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Unique	Memory Type Range Registers

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	Default Memory Types (R/W) See Table 2-2. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MCO_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
402H	1026	IA32_MCO_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
406H	1030	IA32_MC1_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	MSR_MC4_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	MSR_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
413H	1043	MSR_MC3_MISC	Unique	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISCV flag in the IA32_MCI_STATUS register is set.
414H	1044	MSR_MC5_CTL	Unique	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
415H	1045	MSR_MC5_STATUS	Unique	Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
416H	1046	MSR_MC5_ADDR	Unique	Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDRV flag in the IA32_MCI_STATUS register is set.
417H	1047	MSR_MC5_MISC	Unique	Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISCV flag in the IA32_MCI_STATUS register is set.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information". (If CPUID.01H:ECX.[bit 5])
481H	1153	IA32_VMX_PINBASED_CTL	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls". (If CPUID.01H:ECX.[bit 5])
482H	1154	IA32_VMX_PROCBASED_CTL	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls". (If CPUID.01H:ECX.[bit 5])
483H	1155	IA32_VMX_EXIT_CTL	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls". (If CPUID.01H:ECX.[bit 5])
484H	1156	IA32_VMX_ENTRY_CTL	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls". (If CPUID.01H:ECX.[bit 5])
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data". (If CPUID.01H:ECX.[bit 5])
486H	1158	IA32_VMX_CRO_FIXED0	Unique	Capability Reporting Register of CRO Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CRO". (If CPUID.01H:ECX.[bit 5])

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_FIXED1	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0". (If CPUID.01H:ECX.[bit 5])
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4". (If CPUID.01H:ECX.[bit 5])
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4". (If CPUID.01H:ECX.[bit 5])
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration". (If CPUID.01H:ECX.[bit 5])
48BH	1163	IA32_VMX_PROCBASED_CTL2	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls". (If CPUID.01H:ECX.[bit 5] and IA32_VMX_PROCBASED_CTL2[bit 63])
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
		31:0		DS Buffer Management Area Linear address of the first byte of the DS buffer management area.
		63:32		Reserved
C000_0080H		IA32_EFER	Unique	See Table 2-2.
		10:0		Reserved
		11		Execute Disable Bit Enable
		63:12		Reserved

2.21 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 2.22 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table 2-59. MSRs in Pentium M Processors

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.23, "MSRs in Pentium Processors."

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
1H	1	P5_MC_TYPE	See Section 2.23, "MSRs in Pentium Processors."
10H	16	IA32_TIME_STAMP_COUNTER	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
2AH	42	MSR_EBL_CR_POWERON	Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.
		0	Reserved
		1	Data Error Checking Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		2	Response Error Checking Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		3	MCERR# Drive Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		4	Address Parity Enable (R) 0 = Disabled Always 0 on the Pentium M processor.
		6:5	Reserved
		7	BINIT# Driver Enable (R) 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled
		9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled
		10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		11	Reserved
		12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
13	Reserved		

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes Always 0 on the Pentium M processor.
		15	Reserved
		17:16	APIC Cluster ID (R/O) Always 00B on the Pentium M processor.
		18	System Bus Frequency (R/O) 0 = 100 MHz 1 = Reserved Always 0 on the Pentium M processor.
		19	Reserved
		21:20	Symmetric Arbitration ID (R/O) Always 00B on the Pentium M processor.
		26:22	Clock Frequency Ratio (R/O)
40H	64	MSR_LASTBRANCH_0	Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: <ul style="list-style-type: none"> ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)".
41H	65	MSR_LASTBRANCH_1	Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.
119H	281	MSR_BBL_CR_CTL	Control Register Used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.
		63:0	Reserved
11EH	281	MSR_BBL_CR_CTL3	Control register 3 Used to configure the L2 Cache.

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
		4:1	Reserved
		5	ECC Check Enable (R/O) This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default) 1 = Enabled For the Pentium M processor, ECC checking on the cache data bus is always enabled.
		7:6	Reserved
		8	L2 Enabled (R/W) 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9	Reserved
		23	L2 Not Present (R/O) 0 = L2 Present 1 = L2 Not Present
		63:24	Reserved
179H	377	IA32_MCG_CAP	Read-only register that provides information about the machine-check architecture of the processor.
		7:0	Count (R/O) Indicates the number of hardware unit error reporting banks available in the processor.
		8	IA32_MCG_CTL Present (R/O) 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
		63:9	Reserved
17AH	378	IA32_MCG_STATUS	Global Machine Check Status
		0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3	Reserved
198H	408	IA32_PERF_STATUS	See Table 2-2.
199H	409	IA32_PERF_CTL	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation (R/W). See Table 2-2. See Section 14.8.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	Thermal Monitor Status (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor."
19DH	413	MSR_THERM2_CTL	Thermal Monitor 2 Control
		15:0	Reserved
		16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16	Reserved
1A0H	416	IA32_MISC_ENABLE	Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		2:0	Reserved

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
		6:4	Reserved
		7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.
		9:8	Reserved
		10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
			Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
		12	Processor Event Based Sampling Unavailable (R/O) 1 = Processor does not support processor event based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
		15:13	Reserved
		16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
		22:17	Reserved
		23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
		63:24	Reserved

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
1C9H	457	MSR_LASTBRANCH_TOS	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: <ul style="list-style-type: none"> MSR_LASTBRANCH_0_FROM_IP (at 40H). Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)".
1D9H	473	MSR_DEBUGCTLB	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
1DDH	477	MSR_LER_TO_LIP	Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.16.2, "Last Branch and Last Exception MSRs."
1DEH	478	MSR_LER_FROM_LIP	Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.16.2, "Last Branch and Last Exception MSRs."
2FFH	767	IA32_MTRR_DEF_TYPE	Default Memory Types (R/W) Sets the memory type for the regions of physical memory that are not mapped by the MTRRs. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MCO_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MCO_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MCO_ADDR	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs". The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs". The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	See Chapter 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
40AH	1034	IA32_MC2_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
40EH	1038	MSR_MC4_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRs."
412H	1042	MSR_MC3_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
600H	1536	IA32_DS_AREA	DS Save Area (R/W) See Table 2-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
		31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.
		63:32	Reserved

2.22 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table 2-2 for a list of the architectural MSRs.

Table 2-60. MSRs in the P6 Family Processors

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.23, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Section 2.23, "MSRs in Pentium Processors."
10H	16	TSC	See Section 17.17, "Time-Stamp Counter."

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
17H	23	IA32_PLATFORM_ID	Platform ID (R) The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.
		49:0	Reserved
		52:50	Platform Id (R) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
		56:53	L2 Cache Latency Read.
		59:57	Reserved
		60	Clock Frequency Ratio Read.
		63:61	Reserved
		1BH	27
7:0	Reserved		
8	Boot Strap Processor Indicator Bit 1 = BSP		
10:9	Reserved		
11	APIC Global Enable Bit - Permanent till reset 1 = Enabled 0 = Disabled		
31:12	APIC Base Address.		
63:32	Reserved		
2AH	42	EBL_CR_POWERON	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0	Reserved ¹
		1	Data Error Checking Enable (R/W) 1 = Enabled 0 = Disabled
		2	Response Error Checking Enable FRCERR Observation Enable (R/W) 1 = Enabled 0 = Disabled
		3	AERR# Drive Enable (R/W) 1 = Enabled 0 = Disabled

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		4	BERR# Enable for Initiator Bus Requests (R/W) 1 = Enabled 0 = Disabled
		5	Reserved
		6	BERR# Driver Enable for Initiator Internal Errors (R/W) 1 = Enabled 0 = Disabled
		7	BINIT# Driver Enable (R/W) 1 = Enabled 0 = Disabled
		8	Output Tri-state Enabled (R) 1 = Enabled 0 = Disabled
		9	Execute BIST (R) 1 = Enabled 0 = Disabled
		10	AERR# Observation Enabled (R) 1 = Enabled 0 = Disabled
		11	Reserved
		12	BINIT# Observation Enabled (R) 1 = Enabled 0 = Disabled
		13	In Order Queue Depth (R) 1 = 1 0 = 8
		14	1-MByte Power on Reset Vector (R) 1 = 1MByte 0 = 4GBytes
		15	FRC Mode Enable (R) 1 = Enabled 0 = Disabled
		17:16	APIC Cluster ID (R)
		19:18	System Bus Frequency (R) 00 = 66MHz 10 = 100MHz 01 = 133MHz 11 = Reserved
		21:20	Symmetric Arbitration ID (R)
		25:22	Clock Frequency Ratio (R)
		26	Low Power Mode Enable (R/W)

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		27	Clock Frequency Ratio
		63:28	Reserved ¹
33H	51	MSR_TEST_CTRL	Test Control Register
		29:0	Reserved
		30	Streaming Buffer Disable
		31	Disable LOCK# Assertion for split locked access.
79H	121	BIOS_UPDT_TRIG	BIOS Update Trigger Register.
88H	136	BBL_CR_D0[63:0]	Chunk 0 data register D[63:0]: used to write to and read from the L2
89H	137	BBL_CR_D1[63:0]	Chunk 1 data register D[63:0]: used to write to and read from the L2
8AH	138	BBL_CR_D2[63:0]	Chunk 2 data register D[63:0]: used to write to and read from the L2
8BH	139	BIOS_SIGN/BBL_CR_D3[63:0]	BIOS Update Signature Register or Chunk 3 data register D[63:0] Used to write to and read from the L2 depending on the usage model.
C1H	193	PerfCtr0 (PERFCTR0)	Performance Counter Register See Table 2-2.
C2H	194	PerfCtr1 (PERFCTR1)	Performance Counter Register See Table 2-2.
FEH	254	MTRRcap	Memory Type Range Registers
116H	278	BBL_CR_ADDR [63:0]	Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.
		BBL_CR_ADDR [63:32]	Reserved,
		BBL_CR_ADDR [31:3]	Address bits [35:3]
		BBL_CR_ADDR [2:0]	Reserved Set to 0.
118H	280	BBL_CR_DECC[63:0]	Data ECC register D[7:0]: used to write ECC and read ECC to/from L2
119H	281	BBL_CR_CTL	Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response
		BL_CR_CTL[63:22]	Reserved
		BBL_CR_CTL[21]	Processor number ² Disable = 1 Enable = 0 Reserved

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		BBL_CR_CTL[20:19] BBL_CR_CTL[18] BBL_CR_CTL[17] BBL_CR_CTL[16] BBL_CR_CTL[15:14] BBL_CR_CTL[13:12] BBL_CR_CTL[11:10] BBL_CR_CTL[9:8] BBL_CR_CTL[7] BBL_CR_CTL[6:5]	User supplied ECC Reserved L2 Hit Reserved State from L2 Modified - 11, Exclusive - 10, Shared - 01, Invalid - 00 Way from L2 Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11 Way to L2 Reserved State to L2
		BBL_CR_CTL[4:0] 01100 01110 01111 00010 00011 010 + MESI encode 111 + MESI encode 100 + MESI encode	L2 Command Data Read w/ LRU update (RLU) Tag Read w/ Data Read (TRR) Tag Inquire (TI) L2 Control Register Read (CR) L2 Control Register Write (CW) Tag Write w/ Data Read (TWR) Tag Write w/ Data Write (TWW) Tag Write (TW)
11AH	282	BBL_CR_TRIG	Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.
11BH	283	BBL_CR_BUSY	Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY
11EH	286	BBL_CR_CTL3 BBL_CR_CTL3[63:26] BBL_CR_CTL3[25] BBL_CR_CTL3[24] BBL_CR_CTL3[23] BBL_CR_CTL3[22:20] 111 110 101 100 011 010 001 000 BBL_CR_CTL3[19] BBL_CR_CTL3[18]	Control register 3: used to configure the L2 Cache Reserved Cache bus fraction (read only) Reserved L2 Hardware Disable (read only) L2 Physical Address Range support 64GBytes 32GBytes 16GBytes 8GBytes 4GBytes 2GBytes 1GBytes 512MBytes Reserved Cache State error checking enable (read/write)

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		BBL_CR_CTL3[17:13] 00001 00010 00100 01000 10000 BBL_CR_CTL3[12:11] BBL_CR_CTL3[10:9] 00 01 10 11 BBL_CR_CTL3[8] BBL_CR_CTL3[7] BBL_CR_CTL3[6] BBL_CR_CTL3[5] BBL_CR_CTL3[4:1] BBL_CR_CTL3[0]	Cache size per bank (read/write) 256KBytes 512KBytes 1MByte 2MByte 4MBytes Number of L2 banks (read only) L2 Associativity (read only) Direct Mapped 2 Way 4 Way Reserved L2 Enabled (read/write) CRTN Parity Check Enable (read/write) Address Parity Check Enable (read/write) ECC Check Enable (read/write) L2 Cache Latency (read/write) L2 Configured (read/write)
174H	372	SYSENTER_CS_MSR	CS register target for CPL 0 code
175H	373	SYSENTER_ESP_MSR	Stack pointer for CPL 0 stack
176H	374	SYSENTER_EIP_MSR	CPL 0 code entry point
179H	377	MCG_CAP	Machine Check Global Control Register
17AH	378	MCG_STATUS	Machine Check Error Reporting Register - contains information related to machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
17BH	379	MCG_CTL	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
186H	390	PerfEvtSel0 (EVNTSEL0)	Performance Event Select Register 0 (R/W)
		7:0	Event Select Refer to Performance Counter section for a list of event encodings.
		15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
		16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	OS Controls the counting of events at Privilege level of 0.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		18	E Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	PC Enabled the signaling of performance counter overflow via BPO pin
		20	INT Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		22	ENABLE Enables the counting of performance events in both counters 1 = Enable 0 = Disable
		23	INV Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	CMASK (Counter Mask)
187H	391	PerfEvtSel1 (EVNTSEL1)	Performance Event Select for Counter 1 (R/W)
		7:0	Event Select Refer to Performance Counter section for a list of event encodings.
		15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
		16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	OS Controls the counting of events at Privilege level of 0.
		18	E Occurrence/Duration Mode Select. 1 = Occurrence 0 = Duration
		19	PC Enabled the signaling of performance counter overflow via BPO pin.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		20	INT Enables the signaling of counter overflow via input to APIC. 1 = Enable 0 = Disable
		23	INV Inverts the result of the CMASK condition. 1 = Inverted 0 = Non-Inverted
		31:24	CMASK (Counter Mask)
1D9H	473	DEBUGCTLMR	Enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.
		0	Enable/Disable Last Branch Records
		1	Branch Trap Flag
		2	Performance Monitoring/Break Point Pins
		3	Performance Monitoring/Break Point Pins
		4	Performance Monitoring/Break Point Pins
		5	Performance Monitoring/Break Point Pins
		6	Enable/Disable Execution Trace Messages
		31:7	Reserved
1DBH	475	LASTBRANCHFROMIP	32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.
1DCH	476	LASTBRANCHTOIP	32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.
1DDH	477	LASTINTFROMIP	Last INT from IP
1DEH	478	LASTINTTOIP	Last INT to IP
200H	512	MTRRphysBase0	Memory Type Range Registers
201H	513	MTRRphysMask0	Memory Type Range Registers
202H	514	MTRRphysBase1	Memory Type Range Registers
203H	515	MTRRphysMask1	Memory Type Range Registers
204H	516	MTRRphysBase2	Memory Type Range Registers
205H	517	MTRRphysMask2	Memory Type Range Registers
206H	518	MTRRphysBase3	Memory Type Range Registers
207H	519	MTRRphysMask3	Memory Type Range Registers
208H	520	MTRRphysBase4	Memory Type Range Registers
209H	521	MTRRphysMask4	Memory Type Range Registers
20AH	522	MTRRphysBase5	Memory Type Range Registers

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
20BH	523	MTRRphysMask5	Memory Type Range Registers
20CH	524	MTRRphysBase6	Memory Type Range Registers
20DH	525	MTRRphysMask6	Memory Type Range Registers
20EH	526	MTRRphysBase7	Memory Type Range Registers
20FH	527	MTRRphysMask7	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Memory Type Range Registers
2FFH	767	MTRRdefType	Memory Type Range Registers
		2:0	Default memory type
		10	Fixed MTRR enable
		11	MTRR Enable
400H	1024	MCO_CTL	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
401H	1025	MCO_STATUS	Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.
		15:0	MC_STATUS_MCACOD
		31:16	MC_STATUS_MSCOD
		57	MC_STATUS_DAM
		58	MC_STATUS_ADDRV
		59	MC_STATUS_MISCV
		60	MC_STATUS_EN. (Note: For MCO_STATUS only, this bit is hardcoded to 1.)
		61	MC_STATUS_UC
		62	MC_STATUS_O
63	MC_STATUS_V		
402H	1026	MCO_ADDR	
403H	1027	MCO_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
404H	1028	MC1_CTL	
405H	1029	MC1_STATUS	Bit definitions same as MCO_STATUS.
406H	1030	MC1_ADDR	
407H	1031	MC1_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
408H	1032	MC2_CTL	
409H	1033	MC2_STATUS	Bit definitions same as MCO_STATUS.
40AH	1034	MC2_ADDR	
40BH	1035	MC2_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
40CH	1036	MC4_CTL	
40DH	1037	MC4_STATUS	Bit definitions same as MCO_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.
40EH	1038	MC4_ADDR	Defined in MCA architecture but not implemented in P6 Family processors.
40FH	1039	MC4_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
410H	1040	MC3_CTL	
411H	1041	MC3_STATUS	Bit definitions same as MCO_STATUS.
412H	1042	MC3_ADDR	
413H	1043	MC3_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

NOTES

- 1.Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors.
- 2.The processor number feature may be disabled by setting bit 21 of the BBL_CR_CTL MSR (model-specific register address 119h) to "1". Once set, bit 21 of the BBL_CR_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset.
- 3.The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.

2.23 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5_MC_ADDR, P5_MC_TYPE, and TSC MSRs (named IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, and IA32_TIME_STAMP_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section 2.1, "Architectural MSRs"). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table 2-61. MSRs in the Pentium Processor

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
1H	1	P5_MC_TYPE	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
10H	16	TSC	See Section 17.17, "Time-Stamp Counter."
11H	17	CESR	See Section 19.6.9.1, "Control and Event Select Register (CESR)."
12H	18	CTRO	Section 19.6.9.3, "Events Counted."
13H	19	CTR1	Section 19.6.9.3, "Events Counted."

2.24 MSR INDEX

MSRs of recent processors are indexed here for convenience. IA32 MSRs are excluded from this index.

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_ALF_ESCR0	
0FH	See Table 2-55
MSR_ALF_ESCR1	
0FH	See Table 2-55
MSR_ANY_CORE_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_ANY_GFXE_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_BO_PMON_BOX_CTRL	
06_2EH	See Table 2-17
MSR_BO_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_BO_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_BO_PMON_CTRO	
06_2EH	See Table 2-17
MSR_BO_PMON_CTR1	
06_2EH	See Table 2-17
MSR_BO_PMON_CTR2	
06_2EH	See Table 2-17
MSR_BO_PMON_CTR3	
06_2EH	See Table 2-17
MSR_BO_PMON_EVTN_SELO	
06_2EH	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_BO_PMON_EVNT_SEL1 06_2EH	See Table 2-17
MSR_BO_PMON_EVNT_SEL2 06_2EH	See Table 2-17
MSR_BO_PMON_EVNT_SEL3 06_2EH	See Table 2-17
MSR_BO_PMON_MASK 06_2EH	See Table 2-17
MSR_BO_PMON_MATCH 06_2EH	See Table 2-17
MSR_B1_PMON_BOX_CTRL 06_2EH	See Table 2-17
MSR_B1_PMON_BOX_OVF_CTRL 06_2EH	See Table 2-17
MSR_B1_PMON_BOX_STATUS 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL0 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL1 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL2 06_2EH	See Table 2-17
MSR_B1_PMON_CTRL3 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SELO 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL1 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL2 06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL3 06_2EH	See Table 2-17
MSR_B1_PMON_MASK 06_2EH	See Table 2-17
MSR_B1_PMON_MATCH 06_2EH	See Table 2-17
MSR_BBL_CR_CTL 06_09H	See Table 2-59
MSR_BBL_CR_CTL3 06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_BIOS_DEBUG	
06_8CH, 06_8DH	See Table 2-45
MSR_BIOS_DONE	
06_7DH, 06_7EH	See Table 2-44
MSR_BIOS_MCU_ERRORCODE	
06_7DH, 06_7EH	See Table 2-44
06_8CH, 06_8DH	See Table 2-45
MSR_BPU_CCCR0	
0FH	See Table 2-55
MSR_BPU_CCCR1	
0FH	See Table 2-55
MSR_BPU_CCCR2	
0FH	See Table 2-55
MSR_BPU_CCCR3	
0FH	See Table 2-55
MSR_BPU_COUNTER0	
0FH	See Table 2-55
MSR_BPU_COUNTER1	
0FH	See Table 2-55
MSR_BPU_COUNTER2	
0FH	See Table 2-55
MSR_BPU_COUNTER3	
0FH	See Table 2-55
MSR_BPU_ESCR0	
0FH	See Table 2-55
MSR_BPU_ESCR1	
0FH	See Table 2-55
MSR_BR_DETECT_COUNTER_CONFIG_j	
06_66H	See Table 2-42
MSR_BR_DETECT_CTRL	
06_66H	See Table 2-42
MSR_BR_DETECT_STATUS	
06_66H	See Table 2-42
MSR_BSU_ESCR0	
0FH	See Table 2-55
MSR_BSU_ESCR1	
0FH	See Table 2-55
MSR_CO_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH	See Table 2-33
MSR_CO_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_CO_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_CO_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_CO_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_CO_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_CO_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_CO_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTRL1	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_CO_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_CO_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C1_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C1_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C1_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C1_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH	See Table 2-33
MSR_C1_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C1_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C1_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C1_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C10_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C10_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C10_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C11_PMON_BOX_FILTER	
06_3EH	See Table 2-28

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C11_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C11_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C12_PMON_BOX_FILTER 06_3EH	See Table 2-28
MSR_C12_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C12_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C13_PMON_BOX_FILTER 06_3EH	See Table 2-28
MSR_C13_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C13_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C14_PMON_BOX_FILTER 06_3EH	See Table 2-28
MSR_C14_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C14_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C15_PMON_BOX_CTL 06_3FH	See Table 2-33
MSR_C15_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C15_PMON_BOX_FILTER1 06_3FH	See Table 2-33
MSR_C15_PMON_BOX_STATUS 06_3FH	See Table 2-33
MSR_C15_PMON_CTR0 06_3FH	See Table 2-33
MSR_C15_PMON_CTR1 06_3FH	See Table 2-33
MSR_C15_PMON_CTR2 06_3FH	See Table 2-33
MSR_C15_PMON_CTR3 06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C15_PMON_EVNTSELO 06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL1 06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL2 06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL3 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_CTL 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_FILTER1 06_3FH	See Table 2-33
MSR_C16_PMON_BOX_STATUS 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL0 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL3 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL2 06_3FH	See Table 2-33
MSR_C16_PMON_CTRL3 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSELO 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL1 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL2 06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL3 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_CTL 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_FILTER1 06_3FH	See Table 2-33
MSR_C17_PMON_BOX_STATUS 06_3FH	See Table 2-33
MSR_C17_PMON_CTRL0 06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C17_PMON_CTR1 06_3FH	See Table 2-33
MSR_C17_PMON_CTR2 06_3FH	See Table 2-33
MSR_C17_PMON_CTR3 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSELO 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL1 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL2 06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL3 06_3FH	See Table 2-33
MSR_C2_PMON_BOX_CTRL 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_FILTER 06_2DH	See Table 2-24
MSR_C2_PMON_BOX_FILTER0 06_3FH	See Table 2-33
MSR_C2_PMON_BOX_FILTER1 06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_OVF_CTRL 06_2EH	See Table 2-17
MSR_C2_PMON_BOX_STATUS 06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C2_PMON_CTR0 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR1 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR2 06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C2_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C2_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C2_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C2_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C3_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C3_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C3_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C3_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C3_PMON_EVTN_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVTN_SEL4	
06_2EH	See Table 2-17
MSR_C3_PMON_EVTN_SEL5	
06_2EH	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C4_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C4_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C4_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C4_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C4_PMON_EVNT_SELO	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL1	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C4_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C5_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C5_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C5_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C5_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C5_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C5_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C5_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C5_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C6_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C6_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C6_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C6_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C6_PMON_EVTN_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVTN_SEL4	
06_2EH	See Table 2-17
MSR_C6_PMON_EVTN_SEL5	
06_2EH	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C7_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C7_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C7_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTRL4	
06_2EH	See Table 2-17
MSR_C7_PMON_CTRL5	
06_2EH	See Table 2-17
MSR_C7_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL1	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C7_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C8_PMON_BOX_CTRL	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C8_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_OVF_CTRL	
06_2FH	See Table 2-19
MSR_C8_PMON_BOX_STATUS	
06_2FH	See Table 2-19
06_3FH	See Table 2-33
MSR_C8_PMON_CTRL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTRL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTRL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C8_PMON_CTR3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTR4	
06_2FH	See Table 2-19
MSR_C8_PMON_CTR5	
06_2FH	See Table 2-19
MSR_C8_PMON_EVNT_SEL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL4	
06_2FH	See Table 2-19
MSR_C8_PMON_EVNT_SEL5	
06_2FH	See Table 2-19
MSR_C9_PMON_BOX_CTRL	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C9_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_OVF_CTRL	
06_2FH	See Table 2-19
MSR_C9_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2FH	See Table 2-19
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTRL4	
06_2FH	See Table 2-19
MSR_C9_PMON_CTRL5	
06_2FH	See Table 2-19
MSR_C9_PMON_EVTN_SEL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVTN_SEL4	
06_2FH	See Table 2-19
MSR_C9_PMON_EVTN_SEL5	
06_2FH	See Table 2-19

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_CC6_DEMOTION_POLICY_CONFIG	
06_37H	See Table 2-9
MSR_CONFIG_TDP_CONTROL	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_LEVEL1	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_LEVEL2	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_NOMINAL	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CORE_C1_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_66H	See Table 2-42
MSR_CORE_C3_RESIDENCY	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
MSR_CORE_C6_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_CORE_C7_RESIDENCY	
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
MSR_CORE_GFXE_OVERLAP_CO	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_CORE_HDC_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_CORE_PERF_LIMIT_REASONS	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H	See Table 2-30
06_3F	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_56H, 06_4FH	See Table 2-36
06_57H, 06_85H.....	See Table 2-53
MSR_CORE_THREAD_COUNT	
06_3FH.....	See Table 2-32
MSR_CRASHLOG_CONTROL	
06_7DH, 06_7EH	See Table 2-44
MSR_CRU_ESCR0	
0FH	See Table 2-55
MSR_CRU_ESCR1	
0FH	See Table 2-55
MSR_CRU_ESCR2	
0FH	See Table 2-55
MSR_CRU_ESCR3	
0FH	See Table 2-55
MSR_CRU_ESCR4	
0FH	See Table 2-55
MSR_CRU_ESCR5	
0FH	See Table 2-55
MSR_DAC_ESCR0	
0FH	See Table 2-55
MSR_DAC_ESCR1	
0FH	See Table 2-55
MSR_DRAM_ENERGY_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H.....	See Table 2-53
MSR_DRAM_PERF_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H.....	See Table 2-53
MSR_DRAM_POWER_INFO	
06_5CH, 06_7AH	See Table 2-12

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H	See Table 2-53
MSR_DRAM_POWER_LIMIT	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H	See Table 2-53
MSR_EBC_FREQUENCY_ID	
0FH	See Table 2-55
MSR_EBC_HARD_POWERON	
0FH	See Table 2-55
MSR_EBC_SOFT_POWERON	
0FH	See Table 2-55
MSR_EBL_CR_POWERON	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_EFSB_DRDY0	
0F_03H, 0F_04H	See Table 2-56
MSR_EFSB_DRDY1	
0F_03H, 0F_04H	See Table 2-56
MSR_EMON_L3_CTR_CTL0	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL1	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL2	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL3	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_EMON_L3_CTR_CTL4	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL5	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL6	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL7	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_GL_CTL	
06_0FH, 06_17H	See Table 2-3
MSR_ERROR_CONTROL	
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
MSR_FAST_UNCORE_MSRS_CAPABILITY	
06_7DH, 06_7EH	See Table 2-44
MSR_FAST_UNCORE_MSRS_CTL	
06_7DH, 06_7EH	See Table 2-44
MSR_FAST_UNCORE_MSRS_STATUS	
06_7DH, 06_7EH	See Table 2-44
MSR_FEATURE_CONFIG	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_25H, 06_2CH	See Table 2-18
06_2FH	See Table 2-19
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_FIRM_ESCR0	
0FH	See Table 2-55
MSR_FIRM_ESCR1	
0FH	See Table 2-55
MSR_FLAME_CCCR0	
0FH	See Table 2-55
MSR_FLAME_CCCR1	
0FH	See Table 2-55
MSR_FLAME_CCCR2	
0FH	See Table 2-55
MSR_FLAME_CCCR3	
0FH	See Table 2-55

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_FLAME_COUNTER0 OFH	See Table 2-55
MSR_FLAME_COUNTER1 OFH	See Table 2-55
MSR_FLAME_COUNTER2 OFH	See Table 2-55
MSR_FLAME_COUNTER3 OFH	See Table 2-55
MSR_FLAME_ESCRO OFH	See Table 2-55
MSR_FLAME_ESCR1 OFH	See Table 2-55
MSR_FSB_ESCRO OFH	See Table 2-55
MSR_FSB_ESCR1 OFH	See Table 2-55
MSR_FSB_FREQ 06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_4CH	See Table 2-11
06_0EH	See Table 2-58
MSR_GQ_SNOOP_MESF 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_GRAPHICS_PERF_LIMIT_REASONS 06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_IFSB_BUSQ0 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_BUSQ1 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_CNTR7 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_CTL6 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_SNPQ0 OF_03H, OF_04H	See Table 2-56
MSR_IFSB_SNPQ1 OF_03H, OF_04H	See Table 2-56
MSR_IQ_CCCR0 OFH	See Table 2-55
MSR_IQ_CCCR1 OFH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_IQ_CCCR2	
OFH	See Table 2-55
MSR_IQ_CCCR3	
OFH	See Table 2-55
MSR_IQ_CCCR4	
OFH	See Table 2-55
MSR_IQ_CCCR5	
OFH	See Table 2-55
MSR_IQ_COUNTER0	
OFH	See Table 2-55
MSR_IQ_COUNTER1	
OFH	See Table 2-55
MSR_IQ_COUNTER2	
OFH	See Table 2-55
MSR_IQ_COUNTER3	
OFH	See Table 2-55
MSR_IQ_COUNTER4	
OFH	See Table 2-55
MSR_IQ_COUNTER5	
OFH	See Table 2-55
MSR_IQ_ESCR0	
OFH	See Table 2-55
MSR_IQ_ESCR1	
OFH	See Table 2-55
MSR_IS_ESCR0	
OFH	See Table 2-55
MSR_IS_ESCR1	
OFH	See Table 2-55
MSR_ITLB_ESCR0	
OFH	See Table 2-55
MSR_ITLB_ESCR1	
OFH	See Table 2-55
MSR_IX_ESCR0	
OFH	See Table 2-55
MSR_IX_ESCR1	
OFH	See Table 2-55
MSR_LASTBRANCH_0	
OFH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_0_FROM_IP	
06_OFH, 06_17H	See Table 2-3

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH.....	See Table 2-12
06_7AH.....	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_0_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH.....	See Table 2-12
06_7AH.....	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_1_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_1_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_10_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_10_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_11_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_11_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_12_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_12_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_13_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_13_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_14_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_14_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_15_FROM_IP	
06_5CH, 06_7AH	See Table 2-12

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_15_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_16_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_16_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_17_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_17_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_18_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_18_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_19_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_19_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_2	
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_2_FROM_IP	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_2_TO_IP	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_20_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_20_TO_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_21_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_21_TO_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_22_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_22_TO_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_LASTBRANCH_23_FROM_IP	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_23_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_24_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_24_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_25_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_25_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_26_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_26_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_27_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_27_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_28_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_28_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_29_FROM_IP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_29_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_3	
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_3_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_3_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_30_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_30_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_31_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_31_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_4	
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LASTBRANCH_4_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_4_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_5	
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LASTBRANCH_5_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_5_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
0FH.....	See Table 2-55
MSR_LASTBRANCH_6	
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LASTBRANCH_6_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_6_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_7	
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_7_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_7_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_8_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_8_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_9_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_9_TO_IP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_TOS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_INFO_0	
06_7AH	See Table 2-13
MSR_LBR_INFO_1	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_10	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_11	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_12	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_13	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_14	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_15	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_16	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_17	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_18	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_19	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_2	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_20	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_21	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_22	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_23	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_24	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_25	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_26	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_27	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_28	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_29	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_3	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_30	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_31	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_4	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_5	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_6	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_7	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_8	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_INFO_9	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH.....	See Table 2-13
MSR_LBR_SELECT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H.....	See Table 2-53
MSR_LER_FROM_LIP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
0FH	See Table 2-55
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_LER_TO_LIP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
0FH	See Table 2-55
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_MO_PMON_ADDR_MASK	
06_2EH.....	See Table 2-17
MSR_MO_PMON_ADDR_MATCH	
06_2EH.....	See Table 2-17
MSR_MO_PMON_BOX_CTRL	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH.....	See Table 2-17
MSR_MO_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-17
MSR_MO_PMON_BOX_STATUS	
06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL0	
06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL1	
06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL2	
06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL3	
06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL4	
06_2EH.....	See Table 2-17
MSR_MO_PMON_CTRL5	
06_2EH.....	See Table 2-17
MSR_MO_PMON_DSP	
06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SELO	
06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL4	
06_2EH.....	See Table 2-17
MSR_MO_PMON_EVNT_SEL5	
06_2EH.....	See Table 2-17
MSR_MO_PMON_ISS	
06_2EH.....	See Table 2-17
MSR_MO_PMON_MAP	
06_2EH.....	See Table 2-17
MSR_MO_PMON_MM_CONFIG	
06_2EH.....	See Table 2-17
MSR_MO_PMON_MSC_THR	
06_2EH.....	See Table 2-17
MSR_MO_PMON_PGT	
06_2EH.....	See Table 2-17
MSR_MO_PMON_PLD	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH.....	See Table 2-17
MSR_MO_PMON_TIMESTAMP	
06_2EH.....	See Table 2-17
MSR_MO_PMON_ZDP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ADDR_MASK	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ADDR_MATCH	
06_2EH.....	See Table 2-17
MSR_M1_PMON_BOX_CTRL	
06_2EH.....	See Table 2-17
MSR_M1_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-17
MSR_M1_PMON_BOX_STATUS	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL0	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL1	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL2	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL3	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL4	
06_2EH.....	See Table 2-17
MSR_M1_PMON_CTRL5	
06_2EH.....	See Table 2-17
MSR_M1_PMON_DSP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SELO	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL4	
06_2EH.....	See Table 2-17
MSR_M1_PMON_EVNT_SEL5	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ISS	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH.....	See Table 2-17
MSR_M1_PMON_MAP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_MM_CONFIG	
06_2EH.....	See Table 2-17
MSR_M1_PMON_MSC_THR	
06_2EH.....	See Table 2-17
MSR_M1_PMON_PGT	
06_2EH.....	See Table 2-17
MSR_M1_PMON_PLD	
06_2EH.....	See Table 2-17
MSR_M1_PMON_TIMESTAMP	
06_2EH.....	See Table 2-17
MSR_M1_PMON_ZDP	
06_2EH.....	See Table 2-17
IA32_MCO_MISC / MSR_MCO_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_MCO_RESIDENCY	
06_57H, 06_85H.....	See Table 2-53
IA32_MC1_MISC / MSR_MC1_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
IA32_MC10_ADDR / MSR_MC10_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC10_CTL / MSR_MC10_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC10_MISC / MSR_MC10_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC10_STATUS / MSR_MC10_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC11_ADDR / MSR_MC11_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC11_CTL / MSR_MC11_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC11_MISC / MSR_MC11_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC11_STATUS / MSR_MC11_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC12_ADDR / MSR_MC12_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC12_CTL / MSR_MC12_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
IA32_MC12_MISC / MSR_MC12_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC12_STATUS / MSR_MC12_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_ADDR / MSR_MC13_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_CTL / MSR_MC13_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_MISC / MSR_MC13_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC13_STATUS / MSR_MC13_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC14_ADDR / MSR_MC14_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
IA32_MC14_CTL / MSR_MC14_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC14_MISC / MSR_MC14_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC14_STATUS / MSR_MC14_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_ADDR / MSR_MC15_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_CTL / MSR_MC15_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_MISC / MSR_MC15_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC15_STATUS / MSR_MC15_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
IA32_MC16_ADDR / MSR_MC16_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC16_CTL / MSR_MC16_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC16_MISC / MSR_MC16_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC16_STATUS / MSR_MC16_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC17_ADDR / MSR_MC17_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC17_CTL / MSR_MC17_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC17_MISC / MSR_MC17_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC17_STATUS / MSR_MC17_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_ADDR / MSR_MC18_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_CTL / MSR_MC18_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_MISC / MSR_MC18_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC18_STATUS / MSR_MC18_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_ADDR / MSR_MC19_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_CTL / MSR_MC19_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_MISC / MSR_MC19_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC19_STATUS / MSR_MC19_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC2_MISC / MSR_MC2_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
IA32_MC20_ADDR / MSR_MC20_ADDR	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC20_CTL / MSR_MC20_CTL	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC20_MISC / MSR_MC20_MISC	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC20_STATUS / MSR_MC20_STATUS	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC21_ADDR / MSR_MC21_ADDR	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC21_CTL / MSR_MC21_CTL	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC21_MISC / MSR_MC21_MISC	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC21_STATUS / MSR_MC21_STATUS	
06_2EH.....	See Table 2-17
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4F.....	See Table 2-38
IA32_MC22_ADDR / MSR_MC22_ADDR	
06_3EH.....	See Table 2-26
IA32_MC22_CTL / MSR_MC22_CTL	
06_3EH.....	See Table 2-26
IA32_MC22_MISC / MSR_MC22_MISC	
06_3EH.....	See Table 2-26
IA32_MC22_STATUS / MSR_MC22_STATUS	
06_3EH.....	See Table 2-26
IA32_MC23_ADDR / MSR_MC23_ADDR	
06_3EH.....	See Table 2-26
IA32_MC23_CTL / MSR_MC23_CTL	
06_3EH.....	See Table 2-26
IA32_MC23_MISC / MSR_MC23_MISC	
06_3EH.....	See Table 2-26
IA32_MC23_STATUS / MSR_MC23_STATUS	
06_3EH.....	See Table 2-26
IA32_MC24_ADDR / MSR_MC24_ADDR	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
IA32_MC24_CTL / MSR_MC24_CTL	
06_3EH.....	See Table 2-26
IA32_MC24_MISC / MSR_MC24_MISC	
06_3EH.....	See Table 2-26
IA32_MC24_STATUS / MSR_MC24_STATUS	
06_3EH.....	See Table 2-26
IA32_MC25_ADDR / MSR_MC25_ADDR	
06_3EH.....	See Table 2-26
IA32_MC25_CTL / MSR_MC25_CTL	
06_3EH.....	See Table 2-26
IA32_MC25_MISC / MSR_MC25_MISC	
06_3EH.....	See Table 2-26
IA32_MC25_STATUS / MSR_MC25_STATUS	
06_3EH.....	See Table 2-26
IA32_MC26_ADDR / MSR_MC26_ADDR	
06_3EH.....	See Table 2-26
IA32_MC26_CTL / MSR_MC26_CTL	
06_3EH.....	See Table 2-26
IA32_MC26_MISC / MSR_MC26_MISC	
06_3EH.....	See Table 2-26
IA32_MC26_STATUS / MSR_MC26_STATUS	
06_3EH.....	See Table 2-26
IA32_MC27_ADDR / MSR_MC27_ADDR	
06_3EH.....	See Table 2-26
IA32_MC27_CTL / MSR_MC27_CTL	
06_3EH.....	See Table 2-26
IA32_MC27_MISC / MSR_MC27_MISC	
06_3EH.....	See Table 2-26
IA32_MC27_STATUS / MSR_MC27_STATUS	
06_3EH.....	See Table 2-26
IA32_MC28_ADDR / MSR_MC28_ADDR	
06_3EH.....	See Table 2-26
IA32_MC28_CTL / MSR_MC28_CTL	
06_3EH.....	See Table 2-26
IA32_MC28_MISC / MSR_MC28_MISC	
06_3EH.....	See Table 2-26
IA32_MC28_STATUS / MSR_MC28_STATUS	
06_3EH.....	See Table 2-26
IA32_MC29_ADDR / MSR_MC29_ADDR	
06_3EH.....	See Table 2-27
IA32_MC29_CTL / MSR_MC29_CTL	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-27
IA32_MC29_MISC / MSR_MC29_MISC	
06_3EH.....	See Table 2-27
IA32_MC29_STATUS / MSR_MC29_STATUS	
06_3EH.....	See Table 2-27
IA32_MC3_ADDR / MSR_MC3_ADDR	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC3_CTL / MSR_MC3_CTL	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC3_MISC / MSR_MC3_MISC	
06_0FH, 06_17H	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_0EH.....	See Table 2-58
IA32_MC3_STATUS / MSR_MC3_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC30_ADDR / MSR_MC30_ADDR	
06_3EH.....	See Table 2-27
IA32_MC30_CTL / MSR_MC30_CTL	
06_3EH.....	See Table 2-27
IA32_MC30_MISC / MSR_MC30_MISC	
06_3EH.....	See Table 2-27
IA32_MC30_STATUS / MSR_MC30_STATUS	
06_3EH.....	See Table 2-27
IA32_MC31_ADDR / MSR_MC31_ADDR	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-27
IA32_MC31_CTL / MSR_MC31_CTL	
06_3EH.....	See Table 2-27
IA32_MC31_MISC / MSR_MC31_MISC	
06_3EH.....	See Table 2-27
IA32_MC31_STATUS / MSR_MC31_STATUS	
06_3EH.....	See Table 2-27
IA32_MC4_ADDR / MSR_MC4_ADDR	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC4_CTL / MSR_MC4_CTL	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
IA32_MC4_CTL2 / MSR_MC4_CTL2	
06_2AH, 06_2DH	See Table 2-20
IA32_MC4_STATUS / MSR_MC4_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_MC5_ADDR / MSR_MC5_ADDR	
06_0FH, 06_17H	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_57H, 06_85H.....	See Table 2-53

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH.....	See Table 2-58
IA32_MC5_CTL / MSR_MC5_CTL	
06_0FH, 06_17H.....	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
IA32_MC5_MISC / MSR_MC5_MISC	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_0EH.....	See Table 2-58
IA32_MC5_STATUS / MSR_MC5_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_4FH.....	See Table 2-38
06_57H, 06_85H.....	See Table 2-53
06_0EH.....	See Table 2-58
IA32_MC6_ADDR / MSR_MC6_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC6_CTL / MSR_MC6_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH.....	See Table 2-38
MSR_MC6_DEMOTION_POLICY_CONFIG	
06_37H.....	See Table 2-9
IA32_MC6_MISC / MSR_MC6_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
MSR_MC6_RESIDENCY_COUNTER	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_37H.....	See Table 2-9
06_57H, 06_85H.....	See Table 2-53
IA32_CORE_CAPABILITIES (Note there are no architecturally defined bits; all bits are model-specific)	
06_86H.....	See Table 2-14
06_8CH, 06_8DH.....	See Table 2-45
IA32_MC6_STATUS / MSR_MC6_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_ADDR / MSR_MC7_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_CTL / MSR_MC7_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_MISC / MSR_MC7_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC7_STATUS / MSR_MC7_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC8_ADDR / MSR_MC8_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC8_CTL / MSR_MC8_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC8_MISC / MSR_MC8_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC8_STATUS / MSR_MC8_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_4FH.....	See Table 2-38
IA32_MC9_ADDR / MSR_MC9_ADDR	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC9_CTL / MSR_MC9_CTL	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC9_MISC / MSR_MC9_MISC	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
IA32_MC9_STATUS / MSR_MC9_STATUS	
06_2EH.....	See Table 2-17
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26
06_3F.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-37
06_4FH.....	See Table 2-38
MSR_MCG_MISC	
0FH.....	See Table 2-55
MSR_MCG_R10	
0FH.....	See Table 2-55
MSR_MCG_R11	
0FH.....	See Table 2-55
MSR_MCG_R12	
0FH.....	See Table 2-55
MSR_MCG_R13	
0FH.....	See Table 2-55
MSR_MCG_R14	
0FH.....	See Table 2-55
MSR_MCG_R15	
0FH.....	See Table 2-55
MSR_MCG_R8	
0FH.....	See Table 2-55
MSR_MCG_R9	
0FH.....	See Table 2-55
MSR_MCG_RAX	
0FH.....	See Table 2-55
MSR_MCG_RBP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
OFH.....	See Table 2-55
MSR_MCG_RBX	
OFH.....	See Table 2-55
MSR_MCG_RCX	
OFH.....	See Table 2-55
MSR_MCG_RDI	
OFH.....	See Table 2-55
MSR_MCG_RDX	
OFH.....	See Table 2-55
MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5	
OFH.....	See Table 2-55
MSR_MCG_RFLAGS	
OFH.....	See Table 2-55
MSR_MCG_RIP	
OFH.....	See Table 2-55
MSR_MCG_RSI	
OFH.....	See Table 2-55
MSR_MCG_RSP	
OFH.....	See Table 2-55
MSR_MEMORY_CTRL	
06_86H.....	See Table 2-14
06_7DH, 06_7EH.....	See Table 2-44
06_97H, 06_9AH, 06_BFH.....	See Table 2-46
MSR_MISC_FEATURE_CONTROL	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
MSR_MISC_PWR_MGMT	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
MSR_MOB_ESCRO	
OFH.....	See Table 2-55
MSR_MOB_ESCR1	
OFH.....	See Table 2-55
MSR_MS_CCCRO	
OFH.....	See Table 2-55
MSR_MS_CCCR1	
OFH.....	See Table 2-55
MSR_MS_CCCR2	
OFH.....	See Table 2-55
MSR_MS_CCCR3	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
0FH.....	See Table 2-55
MSR_MS_COUNTER0	
0FH.....	See Table 2-55
MSR_MS_COUNTER1	
0FH.....	See Table 2-55
MSR_MS_COUNTER2	
0FH.....	See Table 2-55
MSR_MS_COUNTER3	
0FH.....	See Table 2-55
MSR_MS_ESCRO	
0FH.....	See Table 2-55
MSR_MS_ESCR1	
0FH.....	See Table 2-55
MSR_MTRRCAP	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH....	See Table 2-39
MSR_OFFCORE_RSP_0	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_OFFCORE_RSP_1	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_25H, 06_2CH.....	See Table 2-18
06_2FH.....	See Table 2-19
06_2AH, 06_2DH.....	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_PACKAGE_ENERGY_TIME_STATUS	
06_6AH, 06_6CH.....	See Table 2-51
MSR_PCIE_PLL_RATIO	
06_3FH.....	See Table 2-32
MSR_PCU_PMON_BOX_CTL	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_BOX_FILTER	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_BOX_STATUS	
06_3EH.....	See Table 2-28
06_3FH.....	See Table 2-33
MSR_PCU_PMON_CTRL0	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_PCU_PMON_CTR1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_CTR2	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_CTR3	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSELO	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSEL1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSEL2	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PCU_PMON_EVNTSEL3	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_PEBS_DATA_CFG	
06_7DH, 06_7EH.....	See Table 2-44
MSR_PEBS_ENABLE	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH.....	See Table 2-12
06_7AH.....	See Table 2-13
06_86H.....	See Table 2-14
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_3EH.....	See Table 2-27
06_57H, 06_85H.....	See Table 2-53
0FH.....	See Table 2-55
MSR_PEBS_FRONTEND	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH....	See Table 2-39
MSR_PEBS_LD_LAT	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
MSR_PEBS_MATRIX_VERT	
0FH.....	See Table 2-55

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_PEBS_NUM_ALT	
06_2DH.....	See Table 2-23
MSR_PERF_CAPABILITIES	
06_0FH, 06_17H.....	See Table 2-3
MSR_PERF_GLOBAL_CTRL	
06_0FH, 06_17H.....	See Table 2-3
MSR_PERF_GLOBAL_OVF_CTRL	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_PERF_GLOBAL_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_PERF_METRICS	
06_7DH, 06_7EH.....	See Table 2-44
MSR_PERF_STATUS	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
06_2AH, 06_2DH.....	See Table 2-20
MSR_PKG_C10_RESIDENCY	
06_5CH, 06_7AH.....	See Table 2-12
06_45H.....	See Table 2-30 and Table 2-31
06_4FH.....	See Table 2-38
MSR_PKG_C2_RESIDENCY	
06_27H.....	See Table 2-5
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_PKG_C3_RESIDENCY	
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH.....	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20
06_66H.....	See Table 2-42
06_57H, 06_85H.....	See Table 2-53
MSR_PKG_C4_RESIDENCY	
06_27H.....	See Table 2-5
MSR_PKG_C6_RESIDENCY	
06_27H.....	See Table 2-5
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH.....	See Table 2-7
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH.....	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_57H, 06_85H	See Table 2-53
MSR_PKG_C7_RESIDENCY	
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_C8_RESIDENCY	
06_45H	See Table 2-31
06_4FH	See Table 2-38
MSR_PKG_C9_RESIDENCY	
06_45H	See Table 2-31
06_4FH	See Table 2-38
MSR_PKG_CST_CONFIG_CONTROL	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_4CH	See Table 2-11
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-30
06_45H	See Table 2-31
06_3F	See Table 2-32
06_3DH	See Table 2-35
06_56H, 06_4FH	See Table 2-36
06_57H, 06_85H	See Table 2-53
MSR_PKG_ENERGY_STATUS	
06_37H, 06_4AH, 06_4CH, 06_5AH, 06_5DH	See Table 2-8
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3DH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_HDC_CONFIG	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_HDC_DEEP_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_HDC_SHALLOW_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_PERF_STATUS	
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2DH	See Table 2-23

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3AH, 06_3EH	See Table 2-26
06_3CH, 06_3DH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-29
06_57H, 06_85H.....	See Table 2-53
MSR_PKG_POWER_INFO	
06_4DH.....	See Table 2-10
06_5CH, 06_7AH, 06_86H.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3DH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_POWER_LIMIT	
06_37H, 06_4AH, 06_4CH, 06_5AH, 06_5DH	See Table 2-8
06_4DH.....	See Table 2-10
06_5CH, 06_7AH, 06_86H.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3DH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H.....	See Table 2-53
MSR_PKGC_IRTL1	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
MSR_PKGC_IRTL2	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-29
MSR_PKGC3_IRTL	
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH	See Table 2-20
MSR_PKGC6_IRTL	
06_2AH, 06_2DH	See Table 2-20
MSR_PKGC7_IRTL	
06_2AH.....	See Table 2-21
MSR_PLATFORM_BRV	
0FH.....	See Table 2-55
MSR_PLATFORM_ENERGY_COUNTER	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PLATFORM_ID	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
MSR_PLATFORM_INFO	
06_5CH, 06_7AH	See Table 2-12

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29 and Table 2-30
06_56H, 06_4FH	See Table 2-36
06_57H	See Table 2-53
MSR_PLATFORM_POWER_LIMIT	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PMG_IO_CAPTURE_BASE	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_4CH	See Table 2-11
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_57H	See Table 2-53
MSR_PMH_ESCRO	
0FH	See Table 2-55
MSR_PMH_ESCR1	
0FH	See Table 2-55
MSR_PMON_GLOBAL_CONFIG	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_PMON_GLOBAL_CTL	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_PMON_GLOBAL_STATUS	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_POWER_CTL	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
MSR_PPO_ENERGY_STATUS	
06_37H, 06_4AH, 06_5AH, 06_5DH	See Table 2-8
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H	See Table 2-53
MSR_PPO_POLICY	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_45H	See Table 2-21
MSR_PP0_POWER_LIMIT	
06_4CH.....	See Table 2-11
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H.....	See Table 2-53
MSR_PP1_ENERGY_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PP1_POLICY	
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PP1_POWER_LIMIT	
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PPERF	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
IA32_PPIN / MSR_PPIN	
06_3EH.....	See Table 2-26
06_56H, 06_4FH	See Table 2-36
06_55H.....	See Table 2-50
06_57H, 06_85H	See Table 2-53
IA32_PPIN_CTL / MSR_PPIN_CTL	
06_3EH.....	See Table 2-26
06_56H, 06_4FH	See Table 2-36
06_55H.....	See Table 2-50
06_57H, 06_85H	See Table 2-53
MSR_PRMRR_BASE_0	
06_7DH, 06_7EH	See Table 2-44
MSR_PRMRR_PHYS_BASE	
06_8EH, 06_9EH	See Table 2-41
MSR_PRMRR_PHYS_MASK	
06_8EH, 06_9EH	See Table 2-41
MSR_PRMRR_VALID_CONFIG	
06_8EH, 06_9EH	See Table 2-41
MSR_RELOAD_FIXED_CTRx	
06_86H	See Table 2-14
MSR_RELOAD_PMCx	
06_86H	See Table 2-14
MSR_RING_RATIO_LIMIT	
06_8EH, 06_9EH	See Table 2-41

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_RO_PMON_BOX_CTRL 06_2EH.....	See Table 2-17
MSR_RO_PMON_BOX_OVF_CTRL 06_2EH.....	See Table 2-17
MSR_RO_PMON_BOX_STATUS 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL0 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL1 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL2 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL3 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL4 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL5 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL6 06_2EH.....	See Table 2-17
MSR_RO_PMON_CTRL7 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SELO 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL1 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL2 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL3 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL4 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL5 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL6 06_2EH.....	See Table 2-17
MSR_RO_PMON_EVNT_SEL7 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P0 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P1 06_2EH.....	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_RO_PMON_IPERFO_P2 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P3 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P4 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P5 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P6 06_2EH.....	See Table 2-17
MSR_RO_PMON_IPERFO_P7 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P0 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P1 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P2 06_2EH.....	See Table 2-17
MSR_RO_PMON_QLX_P3 06_2EH.....	See Table 2-17
MSR_R1_PMON_BOX_CTRL 06_2EH.....	See Table 2-17
MSR_R1_PMON_BOX_OVF_CTRL 06_2EH.....	See Table 2-17
MSR_R1_PMON_BOX_STATUS 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR10 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR11 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR12 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR13 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR14 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR15 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR8 06_2EH.....	See Table 2-17
MSR_R1_PMON_CTR9 06_2EH.....	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_R1_PMON_EVTN_SEL10 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVTN_SEL11 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVTN_SEL12 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVTN_SEL13 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVTN_SEL14 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVTN_SEL15 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVTN_SEL8 06_2EH.....	See Table 2-17
MSR_R1_PMON_EVTN_SEL9 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P10 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P11 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P12 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P13 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P14 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P15 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P8 06_2EH.....	See Table 2-17
MSR_R1_PMON_IPERF1_P9 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P4 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P5 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P6 06_2EH.....	See Table 2-17
MSR_R1_PMON_QLX_P7 06_2EH.....	See Table 2-17
MSR_RAPL_POWER_UNIT 06_37H, 06_4AH, 06_5AH, 06_5DH	See Table 2-8

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H.....	See Table 2-20
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-36
06_57H.....	See Table 2-53
MSR_RAT_ESCR0	
0FH.....	See Table 2-55
MSR_RAT_ESCR1	
0FH.....	See Table 2-55
MSR_RING_PERF_LIMIT_REASONS	
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SO_PMON_BOX_CTRL	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_SO_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-17
MSR_SO_PMON_BOX_STATUS	
06_2EH.....	See Table 2-17
MSR_SO_PMON_CTRL0	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_CTRL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_CTRL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_CTRL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_EVNT_SELO	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_SO_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_S0_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S0_PMON_MASK	
06_2EH.....	See Table 2-17
MSR_S0_PMON_MATCH	
06_2EH.....	See Table 2-17
MSR_S1_PMON_BOX_CTRL	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_S1_PMON_BOX_OVF_CTRL	
06_2EH.....	See Table 2-17
MSR_S1_PMON_BOX_STATUS	
06_2EH.....	See Table 2-17
MSR_S1_PMON_CTRL0	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_CTRL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_CTRL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_CTRL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SELO	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SEL1	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SEL2	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_EVNT_SEL3	
06_2EH.....	See Table 2-17
06_3FH.....	See Table 2-33
MSR_S1_PMON_MASK	
06_2EH.....	See Table 2-17

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_S1_PMON_MATCH	
06_2EH.....	See Table 2-17
MSR_S2_PMON_BOX_CTL	
06_3FH.....	See Table 2-33
MSR_S2_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL0	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL1	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL2	
06_3FH.....	See Table 2-33
MSR_S2_PMON_CTRL3	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSELO	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSEL1	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSEL2	
06_3FH.....	See Table 2-33
MSR_S2_PMON_EVTSEL3	
06_3FH.....	See Table 2-33
MSR_S3_PMON_BOX_CTL	
06_3FH.....	See Table 2-33
MSR_S3_PMON_BOX_FILTER	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL0	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL1	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL2	
06_3FH.....	See Table 2-33
MSR_S3_PMON_CTRL3	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSELO	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSEL1	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSEL2	
06_3FH.....	See Table 2-33
MSR_S3_PMON_EVTSEL3	
06_3FH.....	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_SAAT_ESCR0	
0FH.....	See Table 2-55
MSR_SAAT_ESCR1	
0FH.....	See Table 2-55
MSR_SGXOWNEREPOCH0	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_SGXOWNEREPOCH1	
06_5CH, 06_7AH.....	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH.....	See Table 2-39
MSR_SMI_COUNT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_57H.....	See Table 2-53
MSR_SMM_BLOCKED	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SMM_DELAYED	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SMM_FEATURE_CONTROL	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
MSR_SMM_MCA_CAP	
06_5CH, 06_7AH.....	See Table 2-12
06_3CH, 06_45H, 06_46H.....	See Table 2-30
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-36
06_57H.....	See Table 2-53
MSR_SMRR_PHYSBASE	
06_0FH, 06_17H.....	See Table 2-3
MSR_SMRR_PHYSMASK	
06_0FH, 06_17H.....	See Table 2-3
MSR_SSU_ESCR0	
0FH.....	See Table 2-55
MSR_TBPU_ESCR0	
0FH.....	See Table 2-55
MSR_TBPU_ESCR1	
0FH.....	See Table 2-55

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_TC_ESCR0	
0FH.....	See Table 2-55
MSR_TC_ESCR1	
0FH.....	See Table 2-55
MSR_TC_PRECISE_EVENT	
0FH.....	See Table 2-55
MSR_TEMPERATURE_TARGET	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
06_2AH, 06_2DH.....	See Table 2-20
06_3EH.....	See Table 2-26
06_56H, 06_4FH.....	See Table 2-36
06_57H.....	See Table 2-53
MSR_TEST_CTRL	
P6 Family.....	See Table 2-60
MSR_THERM2_CTL	
06_0FH, 06_17H.....	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H.....	See Table 2-4
0FH.....	See Table 2-55
06_0EH.....	See Table 2-58
06_09H.....	See Table 2-59
MSR_THREAD_ID_INFO	
06_3FH.....	See Table 2-32
MSR_TRACE_HUB_STH ACPIBAR_BASE	
06_8EH, 06_9EH.....	See Table 2-41
MSR_TURBO_ACTIVATION_RATIO	
06_5CH, 06_7AH.....	See Table 2-12
06_3AH.....	See Table 2-25
06_3CH, 06_45H, 06_46H.....	See Table 2-29
06_57H.....	See Table 2-53
MSR_TURBO_GROUP_CORECNT	
06_5CH, 06_7AH.....	See Table 2-12
MSR_TURBO_POWER_CURRENT_LIMIT	
06_1AH, 06_1EH, 06_1FH, 06_2EH.....	See Table 2-15
MSR_TURBO_RATIO_LIMIT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.....	See Table 2-6
06_4DH.....	See Table 2-10
06_5CH, 06_7AH.....	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH.....	See Table 2-15
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH.....	See Table 2-16
06_2EH.....	See Table 2-17
06_25H, 06_2CH.....	See Table 2-18

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2FH.....	See Table 2-19
06_2AH, 06_45H.....	See Table 2-21
06_2DH.....	See Table 2-23
06_3EH.....	See Table 2-26 and Table 2-27
06_3CH, 06_45H, 06_46H.....	See Table 2-30
06_3FH.....	See Table 2-32
06_3DH.....	See Table 2-35
06_56H, 06_4FH.....	See Table 2-36
06_55H.....	See Table 2-50
06_57H.....	See Table 2-53
MSR_TURBO_RATIO_LIMIT1	
06_3EH.....	See Table 2-26 and Table 2-27
06_3FH.....	See Table 2-32
06_56H, 06_4FH.....	See Table 2-36
MSR_TURBO_RATIO_LIMIT2	
06_3FH.....	See Table 2-32
MSR_TURBO_RATIO_LIMIT3	
06_56H.....	See Table 2-37
06_4FH.....	See Table 2-38
MSR_TURBO_RATIO_LIMIT_CORES	
06_55H.....	See Table 2-50
MSR_U_PMON_BOX_STATUS	
06_3EH.....	See Table 2-28
06_3FH.....	See Table 2-33
MSR_U_PMON_CTR	
06_2EH.....	See Table 2-17
MSR_U_PMON_CTR0	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_U_PMON_CTR1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_U_PMON_EVNT_SEL	
06_2EH.....	See Table 2-17
MSR_U_PMON_EVNTSELO	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33
MSR_U_PMON_EVNTSEL1	
06_2DH.....	See Table 2-24
06_3FH.....	See Table 2-33

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_U_PMON_GLOBAL_CTRL	
06_2EH	See Table 2-17
MSR_U_PMON_GLOBAL_OVF_CTRL	
06_2EH	See Table 2-17
MSR_U_PMON_GLOBAL_STATUS	
06_2EH	See Table 2-17
MSR_U_PMON_UCLK_FIXED_CTL	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U_PMON_UCLK_FIXED_CTR	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U2L_ESCR0	
0FH	See Table 2-55
MSR_U2L_ESCR1	
0FH	See Table 2-55
MSR_UNC_ARB_PERFCTRO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTRO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTR2	
06_2AH	See Table 2-22

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_0_PERFCTR3 06_2AH	See Table 2-22
MSR_UNC_CBO_0_PERFEVTSELO 06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFEVTSEL1 06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFEVTSEL2 06_2AH	See Table 2-22
MSR_UNC_CBO_0_PERFEVTSEL3 06_2AH	See Table 2-22
MSR_UNC_CBO_0_UNIT_STATUS 06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFCTR0 06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFCTR1 06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFCTR2 06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFCTR3 06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFEVTSELO 06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFEVTSEL1 06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFEVTSEL2 06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFEVTSEL3 06_2AH	See Table 2-22
MSR_UNC_CBO_1_UNIT_STATUS 06_2AH	See Table 2-22

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_2_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFEVTSELO	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_3_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR0	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR1	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSELO	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL1	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_CONFIG	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_FIXED_CTR	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_FIXED_CTRL	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_GLOBAL_CTRL	
06_2AH	See Table 2-22

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_GLOBAL_STATUS	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNCORE_ADDR_OPCODE_MATCH	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_FIXED_CTR_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_FIXED_CTR0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL1	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL2	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL3	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL4	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL5	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL6	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL7	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC1	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC2	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC3	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNCORE_PMC4 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC5 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
06_2EH	See Table 2-17
MSR_UNCORE_PMC6 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC7 06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PRMRR_BASE 06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_UNCORE_PRMRR_MASK 06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_UNCORE_PRMRR_PHYS_BASE 06_8EH, 06_9EH	See Table 2-41
MSR_UNCORE_PRMRR_PHYS_MASK 06_8EH, 06_9EH	See Table 2-41
MSR_VR_CURRENT_CONFIG 06_8CH, 06_8DH	See Table 2-45
MSR_W_PMON_BOX_CTRL 06_2EH	See Table 2-17
MSR_W_PMON_BOX_OVF_CTRL 06_2EH	See Table 2-17
MSR_W_PMON_BOX_STATUS 06_2EH	See Table 2-17
MSR_W_PMON_CTRL0 06_2EH	See Table 2-17
MSR_W_PMON_CTRL1 06_2EH	See Table 2-17
MSR_W_PMON_CTRL2 06_2EH	See Table 2-17
MSR_W_PMON_CTRL3 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SELO 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL1 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL2 06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL3	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
MSR_W_PMON_FIXED_CTR	
06_2EH	See Table 2-17
MSR_W_PMON_FIXED_CTR_CTL	
06_2EH	See Table 2-17
MSR_WEIGHTED_CORE_CO	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MTRRfix16K_80000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix16K_A0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_C0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_C8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_D0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_D8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_E0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_E8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_F0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_F8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix64K_00000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase0	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase1	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase2	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase3	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase4	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase5	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase6	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase7	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask0	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask1	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask2	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask3	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask4	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask5	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask6	

MODEL-SPECIFIC REGISTERS (MSRS)

MSR Name and CUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask7	
06_0EH	See Table 2-58
P6 Family	See Table 2-60