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

Documentation Changes

April 2016

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: 252046-050

Contents

Revision History

Revision	Description	Date
-001	• Initial release	November 2002
-002	• Added 1-10 Documentation Changes. • Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual	December 2002
-003	• Added 9 -17 Documentation Changes. • Removed Documentation Change #6 - References to bits Gen and Len Deleted. • Removed Documentation Change #4 - VIF Information Added to CLI Discussion	February 2003
-004	• Removed Documentation changes 1-17. • Added Documentation changes 1-24.	June 2003
-005	• Removed Documentation Changes 1-24. • Added Documentation Changes 1-15.	September 2003
-006	• Added Documentation Changes 16- 34.	November 2003
-007	• Updated Documentation changes 14, 16, 17, and 28. • Added Documentation Changes 35-45.	January 2004
-008	• Removed Documentation Changes 1-45. • Added Documentation Changes 1-5.	March 2004
-009	• Added Documentation Changes 7-27.	May 2004
-010	• Removed Documentation Changes 1-27. • Added Documentation Changes 1.	August 2004
-011	• Added Documentation Changes 2-28.	November 2004
-012	• Removed Documentation Changes 1-28. • Added Documentation Changes 1-16.	March 2005
-013	• Updated title. • There are no Documentation Changes for this revision of the document.	July 2005
-014	• Added Documentation Changes 1-21.	September 2005
-015	• Removed Documentation Changes 1-21. • Added Documentation Changes 1-20.	March 9, 2006
-016	• Added Documentation changes 21-23.	March 27, 2006
-017	• Removed Documentation Changes 1-23. • Added Documentation Changes 1-36.	September 2006
-018	• Added Documentation Changes 37-42.	October 2006
-019	• Removed Documentation Changes 1-42. • Added Documentation Changes 1-19.	March 2007
-020	• Added Documentation Changes 20-27.	May 2007
-021	• Removed Documentation Changes 1-27. • Added Documentation Changes 1-6	November 2007
-022	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-6	August 2008
-023	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-21	March 2009

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

Revision	Description	Date
-048	• Removed Documentation Changes 1-19 • Add Documentation Changes 1-33	September 2015
-049	• Removed Documentation Changes 1-33 • Add Documentation Changes 1-33	December 2015
-050	• Removed Documentation Changes 1-33 • Add Documentation Changes 1-38	April 2016

§

Preface

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

Affected Documents

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

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(Sheet 1 of 2)

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 1, Volume 1
2	Updates to Chapter 10, Volume 1
3	Updates to Chapter 12, Volume 1
4	Updates to Chapter 14, Volume 1
5	Updates to Chapter 1, Volume 2A
6	Updates to Chapter 3, Volume 2A
7	Updates to Chapter 4, Volume 2B
8	Updates to Appendix A, Volume 2C
9	Updates to Chapter 1, Volume 3A
10	Updates to Chapter 2, Volume 3A
11	Updates to Chapter 6, Volume 3A
12	Updates to Chapter 7, Volume 3A
13	Updates to Chapter 9, Volume 3A
14	Updates to Chapter 10, Volume 3A
15	Updates to Chapter 14, Volume 3B
16	Updates to Chapter 16, Volume 3B
17	Updates to Chapter 17, Volume 3B
18	Updates to Chapter 18, Volume 3B
19	Updates to Chapter 19, Volume 3B
20	Updates to Chapter 23, Volume 3B
21	Updates to Chapter 24, Volume 3B
22	Updates to Chapter 25, Volume 3C
23	Updates to Chapter 26, Volume 3C
24	Updates to Chapter 27, Volume 3C
25	Updates to Chapter 29, Volume 3C
26	Updates to Chapter 30, Volume 3C
27	Updates to Chapter 35, Volume 3C

Documentation Changes(Sheet 2 of 2)

No.	DOCUMENTATION CHANGES
28	Updates to Chapter 36, Volume 3C
29	Updates to Chapter 37, Volume 3D
30	Updates to Chapter 38, Volume 3D
31	Updates to Chapter 39, Volume 3D
32	Updates to Chapter 40, Volume 3D
33	Updates to Chapter 41, Volume 3D
34	Updates to Chapter 42, Volume 3D
35	Updates to Chapter 43, Volume 3D
36	Updates to Appendix A, Volume 3D
37	Updates to Appendix B, Volume 3D
38	Updates to Appendix C, Volume 3D

Documentation Changes

1. Updates to Chapter 1, Volume 1

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

...

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme processor QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processor family

- Intel® Atom™ processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families
- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel® Atom™ processor X7-Z8000 and X5-Z8000 series
- Intel® Atom™ processor Z3400 series
- Intel® Atom™ processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200 and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processor QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel® Atom™ processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Intel® microarchitecture code name Nehalem. Intel® microarchitecture code name Westmere is a 32 nm version of Intel® microarchitecture code name Nehalem. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on Intel® microarchitecture code name Westmere. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Intel® microarchitecture code name Sandy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and the 3rd generation Intel® Core™ processors are based on the Intel® microarchitecture code name Ivy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Intel® microarchitecture code name Ivy Bridge-E and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Intel® microarchitecture code name Haswell and support Intel 64 architecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Intel® microarchitecture code name Broadwell and support Intel 64 architecture.

The Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Intel® microarchitecture code name Skylake and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Intel® microarchitecture code name Haswell-E and support Intel 64 architecture.

The Intel® Atom™ processor Z8000 series is based on the Intel microarchitecture code name Airmont.

The Intel® Atom™ processor Z3400 series and the Intel® Atom™ processor Z3500 series are based on the Intel microarchitecture code name Silvermont.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme processors, Intel Core 2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel’s 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.

...

2. Updates to Chapter 10, Volume 1

Change bars show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 1:* Basic Architecture.

...

10.4.6.2 Caching of Temporal vs. Non-Temporal Data

Data referenced by a program can be temporal (data will be used again) or non-temporal (data will be referenced once and not reused in the immediate future). For example, program code is generally temporal, whereas, multimedia data, such as the display list in a 3-D graphics application, is often non-temporal. To make efficient use of the processor's caches, it is generally desirable to cache temporal data and not cache non-temporal data. Overloading the processor's caches with non-temporal data is sometimes referred to as "polluting the caches." The SSE and SSE2 cacheability control instructions enable a program to write non-temporal data to memory in a manner that minimizes pollution of caches.

These SSE and SSE2 non-temporal store instructions minimize cache pollutions by treating the memory being accessed as the write combining (WC) type. If a program specifies a non-temporal store with one of these instructions and the memory type of the destination region is write back (WB), write through (WT), or write combining (WC), the processor will do the following:

- If the memory location being written to is present in the cache hierarchy, the data in the caches is evicted.[1]
- The non-temporal data is written to memory with WC semantics.

See also: Chapter 11, "Memory Cache Control," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3A.*

Using the WC semantics, the store transaction will be weakly ordered, meaning that the data may not be written to memory in program order, and the store will not write allocate (that is, the processor will not fetch the corresponding cache line into the cache hierarchy, prior to performing the store). Also, different processor implementations may choose to collapse and combine these stores.

The memory type of the region being written to can override the non-temporal hint, if the memory address specified for the non-temporal store is in uncacheable memory. Uncacheable as referred to here means that the region being written to has been mapped with either an uncacheable (UC) or write protected (WP) memory type.

In general, WC semantics require software to ensure coherence, with respect to other processors and other system agents (such as graphics cards). Appropriate use of synchronization and fencing must be performed for producer-consumer usage models. Fencing ensures that all system agents have global visibility of the stored data; for instance, failure to fence may result in a written cache line staying within a processor and not being visible to other agents.

The memory type visible on the bus in the presence of memory type aliasing is implementation specific. As one possible example, the memory type written to the bus may reflect the memory type for the first store to this line, as seen in program order; other alternatives are possible. This behavior should be considered reserved, and dependence on the behavior of any particular implementation risks future incompatibility.

NOTE

Some older CPU implementations (e.g., Pentium M) may implement non-temporal stores by updating in place data that already reside in the cache hierarchy. For such processors, the destination region should also be mapped as WC. If mapped as WB or WT, there is the potential for speculative processor reads to bring the data into the caches; in this case, non-temporal

[1] Some older CPU implementations (e.g., Pentium M) allowed addresses being written with a non-temporal store instruction to be updated in-place if the memory type was not WC and line was already in the cache.

stores would then update in place, and data would not be flushed from the processor by a subsequent fencing operation.

...

3. Updates to Chapter 12, Volume 1

Change bars show changes to Chapter 12 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

...

12.10.3 Streaming Load Hint Instruction

Historically, CPU read accesses of WC memory type regions have significantly lower throughput than accesses to cacheable memory.

The streaming load instruction in SSE4.1, MOVNTDQA, provides a non-temporal hint that can cause adjacent 16-byte items within an aligned 64-byte region of WC memory type (a streaming line) to be fetched and held in a small set of temporary buffers ("streaming load buffers"). Subsequent streaming loads to other aligned 16-byte items in the same streaming line may be satisfied from the streaming load buffer and can improve throughput.

Programmers are advised to use the following practices to improve the efficiency of MOVNTDQA streaming loads from WC memory:

- Streaming loads must be 16-byte aligned.
- Temporally group streaming loads of the same streaming cache line for effective use of the small number of streaming load buffers. If loads to the same streaming line are excessively spaced apart, it may cause the streaming line to be re-fetched from memory.
- Temporally group streaming loads from at most a few streaming lines together. The number of streaming load buffers is small; grouping a modest number of streams will avoid running out of streaming load buffers and the resultant re-fetching of streaming lines from memory.
- Avoid writing to a streaming line until all 16-byte-aligned reads from the streaming line have occurred. Reading a 16-byte item from a streaming line that has been written, may cause the streaming line to be re-fetched.
- Avoid reading a given 16-byte item within a streaming line more than once; repeated loads of a particular 16-byte item are likely to cause the streaming line to be re-fetched.
- The streaming load buffers, reflecting the WC memory type characteristics, are not required to be snooped by operations from other agents. Software should not rely upon such coherency actions to provide any data coherency with respect to other logical processors or bus agents. Rather, software must insure the consistency of WC memory accesses between producers and consumers.
- Streaming loads may be weakly ordered and may appear to software to execute out of order with respect to other memory operations. Software must explicitly use MFENCE if it needs to preserve order among streaming loads or between streaming loads and other memory operations.
- Streaming loads must not be used to reference memory addresses that are mapped to I/O devices having side effects or when reads to these devices are destructive. This is because MOVNTDQA is speculative in nature.

Example 12-1 provides a sketch of the basic assembly sequences that illustrate the principles of using MOVNTDQA in a situation with a producer-consumer accessing a WC memory region.

Example 12-1 Sketch of MOVNTDQA Usage of a Consumer and a PCI Producer

```
// P0: producer is a PCI device writing into the WC space
# the PCI device updates status through a UC flag, "u_dev_status" .
# the protocol for "u_dev_status" : 0: produce; 1: consume; 2: all done

    mov eax, $0
    mov [u_dev_status], eax
producerStart:
    mov eax, [u_dev_status]     # poll status flag to see if consumer is requestion data
    cmp eax, $0                 #
    jne done                    # I no longer need to produce
    commence PCI writes to WC region..

    mov eax, $1  # producer ready to notify the consumer via status flag
    mov  [u_dev_status], eax
# now wait for consumer to signal its status
spinloop:
    cmp [u_dev_status], $1      # did I get a signal from the consumer ?
    jne producerStart           # yes I did
    jmp spinloop                # check again
done:
// producer is finished at this point
```

```
// P1: consumer check PCI status flag to consume WC data
    mov eax, $0  # request to the producer
    mov [u_dev_status], eax
consumerStart:
    mov; eax, [u_dev_status]  # reads the value of the PCI status
    cmp eax, $1                     # has producer written
    jne consumerStart               # tight loop; make it more efficient with pause, etc.
    mfence   # producer finished device writes to WC, ensure WC region is coherent
ntread:
    movntdqa xmm0, [addr]
    movntdqa xmm1, [addr + 16]
    movntdqa xmm2, [addr + 32]
    movntdqa xmm3, [addr + 48]
    ... # do any more NT reads as needed
    mfence  # ensure PCI device reads the correct value of [u_dev_status]
# now decide whether we are done or we need the producer to produce more data
# if we are done write a 2 into the variable, otherwise write a 0 into the variable
    mov eax, $0/$2          # end or continue producing
    mov [u_dev_status], eax
# if I want to consume again I will jump back to consumerStart after storing a 0 into eax
# otherwise I am done
```

...

4. Updates to Chapter 14, Volume 1

Change bars show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

...

14.9 MEMORY ALIGNMENT

Memory alignment requirements on VEX-encoded instruction differs from non-VEX-encoded instructions. Memory alignment applies to non-VEX-encoded SIMD instructions in three categories:

- Explicitly-aligned SIMD load and store instructions accessing 16 bytes of memory (e.g. MOVAPD, MOVAPS, MOVDQA, etc.). These instructions always require memory address to be aligned on 16-byte boundary.
- Explicitly-unaligned SIMD load and store instructions accessing 16 bytes or less of data from memory (e.g. MOVUPD, MOVUPS, MOVDQU, MOVQ, MOVD, etc.). These instructions do not require memory address to be aligned on 16-byte boundary.
- The vast majority of arithmetic and data processing instructions in legacy SSE instructions (non-VEX-encoded SIMD instructions) support memory access semantics. When these instructions access 16 bytes of data from memory, the memory address must be aligned on 16-byte boundary.

Most arithmetic and data processing instructions encoded using the VEX prefix and performing memory accesses have more flexible memory alignment requirements than instructions that are encoded without the VEX prefix. Specifically,

- With the exception of explicitly aligned 16 or 32 byte SIMD load/store instructions, most VEX-encoded, arithmetic and data processing instructions operate in a flexible environment regarding memory address alignment, i.e. VEX-encoded instruction with 32-byte or 16-byte load semantics will support unaligned load operation by default. Memory arguments for most instructions with VEX prefix operate normally without causing #GP(0) on any byte-granularity alignment (unlike Legacy SSE instructions). The instructions that require explicit memory alignment requirements are listed in Table 14-22.

Software may see performance penalties when unaligned accesses cross cacheline 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 list of guaranteed atomic operations are described in Section 8.1.1 of *IA-32 Intel® Architecture Software Developer's Manual, Volumes 3A.* AVX and FMA instructions do not introduce any new guaranteed atomic memory operations.

AVX instructions can generate an #AC(0) fault on misaligned 4 or 8-byte memory references in Ring-3 when CR0.AM=1. 16 and 32-byte memory references will not generate #AC(0) fault. See Table 14-21 for details.

Certain AVX instructions always require 16- or 32-byte alignment (see the complete list of such instructions in Table 14-22). These instructions will #GP(0) if not aligned to 16-byte boundaries (for 16-byte granularity loads and stores) or 32-byte boundaries (for 32-byte loads and stores).

Table 14-21 Alignment Faulting Conditions when Memory Access is Not Aligned

EFLAGS.AC==1 && Ring-3 && CR0.AM == 1			0	1
Instruction Type	AVX, FMA,	16- or 32-byte “explicitly unaligned” loads and stores (see Table 14-23)	no fault	no fault
		VEX op YMM, m256	no fault	no fault
		VEX op XMM, m128	no fault	no fault
		“explicitly aligned” loads and stores (see Table 14-22)	#GP(0)	#GP(0)
		2, 4, or 8-byte loads and stores	no fault	#AC(0)
	SSE	16 byte “explicitly unaligned” loads and stores (see Table 14-23)	no fault	no fault
		op XMM, m128	#GP(0)	#GP(0)
		“explicitly aligned” loads and stores (see Table 14-22)	#GP(0)	#GP(0)
		2, 4, or 8-byte loads and stores	no fault	#AC(0)

Table 14-22 Instructions Requiring Explicitly Aligned Memory

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

...

5. Updates to Chapter 1, Volume 2A

Change bars show changes to Chapter 1 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A:* Instruction Set Reference, A-M.

...

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme processor QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processor family
- Intel® Atom™ processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families

- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel® Atom™ processor X7-Z8000 and X5-Z8000 series
- Intel® Atom™ processor Z3400 series
- Intel® Atom™ processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel® Atom™ processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Intel® microarchitecture code name Nehalem. Intel® microarchitecture code name Westmere is a 32 nm version of Intel® microarchitecture code name Nehalem. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on Intel® microarchitecture code name Westmere. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Intel® microarchitecture code name Sandy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Intel® microarchitecture code name Ivy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Intel® microarchitecture code name Ivy Bridge-E and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Intel® microarchitecture code name Haswell and support Intel 64 architecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Intel® microarchitecture code name Broadwell and support Intel 64 architecture.

The Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Intel® microarchitecture code name Skylake and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Intel® microarchitecture code name Haswell-E and support Intel 64 architecture.

The Intel® Atom™ processor Z8000 series is based on the Intel microarchitecture code name Airmont.

The Intel® Atom™ processor Z3400 series and the Intel® Atom™ processor Z3500 series are based on the Intel microarchitecture code name Silvermont.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme, Intel® Core™2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel’s 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.

...

6. Updates to Chapter 3, Volume 2A

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

...

3.1.1.10 Intel® C/C++ Compiler Intrinsics Equivalents Section

The Intel C/C++ compiler intrinsic functions give access to the full power of the Intel Architecture Instruction Set, while allowing the compiler to optimize register allocation and instruction scheduling for faster execution. Most of these functions are associated with a single IA instruction, although some may generate multiple instructions or different instructions depending upon how they are used. In particular, these functions are used to invoke instructions that perform operations on vector registers that can hold multiple data elements. These SIMD instructions use the following data types.

- __m128, __m256 and __m512 can represent 4, 8 or 16 packed single-precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The __m128 data type is also used with various single-precision floating-point scalar instructions that perform calculations using only the lowest 32 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.
- __m128d, __m256d and __m512d can represent 2, 4 or 8 packed double-precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The __m128d data type is also used with various double-precision floating-point scalar instructions that perform calculations using only the lowest 64 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.

- __m128i, __m256i and __m512i can represent integer data in bytes, words, doublewords, quadwords, and occasionally larger data types.

Each of these data types incorporates in its name the number of bits it can hold. For example, the __m128 type holds 128 bits, and because each single-precision floating-point value is 32 bits long the __m128 type holds (128/ 32) or four values. Normally the compiler will allocate memory for these data types on an even multiple of the size of the type. Such aligned memory locations may be faster to read and write than locations at other addresses.

These SIMD data types are not basic Standard C data types or C++ objects, so they may be used only with the assignment operator, passed as function arguments, and returned from a function call. If you access the internal members of these types directly, or indirectly by using them in a union, there may be side effects affecting optimization, so it is recommended to use them only with the SIMD instruction intrinsic functions described in this manual or the Intel C/C++ compiler documentation.

Many intrinsic functions names are prefixed with an indicator of the vector length and suffixed by an indicator of the vector element data type, although some functions do not follow the rules below. The prefixes are:

- _mm_ indicates that the function operates on 128-bit (or sometimes 64-bit) vectors.
- _mm256_ indicates the function operates on 256-bit vectors.
- _mm512_ indicates that the function operates on 512-bit vectors.

The suffixes include:

- _ps, which indicates a function that operates on packed single-precision floating-point data. Packed single-precision floating-point data corresponds to arrays of the C/C++ type *float* with either 4, 8 or 16 elements. Values of this type can be loaded from an array using the *_mm_loadu_ps*, *_mm256_loadu_ps*, or *_mm512_loadu_ps* functions, or created from individual values using *_mm_set_ps*, *_mm256_set_ps*, or *_mm512_set_ps* functions, and they can be stored in an array using *_mm_storeu_ps*, *_mm256_storeu_ps*, or *_mm512_storeu_ps*.
- _ss, which indicates a function that operates on scalar single-precision floating-point data. Single-precision floating-point data corresponds to the C/C++ type *float*, and values of type float can be converted to type __m128 for use with these functions using the *_mm_set_ss* function, and converted back using the *_mm_cvtss_f32* function. When used with functions that operate on packed single-precision floating-point data the scalar element corresponds with the first packed value.
- _pd, which indicates a function that operates on packed double-precision floating-point data. Packed double-precision floating-point data corresponds to arrays of the C/C++ type *double* with either 2, 4, or 8 elements. Values of this type can be loaded from an array using the *_mm_loadu_pd*, *_mm256_loadu_pd*, or *_mm512_loadu_pd* functions, or created from individual values using *_mm_set_pd*, *_mm2566_set_pd*, or *_mm512_set_pd* functions, and they can be stored in an array using *_mm_storeu_pd*, *_mm256_storeu_pd*, or *_mm512_storeu_pd*.
- _sd, which indicates a function that operates on scalar double-precision floating-point data. Double-precision floating-point data corresponds to the C/C++ type *double*, and values of type double can be converted to type __m128d for use with these functions using the *_mm_set_sd* function, and converted back using the *_mm_cvtsd_f64* function. When used with functions that operate on packed double-precision floating-point data the scalar element corresponds with the first packed value.
- _epi8, which indicates a function that operates on packed 8-bit signed integer values. Packed 8-bit signed integers correspond to an array of *signed char* with 16, 32 or 64 elements. Values of this type can be created from individual elements using *_mm_set_epi8*, *_mm256_set_epi8*, or *_mm512_set_epi8* functions.
- _epi16, which indicates a function that operates on packed 16-bit signed integer values. Packed 16-bit signed integers correspond to an array of *short* with 8, 16 or 32 elements. Values of this type can be created from individual elements using *_mm_set_epi16*, *_mm256_set_epi16*, or *_mm512_set_epi16* functions.
- _epi32, which indicates a function that operates on packed 32-bit signed integer values. Packed 32-bit signed integers correspond to an array of *int* with 4, 8 or 16 elements. Values of this type can be created from individual elements using *_mm_set_epi32*, *_mm256_set_epi32*, or *_mm512_set_epi32* functions.

- _epi64, which indicates a function that operates on packed 64-bit signed integer values. Packed 64-bit signed integers correspond to an array of *long long* (or *long* if it is a 64-bit data type) with 2, 4 or 8 elements. Values of this type can be created from individual elements using *_mm_set_epi32*, *_mm256_set_epi32*, or *_mm512_set_epi32* functions.
- _epu8, which indicates a function that operates on packed 8-bit unsigned integer values. Packed 8-bit unsigned integers correspond to an array of *unsigned char* with 16, 32 or 64 elements.
- _epu16, which indicates a function that operates on packed 16-bit unsigned integer values. Packed 16-bit unsigned integers correspond to an array of *unsigned short* with 8, 16 or 32 elements.
- _epu32, which indicates a function that operates on packed 32-bit unsigned integer values. Packed 32-bit unsigned integers correspond to an array of *unsigned* with 4, 8 or 16 elements.
- _epu64, which indicates a function that operates on packed 64-bit unsigned integer values. Packed 64-bit unsigned integers correspond to an array of *unsigned long long* (or *unsigned long* if it is a 64-bit data type) with 2, 4 or 8 elements.
- _si128, which indicates a function that operates on a single 128-bit value of type __m128i.
- _si256, which indicates a function that operates on a single a 256-bit value of type __m256i.
- _si512, which indicates a function that operates on a single a 512-bit value of type __m512i.

Values of any packed integer type can be loaded from an array using the *_mm_loadu_si128*, *_mm256_loadu_si256*, or *_mm512_loadu_si512* functions, and they can be stored in an array using *_mm_storeu_si128*, *_mm256_storeu_si256*, or *_mm512_storeu_si512*.

These functions and data types are used with the SSE, AVX, and AVX-512 instruction set extension families. In addition there are similar functions that correspond to MMX instructions. These are less frequently used because they require additional state management, and only operate on 64-bit packed integer values.

The declarations of Intel C/C++ compiler intrinsic functions may reference some non-standard data types, such as __int64. The C Standard header *stdint.h* defines similar platform-independent types, and the documentation for that header gives characteristics that apply to corresponding non-standard types according to the following table.

Table 3-3 Standard and Non-standard Data Types

Non-standard Type	Standard Type (from stdint.h)
__int64	int64_t
unsigned __int64	uint64_t
__int32	int32_t
unsigned __int32	uint32_t
__int16	int16_t
unsigned __int16	uint16_t

For a more detailed description of each intrinsic function and additional information related to its usage, refer to the online Intel Intrinsics Guide, https://software.intel.com/sites/landingpage/IntrinsicsGuide.

...

ADD—Add

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
04 *ib*	ADD AL, *imm8*	I	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	I	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	I	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	MI	Valid	Valid	Add *imm8* to *r/m8.*
REX + 80 /0 *ib*	ADD *r/m8*[*], *imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64.*
81 /0 *iw*	ADD *r/m16, imm16*	MI	Valid	Valid	Add *imm16* to *r/m16.*
81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32.*
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	MI	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64.*
83 /0 *ib*	ADD *r/m16, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m16.*
83 /0 *ib*	ADD *r/m32, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m32.*
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64.*
00 /*r*	ADD *r/m8, r8*	MR	Valid	Valid	Add *r8* to *r/m8.*
REX + 00 /*r*	ADD *r/m8*[*], *r8*[*]	MR	Valid	N.E.	Add *r8* to *r/m8.*
01 /*r*	ADD *r/m16, r16*	MR	Valid	Valid	Add *r16* to *r/m16.*
01 /*r*	ADD *r/m32, r32*	MR	Valid	Valid	Add r32 to *r/m32.*
REX.W + 01 /*r*	ADD *r/m64, r64*	MR	Valid	N.E.	Add r64 to *r/m64.*
02 /*r*	ADD *r8, r/m8*	RM	Valid	Valid	Add *r/m8* to *r8.*
REX + 02 /*r*	ADD *r8*[*], *r/m8*[*]	RM	Valid	N.E.	Add *r/m8* to *r8.*
03 /*r*	ADD *r16, r/m16*	RM	Valid	Valid	Add *r/m16* to *r16.*
03 /*r*	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32.*
REX.W + 03 /*r*	ADD *r64, r/m64*	RM	Valid	N.E.	Add *r/m64* to *r64.*

NOTES:

*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (r, w)	ModRM:reg (r)	NA	NA
MI	ModRM:r/m (r, w)	imm8	NA	NA
I	AL/AX/EAX/RAX	imm8	NA	NA

Description

Adds the destination operand (first operand) and the source operand (second operand) and then stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADD instruction performs integer addition. It evaluates the result for both signed and unsigned integer operands and sets the OF and CF flags to indicate a carry (overflow) in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

In 64-bit mode, the instruction’s default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
DEST ← DEST + SRC;
```

Flags Affected

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register 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 but the destination is not a memory operand.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

...

CPUID—CPU Identification

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

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

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

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

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

Table 3-18 shows information returned, depending on the initial value loaded into the EAX register.

Two types of information are returned: basic and extended function information. If a value entered for CPUID.EAX is higher than the maximum input value for basic or extended function for that processor then the data for the highest basic information leaf is returned. For example, using the Intel Core i7 processor, the following is true:

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

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

[1] On Intel 64 processors, CPUID clears the high 32 bits of the RAX/RBX/RCX/RDX registers in all modes.

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

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

See also:

“Serializing Instructions” in Chapter 8, “Multiple-Processor Management,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

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

Table 3-18 Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX	Maximum Input Value for Basic CPUID Information.
	EBX	“Genu”
	ECX	“ntel”
	EDX	“inel”
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-6).
	EBX	Bits 07 - 00: Brand Index. Bits 15 - 08: CLFLUSH line size (Value ∗ 8 = cache line size in bytes; used also by CLFLUSHOPT). Bits 23 - 16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31 - 24: Initial APIC ID.
	ECX	Feature Information (see Figure 3-7 and Table 3-20).
	EDX	Feature Information (see Figure 3-8 and Table 3-21).
		NOTES: * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.1.EDX.HTT[bit 28]= 1.
02H	EAX	Cache and TLB Information (see Table 3-22).
	EBX	Cache and TLB Information.
	ECX	Cache and TLB Information.
	EDX	Cache and TLB Information.
03H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 00 - 31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32 - 63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
CPUID leaves above 2 and below 80000000H are visible only when IA32_MISC_ENABLE[bit 22] has its default value of 0.		
	Deterministic Cache Parameters Leaf	
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX.* See also: “INPUT EAX = 04H: Returns Deterministic Cache Parameters for Each Level” on page 50.
	EAX	Bits 04 - 00: Cache Type Field. 0 = Null - No more caches. 1 = Data Cache. 2 = Instruction Cache. 3 = Unified Cache. 4-31 = Reserved.
		Bits 07 - 05: Cache Level (starts at 1). Bit 08: Self Initializing cache level (does not need SW initialization). Bit 09: Fully Associative cache.
		Bits 13 - 10: Reserved. Bits 25 - 14: Maximum number of addressable IDs for logical processors sharing this cache**, ***. Bits 31 - 26: Maximum number of addressable IDs for processor cores in the physical package**, ****, *****.
	EBX	Bits 11 - 00: L = System Coherency Line Size**. Bits 21 - 12: P = Physical Line partitions**. Bits 31 - 22: W = Ways of associativity**.
	ECX	Bits 31-00: S = Number of Sets**.
	EDX	Bit 00: Write-Back Invalidate/Invalidate. 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 01: Cache Inclusiveness. 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 02: Complex Cache Indexing. 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31 - 03: Reserved = 0.
		NOTES: * If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EAX[4:0] as 0. ** Add one to the return value to get the result. ***The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache. **** The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. ***** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	MONITOR/MWAIT Leaf	
05H	EAX	Bits 15 - 00: Smallest monitor-line size in bytes (default is processor's monitor granularity). Bits 31 - 16: Reserved = 0.
	EBX	Bits 15 - 00: Largest monitor-line size in bytes (default is processor's monitor granularity). Bits 31 - 16: Reserved = 0.
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported. Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled. Bits 31 - 02: Reserved.
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWAIT. Bits 07 - 04: Number of C1* sub C-states supported using MWAIT. Bits 11 - 08: Number of C2* sub C-states supported using MWAIT. Bits 15 - 12: Number of C3* sub C-states supported using MWAIT. Bits 19 - 16: Number of C4* sub C-states supported using MWAIT. Bits 23 - 20: Number of C5* sub C-states supported using MWAIT. Bits 27 - 24: Number of C6* sub C-states supported using MWAIT. Bits 31 - 28: Number of C7* sub C-states supported using MWAIT. **NOTE:** * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
	Thermal and Power Management Leaf	
06H	EAX	Bit 00: Digital temperature sensor is supported if set. Bit 01: Intel Turbo Boost Technology Available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved. Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bit 07: HWP. HWP base registers (IA32_PM_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS) are supported if set. Bit 08: HWP_Notification. IA32_HWP_INTERRUPT MSR is supported if set. Bit 09: HWP_Activity_Window. IA32_HWP_REQUEST[bits 41:32] is supported if set. Bit 10: HWP_Energy_Performance_Preference. IA32_HWP_REQUEST[bits 31:24] is supported if set. Bit 11: HWP_Package_Level_Request. IA32_HWP_REQUEST_PKG MSR is supported if set. Bit 12: Reserved. Bit 13: HDC. HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, IA32_THREAD_STALL MSRs are supported if set. Bits 31 - 15: Reserved.
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor. Bits 31 - 04: Reserved.
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency. Bits 02 - 01: Reserved = 0. Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H). Bits 31 - 04: Reserved = 0.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Reserved = 0.
	Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)	
07H		Sub-leaf 0 (Input ECX = 0). *
	EAX	Bits 31 - 00: Reports the maximum input value for supported leaf 7 sub-leaves.
	EBX	Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: SGX. Supports Intel® Software Guard Extensions (Intel® SGX Extensions) if 1. Bit 03: BMI1. Bit 04: HLE. Bit 05: AVX2. Bit 06: FDP_EXCPTN_ONLY. x87 FPU Data Pointer updated only on x87 exceptions if 1. Bit 07: SMEP. Supports Supervisor-Mode Execution Prevention if 1. Bit 08: BMI2. Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID. If 1, supports INVPCID instruction for system software that manages process-context identifiers. Bit 11: RTM. Bit 12: Supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability if 1. (RDT-M) Bit 13: Deprecates FPU CS and FPU DS values if 1. Bit 14: MPX. Supports Intel® Memory Protection Extensions if 1. Bit 15: Supports Intel® Resource Director Technology (Intel® RDT) Allocation capability if 1. (RDT-A) Bits 17 - 16: Reserved. Bit 18: RDSEED. Bit 19: ADX. Bit 20: SMAP. Supports Supervisor-Mode Access Prevention (and the CLAC/STAC instructions) if 1. Bits 22 - 21: Reserved. Bit 23: CLFLUSHOPT. Bit 24: Reserved. Bit 25: Intel Processor Trace. Bits 28 - 26: Reserved. Bit 29: SHA. supports Intel® Secure Hash Algorithm Extensions (Intel® SHA Extensions) if 1. Bits 31 - 30: Reserved.
	ECX	Bit 00: PREFETCHWT1. Bit 01: Reserved. Bit 02: UMIP. Supports user-mode instruction prevention if 1. Bit 03: PKU. Supports protection keys for user-mode pages if 1. Bit 04: OSPKE. If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions). Bits 21 - 05: Reserved. Bit 22: RDPID. Supports Read Processor ID if 1. Bits 29 - 23: Reserved. Bit 30: SGX_LC. Supports SGX Launch Configuration if 1. Bit 31: Reserved.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Reserved. **NOTE:** * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.
	Direct Cache Access Information Leaf	
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
	Architectural Performance Monitoring Leaf	
0AH	EAX	Bits 07 - 00: Version ID of architectural performance monitoring. Bits 15 - 08: Number of general-purpose performance monitoring counter per logical processor. Bits 23 - 16: Bit width of general-purpose, performance monitoring counter. Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events.
	EBX	Bit 00: Core cycle event not available if 1. Bit 01: Instruction retired event not available if 1. Bit 02: Reference cycles event not available if 1. Bit 03: Last-level cache reference event not available if 1. Bit 04: Last-level cache misses event not available if 1. Bit 05: Branch instruction retired event not available if 1. Bit 06: Branch mispredict retired event not available if 1. Bits 31 - 07: Reserved = 0.
	ECX	Reserved = 0.
	EDX	Bits 04 - 00: Number of fixed-function performance counters (if Version ID > 1). Bits 12 - 05: Bit width of fixed-function performance counters (if Version ID > 1). Reserved = 0.
	Extended Topology Enumeration Leaf	
0BH		**NOTES:** Most of Leaf 0BH output depends on the initial value in ECX. The EDX output of leaf 0BH is always valid and does not vary with input value in ECX. Output value in ECX[7:0] always equals input value in ECX[7:0]. For sub-leaves that return an invalid level-type of 0 in ECX[15:8]; EAX and EBX will return 0. If an input value n in ECX returns the invalid level-type of 0 in ECX[15:8], other input values with ECX > n also return 0 in ECX[15:8].
	EAX	Bits 04 - 00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31 - 05: Reserved.
	EBX	Bits 15 - 00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31- 16: Reserved.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 07 - 00: Level number. Same value in ECX input. Bits 15 - 08: Level type***. Bits 31 - 16: Reserved.
	EDX	Bits 31- 00: x2APIC ID the current logical processor.
		NOTES: * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0: Invalid. 1: SMT. 2: Core. 3-255: Reserved.
	Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)	
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	Bits 31 - 00: Reports the supported bits of the lower 32 bits of XCR0. XCR0[n] can be set to 1 only if EAX[n] is 1. Bit 00: x87 state. Bit 01: SSE state. Bit 02: AVX state. Bits 04 - 03: MPX state. Bits 07 - 05: AVX-512 state. Bit 08: Used for IA32_XSS. Bit 09: PKRU state. Bits 31 - 10: Reserved.
	EBX	Bits 31 - 00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.
	ECX	Bit 31 - 00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e all the valid bit fields in XCR0.
	EDX	Bit 31 - 00: Reports the supported bits of the upper 32 bits of XCR0. XCR0[n+32] can be set to 1 only if EDX[n] is 1. Bits 31 - 00: Reserved.
	Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)	
0DH	EAX	Bit 00: XSAVEOPT is available. Bit 01: Supports XSAVEC and the compacted form of XRSTOR if set. Bit 02: Supports XGETBV with ECX = 1 if set. Bit 03: Supports XSAVES/XRSTORS and IA32_XSS if set. Bits 31 - 04: Reserved.
	EBX	Bits 31 - 00: The size in bytes of the XSAVE area containing all states enabled by XCRO | IA32_XSS.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 31 - 00: Reports the supported bits of the lower 32 bits of the IA32_XSS MSR. IA32_XSS[n] can be set to 1 only if ECX[n] is 1. Bits 07 - 00: Used for XCR0. Bit 08: PT state. Bit 09: Used for XCR0. Bits 31 - 10: Reserved.
	EDX	Bits 31 - 00: Reports the supported bits of the upper 32 bits of the IA32_XSS MSR. IA32_XSS[n+32] can be set to 1 only if EDX[n] is 1. Bits 31 - 00: Reserved.
	Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)	
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. Each sub-leaf index (starting at position 2) is supported if it corresponds to a supported bit in either the XCR0 register or the IA32_XSS MSR. * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n (0 ≤ n ≤ 31) is invalid if sub-leaf 0 returns 0 in EAX[n] and sub-leaf 1 returns 0 in ECX[n]. Sub-leaf n (32 ≤ n ≤ 63) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32].
	EAX	Bits 31 - 0: The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index, *n*.
	EBX	Bits 31 - 0: The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, n, does not map to a valid bit in the XCR0 register*.
	ECX	Bit 00 is set if the bit n (corresponding to the sub-leaf index) is supported in the IA32_XSS MSR; it is clear if bit n is instead supported in XCR0. Bit 01 is set if, when the compacted format of an XSAVE area is used, this extended state component located on the next 64-byte boundary following the preceding state component (otherwise, it is located immediately following the preceding state component). Bits 31 - 02 are reserved. This field reports 0 if the sub-leaf index, n, is invalid*.
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	Intel Resource Director Technology (Intel RDT) Monitoring Enumeration Sub-leaf (EAX = 0FH, ECX = 0)	
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.
	EAX	Reserved.
	EBX	Bits 31 - 00: Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.
	EDX	Bit 00: Reserved. Bit 01: Supports L3 Cache Intel RDT Monitoring if 1. Bits 31 - 02: Reserved.
	L3 Cache Intel RDT Monitoring Capability Enumeration Sub-leaf (EAX = 0FH, ECX = 1)	
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EAX	Reserved.
	EBX	Bits 31 - 00: Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes).
	ECX	Maximum range (zero-based) of RMID of this resource type.
	EDX	Bit 00: Supports L3 occupancy monitoring if 1. Bit 01: Supports L3 Total Bandwidth monitoring if 1. Bit 02: Supports L3 Local Bandwidth monitoring if 1. Bits 31 - 03: Reserved.
	Intel Resource Director Technology (Intel RDT) Allocation Enumeration Sub-leaf (EAX = 10H, ECX = 0)	
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX.
	EAX	Reserved.
	EBX	Bit 00: Reserved. Bit 01: Supports L3 Cache Allocation Technology if 1. Bit 02: Supports L2 Cache Allocation Technology if 1. Bits 31 - 03: Reserved.
	ECX	Reserved.
	EDX	Reserved.
	L3 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID =1)	
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 4 - 00: Length of the capacity bit mask for the corresponding ResID using minus-one notation. Bits 31 - 05: Reserved.
	EBX	Bits 31 - 00: Bit-granular map of isolation/contention of allocation units.
	ECX	Bit 00: Reserved. Bit 01: Updates of COS should be infrequent if 1. Bit 02: Code and Data Prioritization Technology supported if 1. Bits 31 - 03: Reserved.
	EDX	Bits 15 - 00: Highest COS number supported for this ResID. Bits 31 - 16: Reserved.
	L2 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID =2)	
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 4 - 00: Length of the capacity bit mask for the corresponding ResID using minus-one notation. Bits 31 - 05: Reserved.
	EBX	Bits 31 - 00: Bit-granular map of isolation/contention of allocation units.
	ECX	Bits 31 - 00: Reserved.
	EDX	Bits 15 - 00: Highest COS number supported for this ResID. Bits 31 - 16: Reserved.
	Intel SGX Capability Enumeration Leaf, sub-leaf 0 (EAX = 12H, ECX = 0)	

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
12H		**NOTES:** Leaf 12H sub-leaf 0 (ECX = 0) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 00: SGX1. If 1, Indicates Intel SGX supports the collection of SGX1 leaf functions. Bit 01: SGX2. If 1, Indicates Intel SGX supports the collection of SGX2 leaf functions. Bit 31 - 02: Reserved.
	EBX	Bit 31 - 00: MISCSELECT. Bit vector of supported extended SGX features.
	ECX	Bit 31 - 00: Reserved.
	EDX	Bit 07 - 00: MaxEnclaveSize_Not64. The maximum supported enclave size in non-64-bit mode is 2^(EDX[7:0]). Bit 15 - 08: MaxEnclaveSize_64. The maximum supported enclave size in 64-bit mode is 2^(EDX[15:8]). Bits 31 - 16: Reserved.
	Intel SGX Attributes Enumeration Leaf, sub-leaf 1 (EAX = 12H, ECX = 1)	
12H		**NOTES:** Leaf 12H sub-leaf 1 (ECX = 1) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE.
	EBX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE.
	ECX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE.
	EDX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.
	Intel SGX EPC Enumeration Leaf, sub-leaves (EAX = 12H, ECX = 2 or higher)	
12H		**NOTES:** Leaf 12H sub-leaf 2 or higher (ECX >= 2) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1. For sub-leaves (ECX = 2 or higher), definition of EDX,ECX,EBX,EAX[31:4] depends on the sub-leaf type listed below.
	EAX	Bit 03 - 00: Sub-leaf Type 0000b: Indicates this sub-leaf is invalid. 0001b: This sub-leaf enumerates an EPC section. EBX:EAX and EDX:ECX provide information on the Enclave Page Cache (EPC) section. All other type encodings are reserved.
	Type	0000b. This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	Type	0001b. This sub-leaf enumerates an EPC sections with EDX:ECX, EBX:EAX defined as follows. EAX[11:04]: Reserved (enumerate 0). EAX[31:12]: Bits 31:12 of the physical address of the base of the EPC section. EBX[19:00]: Bits 51:32 of the physical address of the base of the EPC section. EBX[31:20]: Reserved. ECX[03:00]: EPC section property encoding defined as follows: If EAX[3:0] 0000b, then all bits of the EDX:ECX pair are enumerated as 0. If EAX[3:0] 0001b, then this section has confidentiality and integrity protection. All other encodings are reserved. ECX[11:04]: Reserved (enumerate 0). ECX[31:12]: Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[19:00]: Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[31:20]: Reserved.
	Intel Processor Trace Enumeration Main Leaf (EAX = 14H, ECX = 0)	
14H		**NOTES:** Leaf 14H main leaf (ECX = 0).
	EAX	Bits 31 - 00: Reports the maximum sub-leaf supported in leaf 14H.
	EBX	Bit 00: If 1, Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed. Bit 01: If 1, Indicates support of Configurable PSB and Cycle-Accurate Mode. Bit 02: If 1, Indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset. Bit 03: If 1, Indicates support of MTC timing packet and suppression of COFI-based packets. Bit 31 - 04: Reserved.
	ECX	Bit 00: If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. Bit 01: If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. Bit 02: If 1, Indicates support of Single-Range Output scheme. Bit 03: If 1, Indicates support of output to Trace Transport subsystem. Bit 30 - 04: Reserved. Bit 31: If 1, Generated packets which contain IP payloads have LIP values, which include the CS base component.
	EDX	Bits 31 - 00: Reserved.
	Intel Processor Trace Enumeration Sub-leaf (EAX = 14H, ECX = 1)	
14H	EAX	Bits 02 - 00: Number of configurable Address Ranges for filtering. Bits 15 - 03: Reserved. Bits 31 - 16: Bitmap of supported MTC period encodings.
	EBX	Bits 15 - 00: Bitmap of supported Cycle Threshold value encodings. Bit 31 - 16: Bitmap of supported Configurable PSB frequency encodings.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 31 - 00: Reserved.
	EDX	Bits 31 - 00: Reserved.
	Time Stamp Counter/Core Crystal Clock Information-leaf	
15H		**NOTES:** If EBX[31:0] is 0, the TSC/”core crystal clock” ratio is not enumerated. EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency. “TSC frequency” = “core crystal clock frequency” * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	EAX	Bits 31 - 00: An unsigned integer which is the denominator of the TSC/”core crystal clock” ratio.
	EBX	Bits 31 - 00: An unsigned integer which is the numerator of the TSC/”core crystal clock” ratio.
	ECX	Bits 31 - 00: Reserved = 0.
	EDX	Bits 31 - 00: Reserved = 0.
	Processor Frequency Information Leaf	
16H	EAX	Bits 15 - 00: Processor Base Frequency (in MHz). Bits 31 - 16: Reserved =0.
	EBX	Bits 15 - 00: Maximum Frequency (in MHz). Bits 31 - 16: Reserved = 0.
	ECX	Bits 15 - 00: Bus (Reference) Frequency (in MHz). Bits 31 - 16: Reserved = 0.
	EDX	Reserved.
		NOTES: * Data is returned from this interface in accordance with the processor’s specification and does not reflect actual values. Suitable use of this data includes the display of processor information in like manner to the processor brand string and for determining the appropriate range to use when displaying processor information e.g. frequency history graphs. The returned information should not be used for any other purpose as the returned information does not accurately correlate to information / counters returned by other processor interfaces. While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.
	System-On-Chip Vendor Attribute Enumeration Main Leaf (EAX = 17H, ECX = 0)	
17H		**NOTES:** Leaf 17H main leaf (ECX = 0). Leaf 17H output depends on the initial value in ECX. Leaf 17H sub-leaves 1 through 3 reports SOC Vendor Brand String. Leaf 17H is valid if MaxSOCID_Index >= 3. Leaf 17H sub-leaves 4 and above are reserved.
	EAX	Bits 31 - 00: MaxSOCID_Index. Reports the maximum input value of supported sub-leaf in leaf 17H.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EBX	Bits 15 - 00: SOC Vendor ID. Bit 16: IsVendorScheme. If 1, the SOC Vendor ID field is assigned via an industry standard enumeration scheme. Otherwise, the SOC Vendor ID field is assigned by Intel. Bits 31 - 17: Reserved = 0.
	ECX	Bits 31 - 00: Project ID. A unique number an SOC vendor assigns to its SOC projects.
	EDX	Bits 31 - 00: Stepping ID. A unique number within an SOC project that an SOC vendor assigns.
	System-On-Chip Vendor Attribute Enumeration Sub-leaf (EAX = 17H, ECX = 1..3)	
17H	EAX	Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string.
	EBX	Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string.
	ECX	Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string.
	EDX	Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string.
		NOTES: Leaf 17H output depends on the initial value in ECX. SOC Vendor Brand String is a UTF-8 encoded string padded with trailing bytes of 00H. The complete SOC Vendor Brand String is constructed by concatenating in ascending order of EAX:EBX:ECX:EDX and from the sub-leaf 1 fragment towards sub-leaf 3.
	System-On-Chip Vendor Attribute Enumeration Sub-leaves (EAX = 17H, ECX > MaxSOCID_Index)	
17H		**NOTES:** Leaf 17H output depends on the initial value in ECX.
	EAX	Bits 31 - 00: Reserved = 0.
	EBX	Bits 31 - 00: Reserved = 0.
	ECX	Bits 31 - 00: Reserved = 0.
	EDX	Bits 31 - 00: Reserved = 0.
	Unimplemented CPUID Leaf Functions	
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information.
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode. Bits 04 - 01: Reserved. Bit 05: LZCNT. Bits 07 - 06: Reserved. Bit 08: PREFETCHW. Bits 31 - 09: Reserved.
	EDX	Bits 10 - 00: Reserved. Bit 11: SYSCALL/SYSRET available in 64-bit mode. Bits 19 - 12: Reserved = 0. Bit 20: Execute Disable Bit available. Bits 25 - 21: Reserved = 0. Bit 26: 1-GByte pages are available if 1. Bit 27: RDTSCP and IA32_TSC_AUX are available if 1. Bit 28: Reserved = 0. Bit 29: Intel® 64 Architecture available if 1. Bits 31 - 30: Reserved = 0.
80000002H	EAX	Processor Brand String.
	EBX	Processor Brand String Continued.
	ECX	Processor Brand String Continued.
	EDX	Processor Brand String Continued.
80000003H	EAX	Processor Brand String Continued.
	EBX	Processor Brand String Continued.
	ECX	Processor Brand String Continued.
	EDX	Processor Brand String Continued.
80000004H	EAX	Processor Brand String Continued.
	EBX	Processor Brand String Continued.
	ECX	Processor Brand String Continued.
	EDX	Processor Brand String Continued.
80000005H	EAX	Reserved = 0.
	EBX	Reserved = 0.
	ECX	Reserved = 0.
	EDX	Reserved = 0.
80000006H	EAX	Reserved = 0.
	EBX	Reserved = 0.
	ECX	Bits 07 - 00: Cache Line size in bytes. Bits 11 - 08: Reserved. Bits 15 - 12: L2 Associativity field *. Bits 31 - 16: Cache size in 1K units.
	EDX	Reserved = 0.

Table 3-18 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		NOTES: * L2 associativity field encodings: 00H - Disabled. 01H - Direct mapped. 02H - 2-way. 04H - 4-way. 06H - 8-way. 08H - 16-way. 0FH - Fully associative.
80000007H	EAX	Reserved = 0.
	EBX	Reserved = 0.
	ECX	Reserved = 0.
	EDX	Bits 07 - 00: Reserved = 0. Bit 08: Invariant TSC available if 1. Bits 31 - 09: Reserved = 0.
80000008H	EAX	Linear/Physical Address size. Bits 07 - 00: #Physical Address Bits*. Bits 15 - 08: #Linear Address Bits. Bits 31 - 16: Reserved = 0.
	EBX	Reserved = 0.
	ECX	Reserved = 0.
	EDX	Reserved = 0.
		NOTES: * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

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

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

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

```
EBX ← 756e6547h (* ”Genu”, with G in the low eight bits of BL *)
EDX ← 49656e69h (* ”inel”, with i in the low eight bits of DL *)
ECX ← 6c65746eh (* ”ntel”, with n in the low eight bits of CL *)
```

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

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

IA32_BIOS_SIGN_ID Returns Microcode Update Signature

For processors that support the microcode update facility, the IA32_BIOS_SIGN_ID MSR is loaded with the update signature whenever CPUID executes. The signature is returned in the upper DWORD. For details, see Chapter 9 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

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

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

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

See Table 3-19 for available processor type values. Stepping IDs are provided as needed.

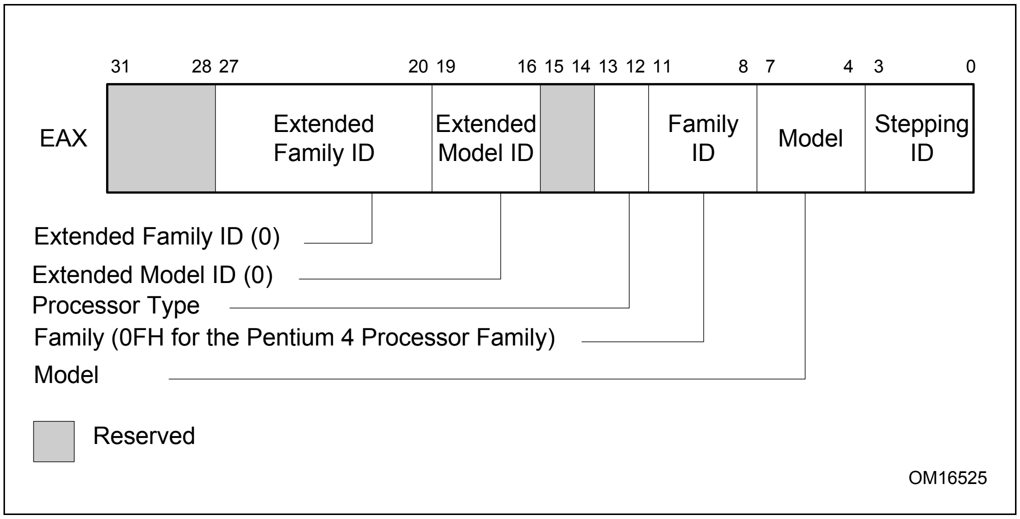


Figure 3-6 Version Information Returned by CPUID in EAX

Table 3-19 Processor Type Field

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

NOTE

See Chapter 18 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for information on identifying earlier IA-32 processors.

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

```
IF Family_ID ≠ 0FH
    THEN DisplayFamily = Family_ID;
    ELSE DisplayFamily = Extended_Family_ID + Family_ID;
    (* Right justify and zero-extend 4-bit field. *)
FI;
(* Show DisplayFamily as HEX field. *)
```

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

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

INPUT EAX = 01H: Returns Additional Information in EBX

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

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

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

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

- Figure 3-7 and Table 3-20 show encodings for ECX.
- Figure 3-8 and Table 3-21 show encodings for EDX.

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

NOTE

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

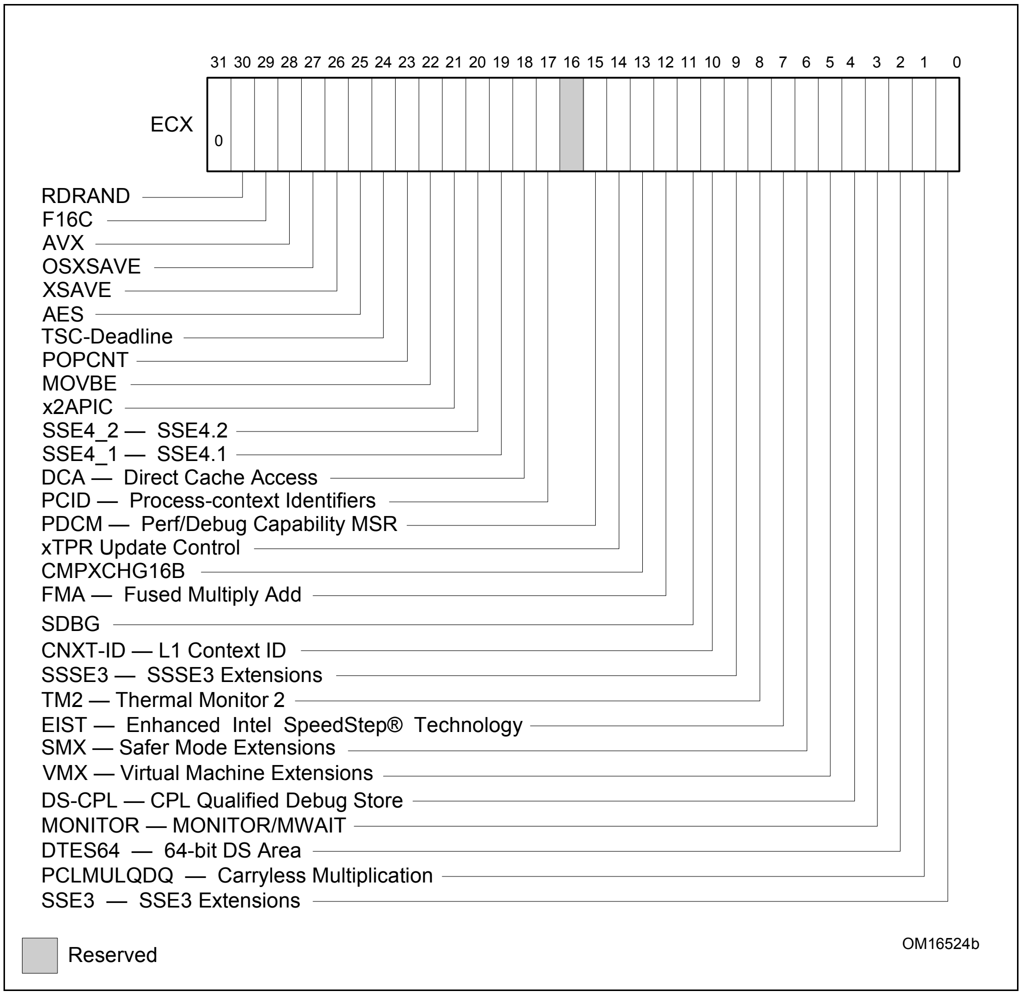


Figure 3-7 Feature Information Returned in the ECX Register

Table 3-20 Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Streaming SIMD Extensions 3 (SSE3).** A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	**PCLMULQDQ.** A value of 1 indicates the processor supports the PCLMULQDQ instruction.
2	DTES64	**64-bit DS Area.** A value of 1 indicates the processor supports DS area using 64-bit layout.
3	MONITOR	**MONITOR/MWAIT.** A value of 1 indicates the processor supports this feature.
4	DS-CPL	**CPL Qualified Debug Store.** A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	**Virtual Machine Extensions.** A value of 1 indicates that the processor supports this technology.
6	SMX	**Safer Mode Extensions.** A value of 1 indicates that the processor supports this technology. See Chapter 5, "Safer Mode Extensions Reference".

Table 3-20 Feature Information Returned in the ECX Register (Contd.)

Bit #	Mnemonic	Description
7	EIST	**Enhanced Intel SpeedStep® technology.** A value of 1 indicates that the processor supports this technology.
8	TM2	**Thermal Monitor 2.** A value of 1 indicates whether the processor supports this technology.
9	SSSE3	A value of 1 indicates the presence of the Supplemental Streaming SIMD Extensions 3 (SSSE3). A value of 0 indicates the instruction extensions are not present in the processor.
10	CNXT-ID	**L1 Context ID.** A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
11	SDBG	A value of 1 indicates the processor supports IA32_DEBUG_INTERFACE MSR for silicon debug.
12	FMA	A value of 1 indicates the processor supports FMA extensions using YMM state.
13	CMPXCHG16B	**CMPXCHG16B Available.** A value of 1 indicates that the feature is available. See the "CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes" section in this chapter for a description.
14	xTPR Update Control	**xTPR Update Control.** A value of 1 indicates that the processor supports changing IA32_MISC_ENABLE[bit 23].
15	PDCM	**Perfmon and Debug Capability:** A value of 1 indicates the processor supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.
16	Reserved	Reserved
17	PCID	**Process-context identifiers.** A value of 1 indicates that the processor supports PCIDs and that software may set CR4.PCIDE to 1.
18	DCA	A value of 1 indicates the processor supports the ability to prefetch data from a memory mapped device.
19	SSE4.1	A value of 1 indicates that the processor supports SSE4.1.
20	SSE4.2	A value of 1 indicates that the processor supports SSE4.2.
21	x2APIC	A value of 1 indicates that the processor supports x2APIC feature.
22	MOVBE	A value of 1 indicates that the processor supports MOVBE instruction.
23	POPCNT	A value of 1 indicates that the processor supports the POPCNT instruction.
24	TSC-Deadline	A value of 1 indicates that the processor's local APIC timer supports one-shot operation using a TSC deadline value.
25	AESNI	A value of 1 indicates that the processor supports the AESNI instruction extensions.
26	XSAVE	A value of 1 indicates that the processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCR0.
27	OSXSAVE	A value of 1 indicates that the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCR0 and to support processor extended state management using XSAVE/XRSTOR.
28	AVX	A value of 1 indicates the processor supports the AVX instruction extensions.
29	F16C	A value of 1 indicates that processor supports 16-bit floating-point conversion instructions.
30	RDRAND	A value of 1 indicates that processor supports RDRAND instruction.
31	Not Used	Always returns 0.

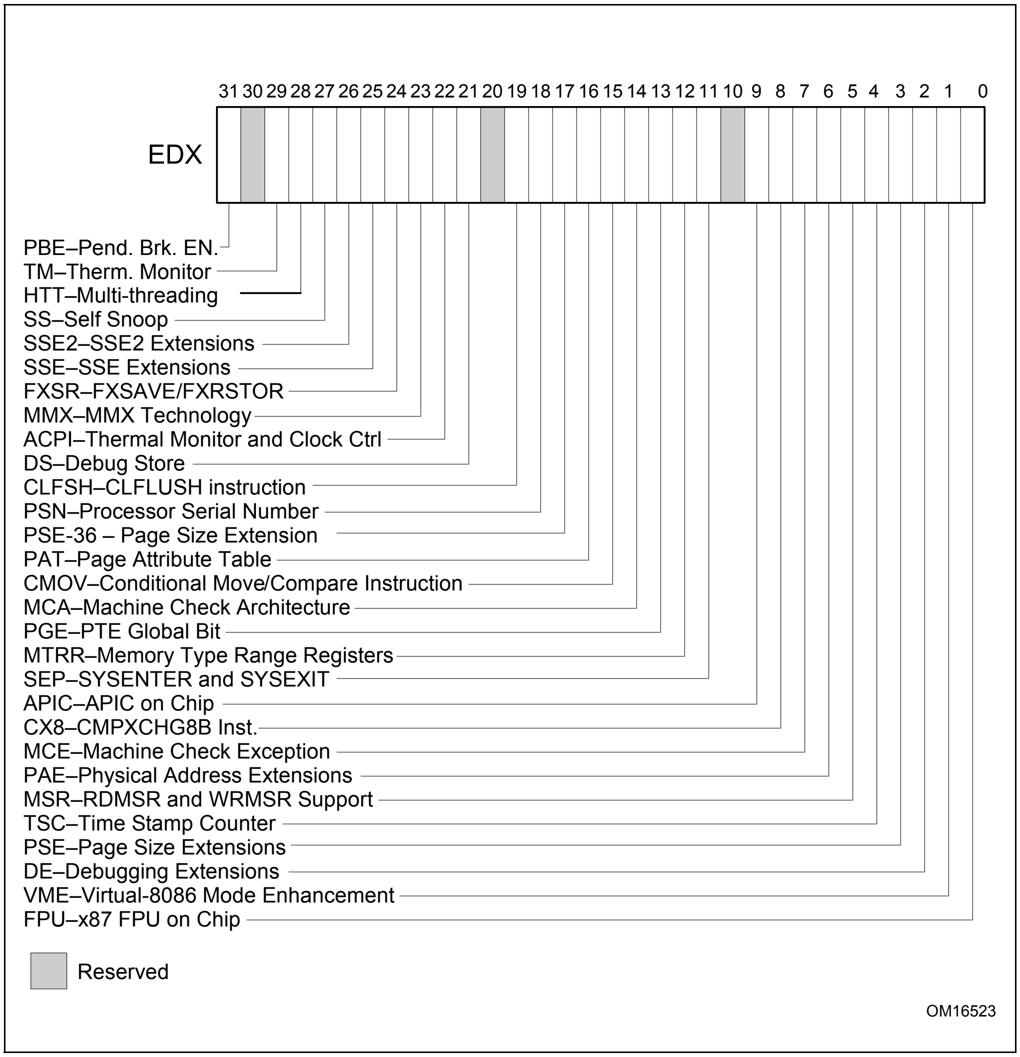


Figure 3-8 Feature Information Returned in the EDX Register

Table 3-21 More on Feature Information Returned in the EDX Register

Bit #	Mnemonic	Description
0	FPU	**Floating Point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.

Table 3-21 More on Feature Information Returned in the EDX Register (Contd.)

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

Table 3-21 More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
21	DS	**Debug Store.** The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities (see Chapter 23, "Introduction to Virtual-Machine Extensions," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Max APIC IDs reserved field is Valid.** A value of 0 for HTT indicates there is only a single logical processor in the package and software should assume only a single APIC ID is reserved. A value of 1 for HTT indicates the value in CPUID.1.EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package) is valid for the package.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved
31	PBE	**Pending Break Enable.** The processor supports the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

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

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

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors. There are four types of encoding values for the byte descriptor, the encoding type is noted in the second column of Table 3-22. Table 3-22 lists the encoding of these descriptors. Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache, prefetch, or TLB types. The descriptors may appear in any order. Note also a processor may report a general descriptor type (FFH) and not report any byte descriptor of "cache type" via CPUID leaf 2.

Table 3-22 Encoding of CPUID Leaf 2 Descriptors

Value	Type	Description
00H	General	Null descriptor, this byte contains no information
01H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries

Table 3-22 Encoding of CPUID Leaf 2 Descriptors (Contd.)

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

Table 3-22 Encoding of CPUID Leaf 2 Descriptors (Contd.)

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

Table 3-22 Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size
A0H	DTLB	DTLB: 4k pages, fully associative, 32 entries
B0H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries
B1H	TLB	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries
B2H	TLB	Instruction TLB: 4KByte pages, 4-way set associative, 64 entries
B3H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 128 entries
B4H	TLB	Data TLB1: 4 KByte pages, 4-way associative, 256 entries
B5H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 64 entries
B6H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 128 entries
BAH	TLB	Data TLB1: 4 KByte pages, 4-way associative, 64 entries
C0H	TLB	Data TLB: 4 KByte and 4 MByte pages, 4-way associative, 8 entries
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries
C2H	DTLB	DTLB: 4 KByte/2 MByte pages, 4-way associative, 16 entries
C3H	STLB	Shared 2nd-Level TLB: 4 KByte /2 MByte pages, 6-way associative, 1536 entries. Also 1GBbyte pages, 4-way, 16 entries.
C4H	DTLB	DTLB: 2M/4M Byte pages, 4-way associative, 32 entries
CAH	STLB	Shared 2nd-Level TLB: 4 KByte pages, 4-way associative, 512 entries
D0H	Cache	3rd-level cache: 512 KByte, 4-way set associative, 64 byte line size
D1H	Cache	3rd-level cache: 1 MByte, 4-way set associative, 64 byte line size
D2H	Cache	3rd-level cache: 2 MByte, 4-way set associative, 64 byte line size
D6H	Cache	3rd-level cache: 1 MByte, 8-way set associative, 64 byte line size
D7H	Cache	3rd-level cache: 2 MByte, 8-way set associative, 64 byte line size
D8H	Cache	3rd-level cache: 4 MByte, 8-way set associative, 64 byte line size
DCH	Cache	3rd-level cache: 1.5 MByte, 12-way set associative, 64 byte line size
DDH	Cache	3rd-level cache: 3 MByte, 12-way set associative, 64 byte line size
DEH	Cache	3rd-level cache: 6 MByte, 12-way set associative, 64 byte line size
E2H	Cache	3rd-level cache: 2 MByte, 16-way set associative, 64 byte line size
E3H	Cache	3rd-level cache: 4 MByte, 16-way set associative, 64 byte line size
E4H	Cache	3rd-level cache: 8 MByte, 16-way set associative, 64 byte line size
EAH	Cache	3rd-level cache: 12MByte, 24-way set associative, 64 byte line size
EBH	Cache	3rd-level cache: 18MByte, 24-way set associative, 64 byte line size
ECH	Cache	3rd-level cache: 24MByte, 24-way set associative, 64 byte line size
F0H	Prefetch	64-Byte prefetching
F1H	Prefetch	128-Byte prefetching
FFH	General	CPUID leaf 2 does not report cache descriptor information, use CPUID leaf 4 to query cache parameters

Example 3-1 Example of Cache and TLB Interpretation

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

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

Which means:

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

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

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

Software can enumerate the deterministic cache parameters for each level of the cache hierarchy starting with an index value of 0, until the parameters report the value associated with the cache type field is 0. The architecturally defined fields reported by deterministic cache parameters are documented in Table 3-18.

This Cache Size in Bytes

= (Ways + 1) * (Partitions + 1) * (Line_Size + 1) * (Sets + 1)

= (EBX[31:22] + 1) * (EBX[21:12] + 1) * (EBX[11:0] + 1) * (ECX + 1)

The CPUID leaf 04H also reports data that can be used to derive the topology of processor cores in a physical package. This information is constant for all valid index values. Software can query the raw data reported by executing CPUID with EAX=04H and ECX=0 and use it as part of the topology enumeration algorithm described in Chapter 8, “Multiple-Processor Management,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

INPUT EAX = 05H: Returns MONITOR and MWAIT Features

When CPUID executes with EAX set to 05H, the processor returns information about features available to MONITOR/MWAIT instructions. The MONITOR instruction is used for address-range monitoring in conjunction with MWAIT instruction. The MWAIT instruction optionally provides additional extensions for advanced power management. See Table 3-18.

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

When CPUID executes with EAX set to 06H, the processor returns information about thermal and power management features. See Table 3-18.

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

When CPUID executes with EAX set to 07H and ECX = 0, the processor returns information about the maximum input value for sub-leaves that contain extended feature flags. See Table 3-18.

When CPUID executes with EAX set to 07H and the input value of ECX is invalid (see leaf 07H entry in Table 3-18), the processor returns 0 in EAX/EBX/ECX/EDX. In subleaf 0, EAX returns the maximum input value of the highest leaf 7 sub-leaf, and EBX, ECX & EDX contain information of extended feature flags.

INPUT EAX = 09H: Returns Direct Cache Access Information

When CPUID executes with EAX set to 09H, the processor returns information about Direct Cache Access capabilities. See Table 3-18.

INPUT EAX = 0AH: Returns Architectural Performance Monitoring Features

When CPUID executes with EAX set to 0AH, the processor returns information about support for architectural performance monitoring capabilities. Architectural performance monitoring is supported if the version ID (see Table 3-18) is greater than Pn 0. See Table 3-18.

For each version of architectural performance monitoring capability, software must enumerate this leaf to discover the programming facilities and the architectural performance events available in the processor. The details are described in Chapter 23, “Introduction to Virtual-Machine Extensions,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*.

INPUT EAX = 0BH: Returns Extended Topology Information

When CPUID executes with EAX set to 0BH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 0BH by verifying (a) the highest leaf index supported by CPUID is >= 0BH, and (b) CPUID.0BH:EBX[15:0] reports a non-zero value. See Table 3-18.

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

When CPUID executes with EAX set to 0DH and ECX = 0, the processor returns information about the bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area. See Table 3-18.

When CPUID executes with EAX set to 0DH and ECX = n (n > 1, and is a valid sub-leaf index), the processor returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. See Table 3-18. Software can use the forward-extendable technique depicted below to query the valid sub-leaves and obtain size and offset information for each processor extended state save area:

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

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

When CPUID executes with EAX set to 0FH and ECX = 0, the processor returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type. See Table 3-18.

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

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

When CPUID executes with EAX set to 10H and ECX = 0, the processor returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type. See Table 3-18.

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

INPUT EAX = 12H: Returns Intel SGX Enumeration Information

When CPUID executes with EAX set to 12H and ECX = 0H, the processor returns information about Intel SGX capabilities. See Table 3-18.

When CPUID executes with EAX set to 12H and ECX = 1H, the processor returns information about Intel SGX attributes. See Table 3-18.

When CPUID executes with EAX set to 12H and ECX = n (n > 1), the processor returns information about Intel SGX Enclave Page Cache. See Table 3-18.

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

When CPUID executes with EAX set to 14H and ECX = 0H, the processor returns information about Intel Processor Trace extensions. See Table 3-18.

When CPUID executes with EAX set to 14H and ECX = n (n > 0 and less than the number of non-zero bits in CPUID.(EAX=14H, ECX= 0H).EAX), the processor returns information about packet generation in Intel Processor Trace. See Table 3-18.

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

When CPUID executes with EAX set to 15H and ECX = 0H, the processor returns information about Time Stamp Counter/Core Crystal Clock. See Table 3-18.

INPUT EAX = 16H: Returns Processor Frequency Information

When CPUID executes with EAX set to 16H, the processor returns information about Processor Frequency Information. See Table 3-18.

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

When CPUID executes with EAX set to 17H, the processor returns information about the System-On-Chip Vendor Attribute Enumeration. See Table 3-18.

METHODS FOR RETURNING BRANDING INFORMATION

Use the following techniques to access branding information:

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

These two methods are discussed in the following sections. For methods that are available in early processors, see Section: “Identification of Earlier IA-32 Processors” in Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

The Processor Brand String Method

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

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

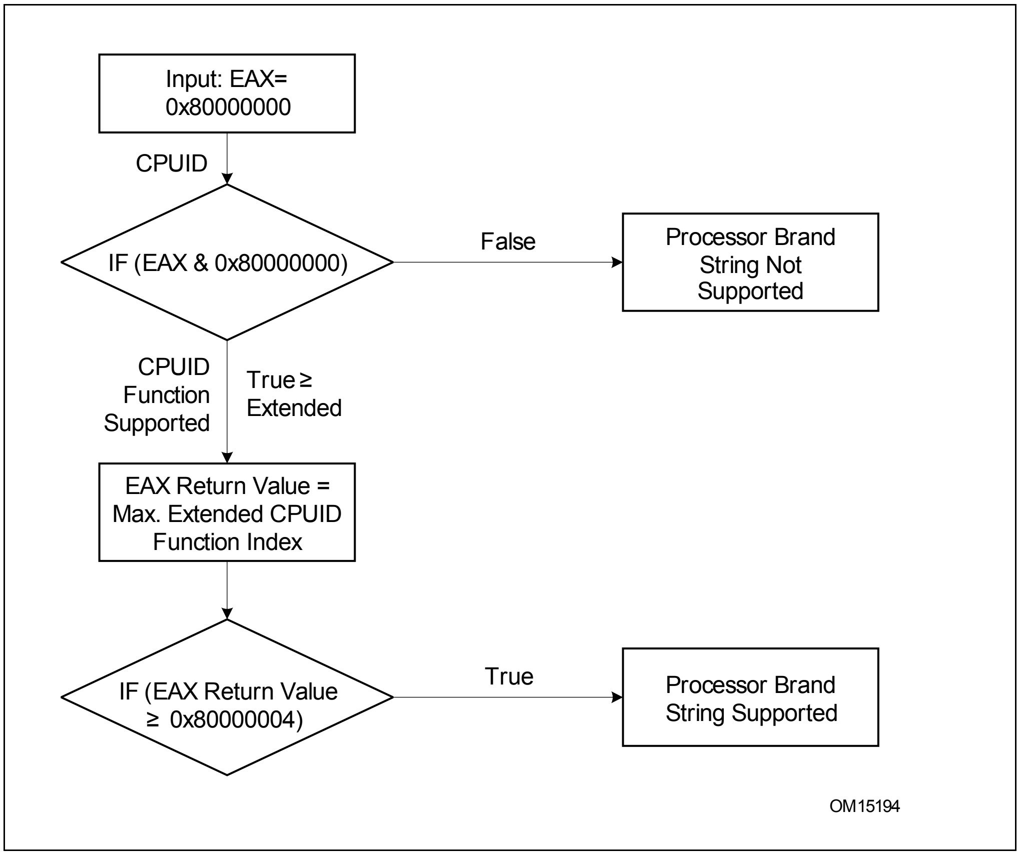

Input: EAX= 0x80000000
CPUID
IF (EAX & 0x80000000)
False
Processor Brand String Not Supported
CPUID Function Supported
True ≥ Extended
EAX Return Value = Max. Extended CPUID Function Index
IF (EAX Return Value ≥ 0x80000004)
True
Processor Brand String Supported
OM15194

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

How Brand Strings Work

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

Table 3-23 shows the brand string that is returned by the first processor in the Pentium 4 processor family.

Table 3-23 Processor Brand String Returned with Pentium 4 Processor

EAX Input Value	Return Values	ASCII Equivalent

80000002H	EAX = 20202020H
	EBX = 20202020H
	ECX = 20202020H
	EDX = 6E492020H
		“ ”
		“ ”
		“ ”
		“nI ”

80000003H	EAX = 286C6574H
	EBX = 50202952H
	ECX = 69746E65H
	EDX = 52286D75H
		“(let”
		“P)R”
		“itne”
		“R(mu”

80000004H	EAX = 20342029H
	EBX = 20555043H
	ECX = 30303531H
	EDX = 007A484DH
		“ 4)”
		“ UPC”
		“0051”
		“\0zHM”

Extracting the Processor Frequency from Brand Strings

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

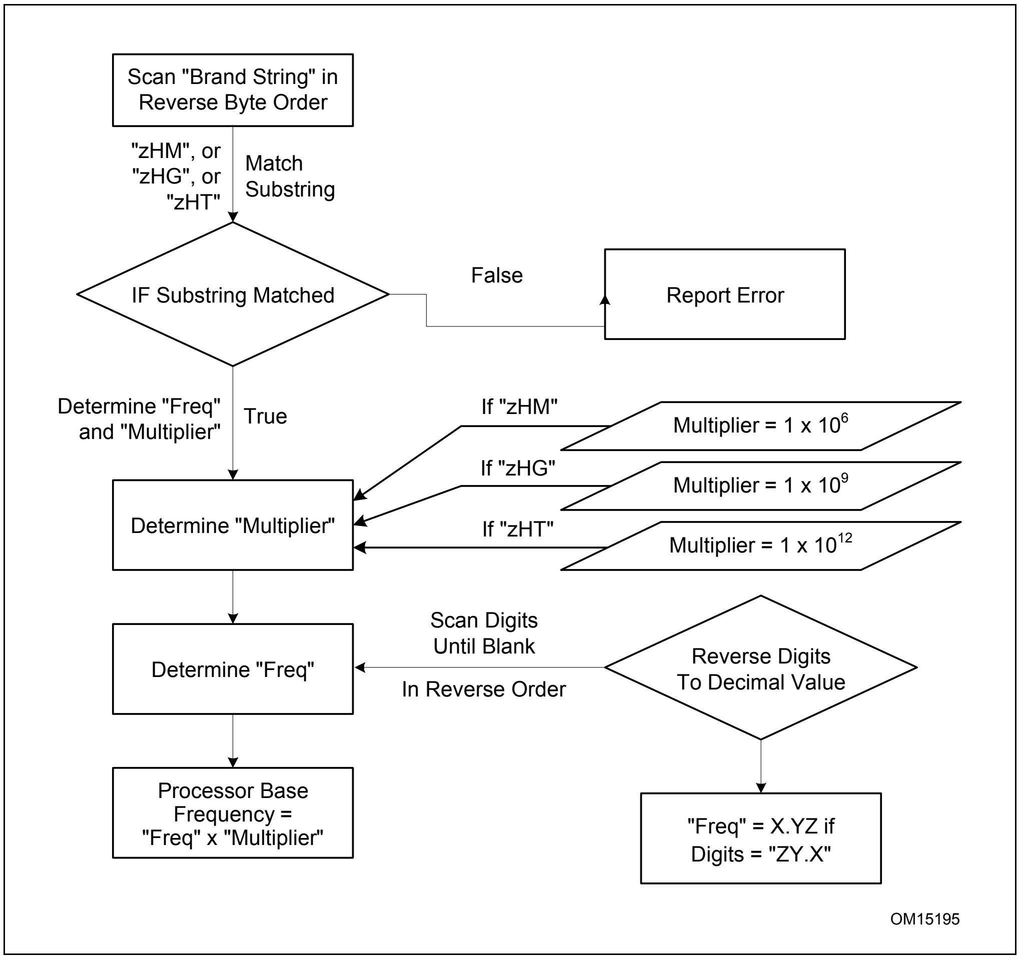


Figure 3-10 Algorithm for Extracting Processor Frequency

The Processor Brand Index Method

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

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

Table 3-24 shows brand indices that have identification strings associated with them.

Table 3-24 Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

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

NOTES:

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

IA-32 Architecture Compatibility

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

Operation

```
IA32_BIOS_SIGN_ID MSR ← Update with installed microcode revision number;

CASE (EAX) OF
    EAX = 0:
        EAX ← Highest basic function input value understood by CPUID;
        EBX ← Vendor identification string;
        EDX ← Vendor identification string;
        ECX ← Vendor identification string;
```

```
BREAK;
EAX = 1H:
    EAX[3:0] ← Stepping ID;
    EAX[7:4] ← Model;
    EAX[11:8] ← Family;
    EAX[13:12] ← Processor type;
    EAX[15:14] ← Reserved;
    EAX[19:16] ← Extended Model;
    EAX[27:20] ← Extended Family;
    EAX[31:28] ← Reserved;
    EBX[7:0] ← Brand Index; (* Reserved if the value is zero. *)
    EBX[15:8] ← CLFLUSH Line Size;
    EBX[16:23] ← Reserved; (* Number of threads enabled = 2 if MT enable fuse set. *)
    EBX[24:31] ← Initial APIC ID;
    ECX ← Feature flags; (* See Figure 3-7. *)
    EDX ← Feature flags; (* See Figure 3-8. *)
BREAK;
EAX = 2H:
    EAX ← Cache and TLB information;
    EBX ← Cache and TLB information;
    ECX ← Cache and TLB information;
    EDX ← Cache and TLB information;
BREAK;
EAX = 3H:
    EAX ← Reserved;
    EBX ← Reserved;
    ECX ← ProcessorSerialNumber[31:0];
    (* Pentium III processors only, otherwise reserved. *)
    EDX ← ProcessorSerialNumber[63:32];
    (* Pentium III processors only, otherwise reserved. *
BREAK
EAX = 4H:
    EAX ← Deterministic Cache Parameters Leaf; (* See Table 3-18. *)
    EBX ← Deterministic Cache Parameters Leaf;
    ECX ← Deterministic Cache Parameters Leaf;
    EDX ← Deterministic Cache Parameters Leaf;
BREAK;
EAX = 5H:
    EAX ← MONITOR/MWAIT Leaf; (* See Table 3-18. *)
    EBX ← MONITOR/MWAIT Leaf;
    ECX ← MONITOR/MWAIT Leaf;
    EDX ← MONITOR/MWAIT Leaf;
BREAK;
EAX = 6H:
    EAX ← Thermal and Power Management Leaf; (* See Table 3-18. *)
    EBX ← Thermal and Power Management Leaf;
    ECX ← Thermal and Power Management Leaf;
    EDX ← Thermal and Power Management Leaf;
BREAK;
EAX = 7H:
    EAX ← Structured Extended Feature Flags Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Structured Extended Feature Flags Enumeration Leaf;
    ECX ← Structured Extended Feature Flags Enumeration Leaf;
    EDX ← Structured Extended Feature Flags Enumeration Leaf;
BREAK;
EAX = 8H:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Reserved = 0;
    EDX ← Reserved = 0;
BREAK;
EAX = 9H:
    EAX ← Direct Cache Access Information Leaf; (* See Table 3-18. *)
    EBX ← Direct Cache Access Information Leaf;
    ECX ← Direct Cache Access Information Leaf;
    EDX ← Direct Cache Access Information Leaf;
BREAK;
EAX = AH:
    EAX ← Architectural Performance Monitoring Leaf; (* See Table 3-18. *)
    EBX ← Architectural Performance Monitoring Leaf;
    ECX ← Architectural Performance Monitoring Leaf;
    EDX ← Architectural Performance Monitoring Leaf;
    BREAK
EAX = BH:
    EAX ← Extended Topology Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Extended Topology Enumeration Leaf;
    ECX ← Extended Topology Enumeration Leaf;
    EDX ← Extended Topology Enumeration Leaf;
BREAK;
EAX = CH:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Reserved = 0;
    EDX ← Reserved = 0;
BREAK;
EAX = DH:
    EAX ← Processor Extended State Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Processor Extended State Enumeration Leaf;
    ECX ← Processor Extended State Enumeration Leaf;
    EDX ← Processor Extended State Enumeration Leaf;
BREAK;
EAX = EH:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Reserved = 0;
    EDX ← Reserved = 0;
BREAK;
EAX = FH:
    EAX ← Intel Resource Director Technology Monitoring Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Intel Resource Director Technology Monitoring Enumeration Leaf;
    ECX ← Intel Resource Director Technology Monitoring Enumeration Leaf;
    EDX ← Intel Resource Director Technology Monitoring Enumeration Leaf;
BREAK;
EAX = 10H:
    EAX ← Intel Resource Director Technology Allocation Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Intel Resource Director Technology Allocation Enumeration Leaf;
    ECX ← Intel Resource Director Technology Allocation Enumeration Leaf;
    EDX ← Intel Resource Director Technology Allocation Enumeration Leaf;
BREAK;
    EAX = 12H:
    EAX ← Intel SGX Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Intel SGX Enumeration Leaf;
    ECX ← Intel SGX Enumeration Leaf;
    EDX ← Intel SGX Enumeration Leaf;
BREAK;
EAX = 14H:
    EAX ← Intel Processor Trace Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Intel Processor Trace Enumeration Leaf;
    ECX ← Intel Processor Trace Enumeration Leaf;
    EDX ← Intel Processor Trace Enumeration Leaf;
BREAK;
EAX = 15H:
    EAX ← Time Stamp Counter/Core Crystal Clock Information Leaf; (* See Table 3-18. *)
    EBX ← Time Stamp Counter/Core Crystal Clock Information Leaf;
    ECX ← Time Stamp Counter/Core Crystal Clock Information Leaf;
    EDX ← Time Stamp Counter/Core Crystal Clock Information Leaf;
BREAK;
EAX = 16H:
    EAX ← Processor Frequency Information Enumeration Leaf; (* See Table 3-18. *)
    EBX ← Processor Frequency Information Enumeration Leaf;
    ECX ← Processor Frequency Information Enumeration Leaf;
    EDX ← Processor Frequency Information Enumeration Leaf;
BREAK;
EAX = 17H:
    EAX ← System-On-Chip Vendor Attribute Enumeration Leaf; (* See Table 3-18. *)
    EBX ← System-On-Chip Vendor Attribute Enumeration Leaf;
    ECX ← System-On-Chip Vendor Attribute Enumeration Leaf;
    EDX ← System-On-Chip Vendor Attribute Enumeration Leaf;
BREAK;
EAX = 80000000H:
    EAX ← Highest extended function input value understood by CPUID;
    EBX ← Reserved;
    ECX ← Reserved;
    EDX ← Reserved;
BREAK;
EAX = 80000001H:
    EAX ← Reserved;
    EBX ← Reserved;
    ECX ← Extended Feature Bits (* See Table 3-18.*);
    EDX ← Extended Feature Bits (* See Table 3-18. *);
```

```
BREAK;
EAX = 80000002H:
    EAX ← Processor Brand String;
    EBX ← Processor Brand String, continued;
    ECX ← Processor Brand String, continued;
    EDX ← Processor Brand String, continued;
BREAK;
EAX = 80000003H:
    EAX ← Processor Brand String, continued;
    EBX ← Processor Brand String, continued;
    ECX ← Processor Brand String, continued;
    EDX ← Processor Brand String, continued;
BREAK;
EAX = 80000004H:
    EAX ← Processor Brand String, continued;
    EBX ← Processor Brand String, continued;
    ECX ← Processor Brand String, continued;
    EDX ← Processor Brand String, continued;
BREAK;
EAX = 80000005H:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Reserved = 0;
    EDX ← Reserved = 0;
BREAK;
EAX = 80000006H:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Cache information;
    EDX ← Reserved = 0;
BREAK;
EAX = 80000007H:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Reserved = 0;
    EDX ← Reserved = Misc Feature Flags;
BREAK;
EAX = 80000008H:
    EAX ← Reserved = Physical Address Size Information;
    EBX ← Reserved = Virtual Address Size Information;
    ECX ← Reserved = 0;
    EDX ← Reserved = 0;
BREAK;
EAX >= 40000000H and EAX <= 4FFFFFFFH:
DEFAULT: (* EAX = Value outside of recognized range for CPUID. *)
    (* If the highest basic information leaf data depend on ECX input value, ECX is honored.*)
    EAX ← Reserved; (* Information returned for highest basic information leaf. *)
    EBX ← Reserved; (* Information returned for highest basic information leaf. *)
    ECX ← Reserved; (* Information returned for highest basic information leaf. *)
    EDX ← Reserved; (* Information returned for highest basic information leaf. *)
    BREAK;
ESAC;
```

Flags Affected

None.

Exceptions (All Operating Modes)

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

...

FCLEX/FNCLEX—Clear Exceptions

Opcode*	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
9B DB E2	FCLEX	Valid	Valid	Clear floating-point exception flags after checking for pending unmasked floating-point exceptions.
DB E2	FNCLEX*	Valid	Valid	Clear floating-point exception flags without checking for pending unmasked floating-point exceptions.

NOTES:

* See IA-32 Architecture Compatibility section below.

Description

Clears the floating-point exception flags (PE, UE, OE, ZE, DE, and IE), the exception summary status flag (ES), the stack fault flag (SF), and the busy flag (B) in the FPU status word. The FCLEX instruction checks for and handles any pending unmasked floating-point exceptions before clearing the exception flags; the FNCLEX instruction does not.

The assembler issues two instructions for the FCLEX instruction (an FWAIT instruction followed by an FNCLEX instruction), and the processor executes each of these instructions separately. If an exception is generated for either of these instructions, the save EIP points to the instruction that caused the exception.

IA-32 Architecture Compatibility

When operating a Pentium or Intel486 processor in MS-DOS* compatibility mode, it is possible (under unusual circumstances) for an FNCLEX instruction to be interrupted prior to being executed to handle a pending FPU exception. See the section titled "No-Wait FPU Instructions Can Get FPU Interrupt in Window" in Appendix D of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a description of these circumstances. An FNCLEX instruction cannot be interrupted in this way on later Intel processors, except for the Intel Quark™ X1000 processor.

This instruction affects only the x87 FPU floating-point exception flags. It does not affect the SIMD floating-point exception flags in the MXCRS register.

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

Operation

```
FPUStatusWord[0:7] ← 0;
FPUStatusWord[15] ← 0;
```

FPU Flags Affected

The PE, UE, OE, ZE, DE, IE, ES, SF, and B flags in the FPU status word are cleared. The C0, C1, C2, and C3 flags are undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM CR0.EM[bit 2] or CR0.TS[bit 3] = 1.

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

FINIT/FNINIT—Initialize Floating-Point Unit

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
9B DB E3	FINIT	Valid	Valid	Initialize FPU after checking for pending unmasked floating-point exceptions.
DB E3	FNINIT*	Valid	Valid	Initialize FPU without checking for pending unmasked floating-point exceptions.

NOTES:

* See IA-32 Architecture Compatibility section below.

Description

Sets the FPU control, status, tag, instruction pointer, and data pointer registers to their default states. The FPU control word is set to 037FH (round to nearest, all exceptions masked, 64-bit precision). The status word is cleared (no exception flags set, TOP is set to 0). The data registers in the register stack are left unchanged, but they are all tagged as empty (11B). Both the instruction and data pointers are cleared.

The FINIT instruction checks for and handles any pending unmasked floating-point exceptions before performing the initialization; the FNINIT instruction does not.

The assembler issues two instructions for the FINIT instruction (an FWAIT instruction followed by an FNINIT instruction), and the processor executes each of these instructions in separately. If an exception is generated for either of these instructions, the save EIP points to the instruction that caused the exception.

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

IA-32 Architecture Compatibility

When operating a Pentium or Intel486 processor in MS-DOS compatibility mode, it is possible (under unusual circumstances) for an FNINIT instruction to be interrupted prior to being executed to handle a pending FPU exception. See the section titled "No-Wait FPU Instructions Can Get FPU Interrupt in Window" in Appendix D of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a description of these circumstances. An FNINIT instruction cannot be interrupted in this way on later Intel processors, except for the Intel Quark™ X1000 processor.

In the Intel387 math coprocessor, the FINIT/FNINIT instruction does not clear the instruction and data pointers.

This instruction affects only the x87 FPU. It does not affect the XMM and MXCSR registers.

Operation

```
FPUControlWord ← 037FH;
FPUStatusWord ← 0;
FPUTagWord ← FFFFH;
FPUDataPointer ← 0;
FPUInstructionPointer ← 0;
FPULastInstructionOpcode ← 0;
```

FPU Flags Affected

C0, C1, C2, C3 set to 0.

Floating-Point Exceptions

None.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

FSAVE/FNSAVE—Store x87 FPU State

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
9B DD /6	FSAVE *m94/108byte*	Valid	Valid	Store FPU state to *m94byte* or *m108byte* after checking for pending unmasked floating-point exceptions. Then re-initialize the FPU.
DD /6	FNSAVE* *m94/108byte*	Valid	Valid	Store FPU environment to *m94byte* or *m108byte* without checking for pending unmasked floating-point exceptions. Then re-initialize the FPU.

NOTES:

* See IA-32 Architecture Compatibility section below.

Description

Stores the current FPU state (operating environment and register stack) at the specified destination in memory, and then re-initializes the FPU. The FSAVE instruction checks for and handles pending unmasked floating-point exceptions before storing the FPU state; the FNSAVE instruction does not.

The FPU operating environment consists of the FPU control word, status word, tag word, instruction pointer, data pointer, and last opcode. Figures 8-9 through 8-12 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, show the layout in memory of the stored environment, depending on the operating mode of the processor (protected or real) and the current operand-size attribute (16-bit or 32-bit). In virtual-8086 mode, the real mode layouts are used. The contents of the FPU register stack are stored in the 80 bytes immediately follow the operating environment image.

The saved image reflects the state of the FPU after all floating-point instructions preceding the FSAVE/FNSAVE instruction in the instruction stream have been executed.

After the FPU state has been saved, the FPU is reset to the same default values it is set to with the FINIT/FNINIT instructions (see "FINIT/FNINIT—Initialize Floating-Point Unit" in this chapter).

The FSAVE/FNSAVE instructions are typically used when the operating system needs to perform a context switch, an exception handler needs to use the FPU, or an application program needs to pass a "clean" FPU to a procedure.

The assembler issues two instructions for the FSAVE instruction (an FWAIT instruction followed by an FNSAVE instruction), and the processor executes each of these instructions separately. If an exception is generated for either of these instructions, the save EIP points to the instruction that caused the exception.

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

IA-32 Architecture Compatibility

For Intel math coprocessors and FPUs prior to the Intel Pentium processor, an FWAIT instruction should be executed before attempting to read from the memory image stored with a prior FSAVE/FNSAVE instruction. This FWAIT instruction helps ensure that the storage operation has been completed.

When operating a Pentium or Intel486 processor in MS-DOS compatibility mode, it is possible (under unusual circumstances) for an FNSAVE instruction to be interrupted prior to being executed to handle a pending FPU exception. See the section titled "No-Wait FPU Instructions Can Get FPU Interrupt in Window" in Appendix D of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a description of these circumstances. An FNSAVE instruction cannot be interrupted in this way on later Intel processors, except for the Intel Quark^{TM} X1000 processor.

Operation

```
(* Save FPU State and Registers *)

DEST[FPUControlWord] ← FPUControlWord;
DEST[FPUStatusWord] ← FPUStatusWord;
DEST[FPUTagWord] ← FPUTagWord;
DEST[FPUDataPointer] ← FPUDataPointer;
DEST[FPUInstructionPointer] ← FPUInstructionPointer;
DEST[FPULastInstructionOpcode] ← FPULastInstructionOpcode;

DEST[ST(0)] ← ST(0);
DEST[ST(1)] ← ST(1);
DEST[ST(2)] ← ST(2);
DEST[ST(3)] ← ST(3);
DEST[ST(4)]← ST(4);
DEST[ST(5)] ← ST(5);
DEST[ST(6)] ← ST(6);
DEST[ST(7)] ← ST(7);

(* Initialize FPU *)

FPUControlWord ← 037FH;
FPUStatusWord ← 0;
FPUTagWord ← FFFFH;
FPUDataPointer ← 0;
FPUInstructionPointer ← 0;
FPULastInstructionOpcode ← 0;
```

FPU Flags Affected

The C0, C1, C2, and C3 flags are saved and then cleared.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	If destination is located in a non-writable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register 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.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#MF	If there is a pending x87 FPU exception.
#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.

...

FSTCW/FNSTCW—Store x87 FPU Control Word

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
9B D9 /7	FSTCW *m2byte*	Valid	Valid	Store FPU control word to *m2byte* after checking for pending unmasked floating-point exceptions.
D9 /7	FNSTCW* *m2byte*	Valid	Valid	Store FPU control word to *m2byte* without checking for pending unmasked floating-point exceptions.

NOTES:

* See IA-32 Architecture Compatibility section below.

Description

Stores the current value of the FPU control word at the specified destination in memory. The FSTCW instruction checks for and handles pending unmasked floating-point exceptions before storing the control word; the FNSTCW instruction does not.

The assembler issues two instructions for the FSTCW instruction (an FWAIT instruction followed by an FNSTCW instruction), and the processor executes each of these instructions in separately. If an exception is generated for either of these instructions, the save EIP points to the instruction that caused the exception.

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

IA-32 Architecture Compatibility

When operating a Pentium or Intel486 processor in MS-DOS compatibility mode, it is possible (under unusual circumstances) for an FNSTCW instruction to be interrupted prior to being executed to handle a pending FPU exception. See the section titled "No-Wait FPU Instructions Can Get FPU Interrupt in Window" in Appendix D of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 1*, for a description of these circumstances. An FNSTCW instruction cannot be interrupted in this way on later Intel processors, except for the Intel Quark™ X1000 processor.

Operation

```
DEST ← FPUControlWord;
```

FPU Flags Affected

The C0, C1, C2, and C3 flags are undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register 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.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#MF	If there is a pending x87 FPU exception.
#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.

...

FSTENV/FNSTENV—Store x87 FPU Environment

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
9B D9 /6	FSTENV *m14/28byte*	Valid	Valid	Store FPU environment to *m14byte* or *m28byte* after checking for pending unmasked floating-point exceptions. Then mask all floating-point exceptions.
D9 /6	FNSTENV* *m14/28byte*	Valid	Valid	Store FPU environment to *m14byte* or *m28byte* without checking for pending unmasked floating-point exceptions. Then mask all floating-point exceptions.

NOTES:

* See IA-32 Architecture Compatibility section below.

Description

Saves the current FPU operating environment at the memory location specified with the destination operand, and then masks all floating-point exceptions. The FPU operating environment consists of the FPU control word, status word, tag word, instruction pointer, data pointer, and last opcode. Figures 8-9 through 8-12 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, show the layout in memory of the stored environment, depending on the operating mode of the processor (protected or real) and the current operand-size attribute (16-bit or 32-bit). In virtual-8086 mode, the real mode layouts are used.

The FSTENV instruction checks for and handles any pending unmasked floating-point exceptions before storing the FPU environment; the FNSTENV instruction does not. The saved image reflects the state of the FPU after all floating-point instructions preceding the FSTENV/FNSTENV instruction in the instruction stream have been executed.

These instructions are often used by exception handlers because they provide access to the FPU instruction and data pointers. The environment is typically saved in the stack. Masking all exceptions after saving the environment prevents floating-point exceptions from interrupting the exception handler.

The assembler issues two instructions for the FSTENV instruction (an FWAIT instruction followed by an FNSTENV instruction), and the processor executes each of these instructions separately. If an exception is generated for either of these instructions, the save EIP points to the instruction that caused the exception.

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

IA-32 Architecture Compatibility

When operating a Pentium or Intel486 processor in MS-DOS compatibility mode, it is possible (under unusual circumstances) for an FNSTENV instruction to be interrupted prior to being executed to handle a pending FPU exception. See the section titled "No-Wait FPU Instructions Can Get FPU Interrupt in Window" in Appendix D of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 1*, for a description of these circumstances. An FNSTENV instruction cannot be interrupted in this way on later Intel processors, except for the Intel Quark™ X1000 processor.

Operation

```
DEST[FPUControlWord] ← FPUControlWord;
DEST[FPUStatusWord] ← FPUStatusWord;
DEST[FPUTagWord] ← FPUTagWord;
DEST[FPUDataPointer] ← FPUDataPointer;
DEST[FPUInstructionPointer] ← FPUInstructionPointer;
DEST[FPULastInstructionOpcode] ← FPULastInstructionOpcode;
```

FPU Flags Affected

The C0, C1, C2, and C3 are undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register 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.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#MF	If there is a pending x87 FPU exception.
#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.

...

FSTSW/FNSTSW—Store x87 FPU Status Word

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
9B DD /7	FSTSW *m2byte*	Valid	Valid	Store FPU status word at *m2byte* after checking for pending unmasked floating-point exceptions.
9B DF E0	FSTSW AX	Valid	Valid	Store FPU status word in AX register after checking for pending unmasked floating-point exceptions.
DD /7	FNSTSW* *m2byte*	Valid	Valid	Store FPU status word at *m2byte* without checking for pending unmasked floating-point exceptions.
DF E0	FNSTSW* AX	Valid	Valid	Store FPU status word in AX register without checking for pending unmasked floating-point exceptions.

NOTES:

* See IA-32 Architecture Compatibility section below.

Description

Stores the current value of the x87 FPU status word in the destination location. The destination operand can be either a two-byte memory location or the AX register. The FSTSW instruction checks for and handles pending unmasked floating-point exceptions before storing the status word; the FNSTSW instruction does not.

The FNSTSW AX form of the instruction is used primarily in conditional branching (for instance, after an FPU comparison instruction or an FPREM, FPREM1, or FXAM instruction), where the direction of the branch depends on the state of the FPU condition code flags. (See the section titled “Branching and Conditional Moves on FPU Condition Codes” in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.) This instruction can also be used to invoke exception handlers (by examining the exception flags) in environments that do not use interrupts. When the FNSTSW AX instruction is executed, the AX register is updated before the processor executes any further instructions. The status stored in the AX register is thus guaranteed to be from the completion of the prior FPU instruction.

The assembler issues two instructions for the FSTSW instruction (an FWAIT instruction followed by an FNSTSW instruction), and the processor executes each of these instructions separately. If an exception is generated for either of these instructions, the save EIP points to the instruction that caused the exception.

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

IA-32 Architecture Compatibility

When operating a Pentium or Intel486 processor in MS-DOS compatibility mode, it is possible (under unusual circumstances) for an FNSTSW instruction to be interrupted prior to being executed to handle a pending FPU exception. See the section titled “No-Wait FPU Instructions Can Get FPU Interrupt in Window” in Appendix D of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*, for a description of these circumstances. An FNSTSW instruction cannot be interrupted in this way on later Intel processors, except for the Intel Quark^{TM} X1000 processor.

Operation

```
DEST ← FPUStatusWord;
```

FPU Flags Affected

The C0, C1, C2, and C3 are undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register 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.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

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

#NM CR0.EM[bit 2] or CR0.TS[bit 3] = 1.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

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

#NM CR0.EM[bit 2] or CR0.TS[bit 3] = 1.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

#UD If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0) If a memory address referencing the SS segment is in a non-canonical form.

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

#NM CR0.EM[bit 2] or CR0.TS[bit 3] = 1.

#MF If there is a pending x87 FPU exception.

#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.

...

IMUL—Signed Multiply

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F6 /5	IMUL *r/m8**	M	Valid	Valid	AX← AL ∗ *r/m* byte.
F7 /5	IMUL *r/m16*	M	Valid	Valid	DX:AX ← AX ∗ *r/m* word.
F7 /5	IMUL *r/m32*	M	Valid	Valid	EDX:EAX ← EAX ∗ *r/m*32.
REX.W + F7 /5	IMUL *r/m64*	M	Valid	N.E.	RDX:RAX ← RAX ∗ *r/m*64.
0F AF /*r*	IMUL *r16, r/m16*	RM	Valid	Valid	word register ← word register ∗ *r/m*16.
0F AF /*r*	IMUL *r32, r/m32*	RM	Valid	Valid	doubleword register ← doubleword register ∗ *r/m32.*
REX.W + 0F AF /*r*	IMUL *r64, r/m64*	RM	Valid	N.E.	Quadword register ← Quadword register ∗ *r/m64*.
6B /*r ib*	IMUL *r16, r/m16, imm8*	RMI	Valid	Valid	word register ← *r/m16* ∗ sign-extended immediate byte.
6B /*r ib*	IMUL *r32, r/m32, imm8*	RMI	Valid	Valid	doubleword register ← *r/m32* ∗ sign-extended immediate byte.
REX.W + 6B /*r ib*	IMUL *r64, r/m64, imm8*	RMI	Valid	N.E.	Quadword register ← *r/m64* ∗ sign-extended immediate byte.
69 /*r iw*	IMUL *r16, r/m16, imm16*	RMI	Valid	Valid	word register ← *r/m16* ∗ immediate word.
69 /*r id*	IMUL *r32, r/m32, imm32*	RMI	Valid	Valid	doubleword register ← *r/m32* ∗ immediate doubleword.
REX.W + 69 /*r id*	IMUL *r64, r/m64, imm32*	RMI	Valid	N.E.	Quadword register ← *r/m64* ∗ immediate doubleword.

NOTES:

* In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r, w)	NA	NA	NA
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	imm8/16/32	NA

Description

Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.

- **One-operand form** — This form is identical to that used by the MUL instruction. Here, the source operand (in a general-purpose register or memory location) is multiplied by the value in the AL, AX, EAX, or RAX register (depending on the operand size) and the product (twice the size of the input operand) is stored in the AX, DX:AX, EDX:EAX, or RDX:RAX registers, respectively.
- **Two-operand form** — With this form the destination operand (the first operand) is multiplied by the source operand (second operand). The destination operand is a general-purpose register and the source operand is an immediate value, a general-purpose register, or a memory location. The intermediate product (twice the size of the input operand) is truncated and stored in the destination operand location.

- **Three-operand form** — This form requires a destination operand (the first operand) and two source operands (the second and the third operands). Here, the first source operand (which can be a general-purpose register or a memory location) is multiplied by the second source operand (an immediate value). The intermediate product (twice the size of the first source operand) is truncated and stored in the destination operand (a general-purpose register).

When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The CF and OF flags are set when the signed integer value of the intermediate product differs from the sign extended operand-size-truncated product, otherwise the CF and OF flags are cleared.

The three forms of the IMUL instruction are similar in that the length of the product is calculated to twice the length of the operands. With the one-operand form, the product is stored exactly in the destination. With the two- and three- operand forms, however, the result is truncated to the length of the destination before it is stored in the destination register. Because of this truncation, the CF or OF flag should be tested to ensure that no significant bits are lost.

The two- and three-operand forms may also be used with unsigned operands because the lower half of the product is the same regardless if the operands are signed or unsigned. The CF and OF flags, however, cannot be used to determine if the upper half of the result is non-zero.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. Use of REX.W modifies the three forms of the instruction as follows.

- **One-operand form** —The source operand (in a 64-bit general-purpose register or memory location) is multiplied by the value in the RAX register and the product is stored in the RDX:RAX registers.
- **Two-operand form** — The source operand is promoted to 64 bits if it is a register or a memory location. The destination operand is promoted to 64 bits.
- **Three-operand form** — The first source operand (either a register or a memory location) and destination operand are promoted to 64 bits. If the source operand is an immediate, it is sign extended to 64 bits.

Operation

```
IF (NumberOfOperands = 1)
   THEN IF (OperandSize = 8)
       THEN
           TMP_XP ← AL ∗ SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *);
           AX ← TMP_XP[15:0];
           SF ← TMP_XP[7];
           IF SignExtend(TMP_XP[7:0]) = TMP_XP
               THEN CF ← 0; OF ← 0;
               ELSE CF ← 1; OF ← 1; FI;
       ELSE IF OperandSize = 16
           THEN
               TMP_XP ← AX ∗ SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *)
               DX:AX ← TMP_XP[31:0];
               SF ← TMP_XP[15];
               IF SignExtend(TMP_XP[15:0]) = TMP_XP
                   THEN CF ← 0; OF ← 0;
                   ELSE CF ← 1; OF ← 1; FI;
           ELSE IF OperandSize = 32
               THEN
                   TMP_XP ← EAX ∗ SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC*)
```

```
                    EDX:EAX ← TMP_XP[63:0];
                    SF ← TMP_XP[31];
                    IF SignExtend(TMP_XP[31:0]) = TMP_XP
                        THEN CF ← 0; OF ← 0;
                        ELSE CF ← 1; OF ← 1; FI;
                ELSE (* OperandSize = 64 *)
                    TMP_XP ← RAX ∗ SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *)
                    EDX:EAX ← TMP_XP[127:0];
                    SF ← TMP_XP[63];
                    IF SignExtend(TMP_XP[63:0]) = TMP_XP
                        THEN CF ← 0; OF ← 0;
                        ELSE CF ← 1; OF ← 1; FI;
                FI;
        FI;
    ELSE IF (NumberOfOperands = 2)
        THEN
            TMP_XP ← DEST ∗ SRC (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC *)
            DEST ← TruncateToOperandSize(TMP_XP);
            SF ← MSB(DEST);
            IF SignExtend(DEST) ≠ TMP_XP
                THEN CF ← 1; OF ← 1;
                ELSE CF ← 0; OF ← 0; FI;
        ELSE (* NumberOfOperands = 3 *)
            TMP_XP ← SRC1 ∗ SRC2 (* Signed multiplication; TMP_XP is a signed integer at twice the width of the SRC1 *)
            DEST ← TruncateToOperandSize(TMP_XP);
            SF ← MSB(DEST);
            IF SignExtend(DEST) ≠ TMP_XP
                THEN CF ← 1; OF ← 1;
                ELSE CF ← 0; OF ← 0; FI;
    FI;
FI;
```

Flags Affected

SF is updated according to the most significant bit of the operand-size-truncated result in the destination. For the one operand form of the instruction, the CF and OF flags are set when significant bits are carried into the upper half of the result and cleared when the result fits exactly in the lower half of the result. For the two- and three-operand forms of the instruction, the CF and OF flags are set when the result must be truncated to fit in the destination operand size and cleared when the result fits exactly in the destination operand size. The ZF, AF, and PF flags are undefined.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a NULL NULL segment selector.

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

...

LMSW—Load Machine Status Word

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 01 /6	LMSW *r/m*16	M	Valid	Valid	Loads *r/m*16 in machine status word of CR0.

Instruction Operand Encoding

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

Description

Loads the source operand into the machine status word, bits 0 through 15 of register CR0. The source operand can be a 16-bit general-purpose register or a memory location. Only the low-order 4 bits of the source operand (which contains the PE, MP, EM, and TS flags) are loaded into CR0. The PG, CD, NW, AM, WP, NE, and ET flags of CR0 are not affected. The operand-size attribute has no effect on this instruction.

If the PE flag of the source operand (bit 0) is set to 1, the instruction causes the processor to switch to protected mode. While in protected mode, the LMSW instruction cannot be used to clear the PE flag and force a switch back to real-address mode.

The LMSW instruction is provided for use in operating-system software; it should not be used in application programs. In protected or virtual-8086 mode, it can only be executed at CPL 0.

This instruction is provided for compatibility with the Intel 286 processor; programs and procedures intended to run on IA-32 and Intel 64 processors beginning with Intel386 processors should use the MOV (control registers) instruction to load the whole CR0 register. The MOV CR0 instruction can be used to set and clear the PE flag in CR0, allowing a procedure or program to switch between protected and real-address modes.

This instruction is a serializing instruction.

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

See “Changes to Instruction Behavior in VMX Non-Root Operation” in Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
CR0[0:3] ← SRC[0:3];
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 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.
#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.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	The LMSW instruction is not recognized in real-address 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 current privilege level is not 0.
	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.

...

LOCK—Assert LOCK# Signal Prefix

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F0	LOCK	NP	Valid	Valid	Asserts LOCK# signal for duration of the accompanying instruction.

NOTES:

* See IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Causes the processor’s LOCK# signal to be asserted during execution of the accompanying instruction (turns the instruction into an atomic instruction). In a multiprocessor environment, the LOCK# signal ensures that the processor has exclusive use of any shared memory while the signal is asserted.

In most IA-32 and all Intel 64 processors, locking may occur without the LOCK# signal being asserted. See the “IA-32 Architecture Compatibility” section below for more details.

The LOCK prefix can be prepended only to the following instructions and only to those forms of the instructions where the destination operand is a memory operand: ADD, ADC, AND, BTC, BTR, BTS, CMPXCHG, CMPXCH8B, CMPXCHG16B, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR, XADD, and XCHG. If the LOCK prefix is used with one of these instructions and the source operand is a memory operand, an undefined opcode exception (#UD) may be generated. An undefined opcode exception will also be generated if the LOCK prefix is used with any instruction not in the above list. The XCHG instruction always asserts the LOCK# signal regardless of the presence or absence of the LOCK prefix.

The LOCK prefix is typically used with the BTS instruction to perform a read-modify-write operation on a memory location in shared memory environment.

The integrity of the LOCK prefix is not affected by the alignment of the memory field. Memory locking is observed for arbitrarily misaligned fields.

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

IA-32 Architecture Compatibility

Beginning with the P6 family processors, when the LOCK prefix is prefixed to an instruction and the memory area being accessed is cached internally in the processor, the LOCK# signal is generally not asserted. Instead, only the processor’s cache is locked. Here, the processor’s cache coherency mechanism ensures that the operation is carried out atomically with regards to memory. See “Effects of a Locked Operation on Internal Processor Caches” in Chapter 8 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual*, *Volume 3A*, the for more information on locking of caches.

Operation

```
AssertLOCK#(DurationOfAccompaningInstruction);
```

Flags Affected

None.

Protected Mode Exceptions

#UD If the LOCK prefix is used with an instruction not listed: ADD, ADC, AND, BTC, BTR, BTS, CMPXCHG, CMPXCH8B, CMPXCHG16B, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR, XADD, XCHG.

Other exceptions can be generated by the instruction when the LOCK prefix is applied.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

MOV—Move

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
88 /_r_	MOV _r/m8,r8_	MR	Valid	Valid	Move _r8_ to _r/m8._
REX + 88 /_r_	MOV _r/m8***,r8***_	MR	Valid	N.E.	Move _r8_ to _r/m8._
89 /_r_	MOV _r/m16,r16_	MR	Valid	Valid	Move _r16_ to _r/m16._
89 /_r_	MOV _r/m32,r32_	MR	Valid	Valid	Move _r32_ to _r/m32._
REX.W + 89 /_r_	MOV _r/m64,r64_	MR	Valid	N.E.	Move _r64_ to _r/m64._
8A /_r_	MOV _r8,r/m8_	RM	Valid	Valid	Move _r/m8_ to _r8._
REX + 8A /_r_	MOV _r8***,r/m8***_	RM	Valid	N.E.	Move _r/m8_ to _r8._
8B /_r_	MOV _r16,r/m16_	RM	Valid	Valid	Move _r/m16_ to _r16._
8B /_r_	MOV _r32,r/m32_	RM	Valid	Valid	Move _r/m32_ to _r32._
REX.W + 8B /_r_	MOV _r64,r/m64_	RM	Valid	N.E.	Move _r/m64_ to _r64._
8C /_r_	MOV _r/m16,Sreg**_	MR	Valid	Valid	Move segment register to _r/m16._
REX.W + 8C /_r_	MOV _r/m64,Sreg**_	MR	Valid	Valid	Move zero extended 16-bit segment register to _r/m64._
8E /_r_	MOV _Sreg,r/m16**_	RM	Valid	Valid	Move _r/m16_ to segment register.
REX.W + 8E /_r_	MOV _Sreg,r/m64**_	RM	Valid	Valid	Move _lower 16 bits of r/m64_ to segment register.
A0	MOV AL,_moffs8*_	FD	Valid	Valid	Move byte at (_seg:offset_) to AL.
REX.W + A0	MOV AL,_moffs8*_	FD	Valid	N.E.	Move byte at (_offset_) to AL.
A1	MOV AX,_moffs16*_	FD	Valid	Valid	Move word at (_seg:offset_) to AX.
A1	MOV EAX,_moffs32*_	FD	Valid	Valid	Move doubleword at (_seg:offset_) to EAX.
REX.W + A1	MOV RAX,_moffs64*_	FD	Valid	N.E.	Move quadword at (_offset_) to RAX.
A2	MOV _moffs8_,AL	TD	Valid	Valid	Move AL to (_seg:offset_).
REX.W + A2	MOV _moffs8***_,AL	TD	Valid	N.E.	Move AL to (_offset_).
A3	MOV _moffs16*_,AX	TD	Valid	Valid	Move AX to (_seg:offset_).
A3	MOV _moffs32*_,EAX	TD	Valid	Valid	Move EAX to (_seg:offset_).
REX.W + A3	MOV _moffs64*_,RAX	TD	Valid	N.E.	Move RAX to (_offset_).
B0+ _rb ib_	MOV _r8, imm8_	OI	Valid	Valid	Move _imm8_ to _r8._
REX + B0+ _rb ib_	MOV _r8***, imm8_	OI	Valid	N.E.	Move _imm8_ to _r8._
B8+ _rw iw_	MOV _r16, imm16_	OI	Valid	Valid	Move _imm16_ to _r16._
B8+ _rd id_	MOV _r32, imm32_	OI	Valid	Valid	Move _imm32_ to _r32._
REX.W + B8+ _rd io_	MOV _r64, imm64_	OI	Valid	N.E.	Move _imm64_ to _r64._
C6 _/0 ib_	MOV _r/m8, imm8_	MI	Valid	Valid	Move _imm8_ to _r/m8._
REX + C6 _/0 ib_	MOV _r/m8***, imm8_	MI	Valid	N.E.	Move _imm8_ to _r/m8._
C7 _/0 iw_	MOV _r/m16, imm16_	MI	Valid	Valid	Move _imm16_ to _r/m16._
C7 _/0 id_	MOV _r/m32, imm32_	MI	Valid	Valid	Move _imm32_ to _r/m32._
REX.W + C7 _/0 io_	MOV _r/m64, imm32_	MI	Valid	N.E.	Move _imm32 sign extended to 64-bits_ to _r/m64._

NOTES:

* The *moffs8*, *moffs16*, *moffs32* and *moffs64* operands specify a simple offset relative to the segment base, where 8, 16, 32 and 64 refer to the size of the data. The address-size attribute of the instruction determines the size of the offset, either 16, 32 or 64 bits.

** In 32-bit mode, the assembler may insert the 16-bit operand-size prefix with this instruction (see the following "Description" section for further information).

***In 64-bit mode, *r/m8* can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA
FD	AL/AX/EAX/RAX	Moffs	NA	NA
TD	Moffs (w)	AL/AX/EAX/RAX	NA	NA
OI	opcode + rd (w)	imm8/16/32/64	NA	NA
MI	ModRM:r/m (w)	imm8/16/32/64	NA	NA

Description

Copies the second operand (source operand) to the first operand (destination operand). The source operand can be an immediate value, general-purpose register, segment register, or memory location; the destination register can be a general-purpose register, segment register, or memory location. Both operands must be the same size, which can be a byte, a word, a doubleword, or a quadword.

The MOV instruction cannot be used to load the CS register. Attempting to do so results in an invalid opcode exception (#UD). To load the CS register, use the far JMP, CALL, or RET instruction.

If the destination operand is a segment register (DS, ES, FS, GS, or SS), the source operand must be a valid segment selector. In protected mode, moving a segment selector into a segment register automatically causes the segment descriptor information associated with that segment selector to be loaded into the hidden (shadow) part of the segment register. While loading this information, the segment selector and segment descriptor information is validated (see the "Operation" algorithm below). The segment descriptor data is obtained from the GDT or LDT entry for the specified segment selector.

A NULL segment selector (values 0000-0003) can be loaded into the DS, ES, FS, and GS registers without causing a protection exception. However, any subsequent attempt to reference a segment whose corresponding segment register is loaded with a NULL value causes a general protection exception (#GP) and no memory reference occurs.

Loading the SS register with a MOV instruction inhibits all interrupts until after the execution of the next instruction. This operation allows a stack pointer to be loaded into the ESP register with the next instruction (MOV ESP, **stack-pointer value**) before an interrupt occurs[1]. Be aware that the LSS instruction offers a more efficient method of loading the SS and ESP registers.

When executing MOV Reg, Sreg, the processor copies the content of Sreg to the 16 least significant bits of the general-purpose register. The upper bits of the destination register are zero for most IA-32 processors (Pentium Pro processors and later) and all Intel 64 processors, with the exception that bits 31:16 are undefined for Intel Quark X1000 processors, Pentium and earlier processors.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
DEST ← SRC;
```

Loading a segment register while in protected mode results in special checks and actions, as described in the following listing. These checks are performed on the segment selector and the segment descriptor to which it points.

```
IF SS is loaded
    THEN
        IF segment selector is NULL
            THEN #GP(0); FI;
        IF segment selector index is outside descriptor table limits
        or segment selector's RPL ≠ CPL
        or segment is not a writable data segment
        or DPL ≠ CPL
            THEN #GP(selector); FI;
        IF segment not marked present
            THEN #SS(selector);
            ELSE
                SS ← segment selector;
                SS ← segment descriptor; FI;
FI;

IF DS, ES, FS, or GS is loaded with non-NULL selector
THEN
    IF segment selector index is outside descriptor table limits
    or segment is not a data or readable code segment
    or ((segment is a data or nonconforming code segment)
    or ((RPL > DPL) and (CPL > DPL))
        THEN #GP(selector); FI;
    IF segment not marked present
        THEN #NP(selector);
        ELSE
            SegmentRegister ← segment selector;
            SegmentRegister ← segment descriptor; FI;
FI;

IF DS, ES, FS, or GS is loaded with NULL selector
    THEN
        SegmentRegister ← segment selector;
        SegmentRegister ← segment descriptor;
FI;
```

[1] If a code instruction breakpoint (for debug) is placed on an instruction located immediately after a MOV SS instruction, the breakpoint may not be triggered. However, in a sequence of instructions that load the SS register, only the first instruction in the sequence is guaranteed to delay an interrupt.

In the following sequence, interrupts may be recognized before MOV ESP, EBP executes:

```
MOV SS, EDX
MOV SS, EAX
MOV ESP, EBP
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0) If attempt is made to load SS register with NULL segment selector.

If the destination operand is in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a NULL segment selector.

#GP(selector) If segment selector index is outside descriptor table limits.

If the SS register is being loaded and the segment selector's RPL and the segment descriptor’s DPL are not equal to the CPL.

If the SS register is being loaded and the segment pointed to is a non-writable data segment.

If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.

If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the 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 pointed to is marked not present.

#NP If the DS, ES, FS, or GS register is being loaded and the segment pointed to 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 attempt is made to load the CS register.

If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

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

#UD If attempt is made to load the CS register.

If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If attempt is made to load the CS register.
	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If an attempt is made to load SS register with NULL segment selector when CPL = 3.
	If an attempt is made to load SS register with NULL segment selector when CPL < 3 and CPL ≠ RPL.
#GP(selector)	If segment selector index is outside descriptor table limits.
	If the memory access to the descriptor table is non-canonical.
	If the SS register is being loaded and the segment selector's RPL and the segment descriptor’s DPL are not equal to the CPL.
	If the SS register is being loaded and the segment pointed to is a nonwritable data segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
#SS(0)	If the stack address is in a non-canonical form.
#SS(selector)	If the SS register is being loaded and the segment pointed to 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 attempt is made to load the CS register.
	If the LOCK prefix is used.

...

MOVNTDQA — Load Double Quadword Non-Temporal Aligned Hint

Opcode/ Instruction	**Op/ En**	**64/32-bit Mode**	**CPUID Feature Flag**	**Description**
66 0F 38 2A /r MOVNTDQA *xmm1, m128*	RM	V/V	SSE4_1	Move double quadword from *m128* to *xmm* using non-temporal hint if WC memory type.
VEX.128.66.0F38.WIG 2A /r VMOVNTDQA *xmm1, m128*	RM	V/V	AVX	Move double quadword from *m128* to *xmm* using non-temporal hint if WC memory type.
VEX.256.66.0F38.WIG 2A /r VMOVNTDQA *ymm1, m256*	RM	V/V	AVX2	Move 256-bit data from *m256* to ymm using non-temporal hint if WC memory type.

Instruction Operand Encoding

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

Description

(V)MOVNTDQA loads a double quadword from the source operand (second operand) to the destination operand (first operand) using a non-temporal hint. A processor implementation may make use of the non-temporal hint associated with this instruction if the memory source is WC (write combining) memory type. An implementation may also make use of the non-temporal hint associated with this instruction if the memory source is WB (write back) memory type.

A processor's implementation of the non-temporal hint does not override the effective memory type semantics, but the implementation of the hint is processor dependent. For example, a processor implementation may choose to ignore the hint and process the instruction as a normal MOVDQA for any memory type. Another implementation of the hint for WC memory type may optimize data transfer throughput of WC reads. A third implementation may optimize cache reads generated by (V)MOVNTDQA on WB memory type to reduce cache evictions.

WC Streaming Load Hint

For WC memory type in particular, the processor never appears to read the data into the cache hierarchy. Instead, the non-temporal hint may be implemented by loading a temporary internal buffer with the equivalent of an aligned cache line without filling this data to the cache. Any memory-type aliased lines in the cache will be snooped and flushed. Subsequent MOVNTDQA reads to unread portions of the WC cache line will receive data from the temporary internal buffer if data is available. The temporary internal buffer may be flushed by the processor at any time for any reason, for example:

- A load operation other than a (V)MOVNTDQA which references memory already resident in a temporary internal buffer.
- A non-WC reference to memory already resident in a temporary internal buffer.
- Interleaving of reads and writes to memory currently residing in a single temporary internal buffer.
- Repeated (V)MOVNTDQA loads of a particular 16-byte item in a streaming line.
- Certain micro-architectural conditions including resource shortages, detection of a mis-speculation condition, and various fault conditions

The memory type of the region being read can override the non-temporal hint, if the memory address specified for the non-temporal read is not a WC memory region. Information on non-temporal reads and writes can be found in Chapter 11, "Memory Cache Control" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Because the WC protocol uses a weakly-ordered memory consistency model, an MFENCE should be used in conjunction with MOVNTDQA instructions if multiple processors might reference the same WC memory locations or in order to synchronize reads of a processor with writes by other agents in the system. Because of the speculative nature of fetching due to MOVNTDQA, Streaming loads must not be used to reference memory addresses that are mapped to I/O devices having side effects or when reads to these devices are destructive. For additional information on MOVNTDQA usages, see Section 12.10.3 in Chapter 12, "Programming with SSE3, SSSE3 and SSE4" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

The 128-bit (V)MOVNTDQA addresses must be 16-byte aligned or the instruction will cause a #GP.

The 256-bit VMOVNTDQA addresses must be 32-byte aligned or the instruction will cause a #GP.

Note: In VEX-128 encoded versions, VEX.vvvv is reserved and must be 1111b, VEX.L must be 0; otherwise instructions will #UD.

Operation

MOVNTDQA (128bit- Legacy SSE form)

```
DEST ← SRC
DEST[VLMAX-1:128] (Unmodified)
```

VMOVNTDQA (VEX.128 encoded form)

```
DEST ← SRC
DEST[VLMAX-1:128] ← 0
```

VMOVNTDQA (VEX.256 encoded form)

```
DEST[255:0] ← SRC[255:0]
```

Intel C/C++ Compiler Intrinsic Equivalent

```
(V)MOVNTDQA:     __m128i _mm_stream_load_si128 (__m128i *p);
VMOVNTDQA:       __m256i _mm256_stream_load_si256 (const __m256i *p);
```

Flags Affected

None

Other Exceptions

See Exceptions Type 1.SSE4.1; additionally

#UD If VEX.L= 1.

If VEX.vvvv ≠ 1111B.

...

7. Updates to Chapter 4, Volume 2B

Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual*, *Volume 2B:* Instruction Set Reference, N-Z.

...

4.2 COMMON TRANSFORMATION AND PRIMITIVE FUNCTIONS FOR SHA1XXX AND SHA256XXX

The following primitive functions and transformations are used in the algorithmic descriptions of SHA1 and SHA256 instruction extensions SHA1NEXTE, SHA1RNDS4, SHA1MSG1, SHA1MSG2, SHA256RNDS4, SHA256MSG1 and SHA256MSG2. The operands of these primitives and transformation are generally 32-bit DWORD integers.

- f0(): A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 1 to 20 processing.

  f0(B,C,D) ← (B AND C) XOR ((NOT(B) AND D)

- f1(): A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 21 to 40 processing.

  f1(B,C,D) ← B XOR C XOR D

- f2(): A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 41 to 60 processing.

  f2(B,C,D) ← (B AND C) XOR (B AND D) XOR (C AND D)

- f3(): A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 61 to 80 processing. It is the same as f1().

  f3(B,C,D) ← B XOR C XOR D

- Ch(): A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).

  Ch(E,F,G) ← (E AND F) XOR ((NOT E) AND G)

- Maj(): A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).

  Maj(A,B,C) ← (A AND B) XOR (A AND C) XOR (B AND C)

ROR is rotate right operation

(A ROR N) ← A[N-1:0] || A[Width-1:N]

ROL is rotate left operation

(A ROL N) ← A ROR (Width-N)

SHR is the right shift operation

(A SHR N) ← ZEROES[N-1:0] || A[Width-1:N]

- Σ_0( ): A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.

  Σ_0(A) ← (A ROR 2) XOR (A ROR 13) XOR (A ROR 22)

- Σ_1( ): A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.

  Σ_1(E) ← (E ROR 6) XOR (E ROR 11) XOR (E ROR 25)

- σ_0( ): A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.

  σ_0(W) ← (W ROR 7) XOR (W ROR 18) XOR (W SHR 3)

- σ_1( ): A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.

  σ_1(W) ← (W ROR 17) XOR (W ROR 19) XOR (W SHR 10)

- K_i: SHA1 Constants dependent on immediate i.

  K0 = 0x5A827999

  K1 = 0x6ED9EBA1

  K2 = 0X8F1BBCDC

  K3 = 0xCA62C1D6

4.3 INSTRUCTIONS (N-Z)

Chapter 4 continues an alphabetical discussion of Intel® 64 and IA-32 instructions (N-Z). See also: Chapter 3, “Instruction Set Reference, A-M,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*.

...

POP—Pop a Value from the Stack

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
8F /0	POP r/*m16*	M	Valid	Valid	Pop top of stack into *m16*; increment stack pointer.
8F /0	POP r/*m32*	M	N.E.	Valid	Pop top of stack into *m32*; increment stack pointer.
8F /0	POP r/*m64*	M	Valid	N.E.	Pop top of stack into *m64*; increment stack pointer. Cannot encode 32-bit operand size.
58+ *rw*	POP *r16*	O	Valid	Valid	Pop top of stack into *r16*; increment stack pointer.
58+ *rd*	POP *r32*	O	N.E.	Valid	Pop top of stack into *r32*; increment stack pointer.
58+ *rd*	POP *r64*	O	Valid	N.E.	Pop top of stack into *r64*; increment stack pointer. Cannot encode 32-bit operand size.
1F	POP DS	NP	Invalid	Valid	Pop top of stack into DS; increment stack pointer.
07	POP ES	NP	Invalid	Valid	Pop top of stack into ES; increment stack pointer.
17	POP SS	NP	Invalid	Valid	Pop top of stack into SS; increment stack pointer.
0F A1	POP FS	NP	Valid	Valid	Pop top of stack into FS; increment stack pointer by 16 bits.
0F A1	POP FS	NP	N.E.	Valid	Pop top of stack into FS; increment stack pointer by 32 bits.
0F A1	POP FS	NP	Valid	N.E.	Pop top of stack into FS; increment stack pointer by 64 bits.
0F A9	POP GS	NP	Valid	Valid	Pop top of stack into GS; increment stack pointer by 16 bits.
0F A9	POP GS	NP	N.E.	Valid	Pop top of stack into GS; increment stack pointer by 32 bits.
0F A9	POP GS	NP	Valid	N.E.	Pop top of stack into GS; increment stack pointer by 64 bits.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (w)	NA	NA	NA
O	opcode + rd (w)	NA	NA	NA
NP	NA	NA	NA	NA

Description

Loads the value from the top of the stack to the location specified with the destination operand (or explicit opcode) and then increments the stack pointer. The destination operand can be a general-purpose register, memory location, or segment register.

Address and operand sizes are determined and used as follows:

- Address size. The D flag in the current code-segment descriptor determines the default address size; it may be overridden by an instruction prefix (67H).

  The address size is used only when writing to a destination operand in memory.

- Operand size. The D flag in the current code-segment descriptor determines the default operand size; it may be overridden by instruction prefixes (66H or REX.W).

  The operand size (16, 32, or 64 bits) determines the amount by which the stack pointer is incremented (2, 4 or 8).

- Stack-address size. Outside of 64-bit mode, the B flag in the current stack-segment descriptor determines the size of the stack pointer (16 or 32 bits); in 64-bit mode, the size of the stack pointer is always 64 bits.

  The stack-address size determines the width of the stack pointer when reading from the stack in memory and when incrementing the stack pointer. (As stated above, the amount by which the stack pointer is incremented is determined by the operand size.)

If the destination operand is one of the segment registers DS, ES, FS, GS, or SS, the value loaded into the register must be a valid segment selector. In protected mode, popping a segment selector into a segment register automatically causes the descriptor information associated with that segment selector to be loaded into the hidden (shadow) part of the segment register and causes the selector and the descriptor information to be validated (see the "Operation" section below).

A NULL value (0000-0003) may be popped into the DS, ES, FS, or GS register without causing a general protection fault. However, any subsequent attempt to reference a segment whose corresponding segment register is loaded with a NULL value causes a general protection exception (#GP). In this situation, no memory reference occurs and the saved value of the segment register is NULL.

The POP instruction cannot pop a value into the CS register. To load the CS register from the stack, use the RET instruction.

If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register. For the case of a 16-bit stack where ESP wraps to 0H as a result of the POP instruction, the resulting location of the memory write is processor-family-specific.

The POP ESP instruction increments the stack pointer (ESP) before data at the old top of stack is written into the destination.

A POP SS instruction inhibits all interrupts, including the NMI interrupt, until after execution of the next instruction. This action allows sequential execution of POP SS and MOV ESP, EBP instructions without the danger of having an invalid stack during an interrupt[1]. However, use of the LSS instruction is the preferred method of loading the SS and ESP registers.

In 64-bit mode, using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). When in 64-bit mode, POPs using 32-bit operands are not encodable and POPs to DS, ES, SS are not valid. See the summary chart at the beginning of this section for encoding data and limits.

Operation

```
IF StackAddrSize = 32
```

[1] If a code instruction breakpoint (for debug) is placed on an instruction located immediately after a POP SS instruction, the breakpoint may not be triggered. However, in a sequence of instructions that POP the SS register, only the first instruction in the sequence is guaranteed to delay an interrupt.

In the following sequence, interrupts may be recognized before POP ESP executes:

```
POP SS
POP SS
POP ESP
```

```
    THEN
        IF OperandSize = 32
            THEN
                DEST ← SS:ESP; (* Copy a doubleword *)
                ESP ← ESP + 4;
            ELSE (* OperandSize = 16*)
                DEST ← SS:ESP; (* Copy a word *)
                ESP ← ESP + 2;
        FI;
    ELSE IF StackAddrSize = 64
        THEN
            IF OperandSize = 64
                THEN
                    DEST ← SS:RSP; (* Copy quadword *)
                    RSP ← RSP + 8;
                ELSE (* OperandSize = 16*)
                    DEST ← SS:RSP; (* Copy a word *)
                    RSP ← RSP + 2;
            FI;
        FI;
    ELSE StackAddrSize = 16
        THEN
            IF OperandSize = 16
                THEN
                    DEST ← SS:SP; (* Copy a word *)
                    SP ← SP + 2;
                ELSE (* OperandSize = 32 *)
                    DEST ← SS:SP; (* Copy a doubleword *)
                    SP ← SP + 4;
            FI;

FI;
```

Loading a segment register while in protected mode results in special actions, as described in the following listing. These checks are performed on the segment selector and the segment descriptor it points to.

```
64-BIT_MODE
IF FS, or GS is loaded with non-NULL selector;
    THEN
        IF segment selector index is outside descriptor table limits
            OR segment is not a data or readable code segment
            OR ((segment is a data or nonconforming code segment)
                AND (both RPL and CPL > DPL))
                    THEN #GP(selector);
            IF segment not marked present
                THEN #NP(selector);
        ELSE
            SegmentRegister ← segment selector;
            SegmentRegister ← segment descriptor;
        FI;
```

```
FI;
IF FS, or GS is loaded with a NULL selector;
        THEN
            SegmentRegister ← segment selector;
            SegmentRegister ← segment descriptor;
FI;


PREOTECTED MODE OR COMPATIBILITY MODE;

IF SS is loaded;
    THEN
        IF segment selector is NULL
            THEN #GP(0);
        FI;
        IF segment selector index is outside descriptor table limits
            or segment selector's RPL ≠ CPL
            or segment is not a writable data segment
            or DPL ≠ CPL
                THEN #GP(selector);
        FI;
        IF segment not marked present
            THEN #SS(selector);
            ELSE
                SS ← segment selector;
                SS ← segment descriptor;
        FI;
FI;

IF DS, ES, FS, or GS is loaded with non-NULL selector;
    THEN
        IF segment selector index is outside descriptor table limits
            or segment is not a data or readable code segment
            or ((segment is a data or nonconforming code segment)
            and (both RPL and CPL > DPL))
                THEN #GP(selector);
        FI;
        IF segment not marked present
            THEN #NP(selector);
            ELSE
                SegmentRegister ← segment selector;
                SegmentRegister ← segment descriptor;
         FI;
FI;

IF DS, ES, FS, or GS is loaded with a NULL selector
    THEN
        SegmentRegister ← segment selector;
        SegmentRegister ← segment descriptor;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0) If attempt is made to load SS register with NULL segment selector.

If the destination operand is in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.

#GP(selector) If segment selector index is outside descriptor table limits.

If the SS register is being loaded and the segment selector's RPL and the segment descriptor’s DPL are not equal to the CPL.

If the SS register is being loaded and the segment pointed to is a non-writable data segment.

If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.

If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.

#SS(0) If the current top of stack is not within the stack segment.

If a memory operand effective address is outside the SS segment limit.

#SS(selector) If the SS register is being loaded and the segment pointed to is marked not present.

#NP If the DS, ES, FS, or GS register is being loaded and the segment pointed to is marked not present.

#PF(fault-code) If a page fault occurs.

#AC(0) If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.

#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.

#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.

#PF(fault-code) If a page fault occurs.

#AC(0) If an unaligned memory reference is made while alignment checking is enabled.

#UD If the LOCK prefix is used.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

#SS(0) If the stack address is in a non-canonical form.

#GP(selector) If the descriptor is outside the descriptor table limit.

If the FS or GS register is being loaded and the segment pointed to is not a data or readable code segment.

If the FS or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.

#AC(0) If an unaligned memory reference is made while alignment checking is enabled.

#PF(fault-code) If a page fault occurs.

#NP If the FS or GS register is being loaded and the segment pointed to is marked not present.

#UD If the LOCK prefix is used.

...

POPF/POPFD/POPFQ—Pop Stack into EFLAGS Register

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
9D	POPF	NP	Valid	Valid	Pop top of stack into lower 16 bits of EFLAGS.
9D	POPFD	NP	N.E.	Valid	Pop top of stack into EFLAGS.
9D	POPFQ	NP	Valid	N.E.	Pop top of stack and zero-extend into RFLAGS.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Pops a doubleword (POPFD) from the top of the stack (if the current operand-size attribute is 32) and stores the value in the EFLAGS register, or pops a word from the top of the stack (if the operand-size attribute is 16) and stores it in the lower 16 bits of the EFLAGS register (that is, the FLAGS register). These instructions reverse the operation of the PUSHF/PUSHFD instructions.

The POPF (pop flags) and POPFD (pop flags double) mnemonics reference the same opcode. The POPF instruction is intended for use when the operand-size attribute is 16; the POPFD instruction is intended for use when the operand-size attribute is 32. Some assemblers may force the operand size to 16 for POPF and to 32 for POPFD. Others may treat the mnemonics as synonyms (POPF/POPFD) and use the setting of the operand-size attribute to determine the size of values to pop from the stack.

The effect of POPF/POPFD on the EFLAGS register changes, depending on the mode of operation. See the Table 4-12 and key below for details.

When operating in protected, compatibility, or 64-bit mode at privilege level 0 (or in real-address mode, the equivalent to privilege level 0), all non-reserved flags in the EFLAGS register except RF[1], VIP, VIF, and VM may be modified. VIP, VIF and VM remain unaffected.

When operating in protected, compatibility, or 64-bit mode with a privilege level greater than 0, but less than or equal to IOPL, all flags can be modified except the IOPL field and RF[1], IF, VIP, VIF, and VM; these remain unaffected. The AC and ID flags can only be modified if the operand-size attribute is 32. The interrupt flag (IF) is altered only when executing at a level at least as privileged as the IOPL. If a POPF/POPFD instruction is executed with insufficient privilege, an exception does not occur but privileged bits do not change.

When operating in virtual-8086 mode (EFLAGS.VM = 1) without the virtual-8086 mode extensions (CR4.VME = 0), the POPF/POPFD instructions can be used only if IOPL = 3; otherwise, a general-protection exception (#GP)

[1] RF is always zero after the execution of POPF. This is because POPF, like all instructions, clears RF as it begins to execute.

occurs. If the virtual-8086 mode extensions are enabled (CR4.VME = 1), POPF (but not POPFD) can be executed in virtual-8086 mode with IOPL < 3.

In 64-bit mode, the mnemonic assigned is POPFQ (note that the 32-bit operand is not encodable). POPFQ pops 64 bits from the stack. Reserved bits of RFLAGS (including the upper 32 bits of RFLAGS) are not affected.

See Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for more information about the EFLAGS registers.

Table 4-12 Effect of POPF/POPFD on the EFLAGS Register

Mode	Operand Size	CPL	IOPL	Flags																	Notes
				21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0	
				ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF	
Real-Address Mode (CR0.PE = 0)	16	0	0-3	N	N	N	N	N	0	S	S	S	S	S	S	S	S	S	S	S	
	32	0	0-3	S	N	N	S	N	0	S	S	S	S	S	S	S	S	S	S	S	
Protected, Compatibility, and 64-Bit Modes (CR0.PE = 1, EFLAGS.VM = 0)	16	0	0-3	N	N	N	N	N	0	S	S	S	S	S	S	S	S	S	S	S	
	16	1-3	<CPL	N	N	N	N	N	0	S	N	S	S	N	S	S	S	S	S	S	
	16	1-3	≥CPL	N	N	N	N	N	0	S	N	S	S	S	S	S	S	S	S	S	
	32, 64	0	0-3	S	N	N	S	N	0	S	S	S	S	S	S	S	S	S	S	S	
	32, 64	1-3	<CPL	S	N	N	S	N	0	S	N	S	S	N	S	S	S	S	S	S	
	32, 64	1-3	≥CPL	S	N	N	S	N	0	S	N	S	S	S	S	S	S	S	S	S	
Virtual-8086 (CR0.PE = 1, EFLAGS.VM = 1, CR4.VME = 0)	16	3	0-2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
	16	3	3	N	N	N	N	N	0	S	N	S	S	S	S	S	S	S	S	S	
	32	3	0-2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
	32	3	3	S	N	N	S	N	0	S	N	S	S	S	S	S	S	S	S	S	
VME (CR0.PE = 1, EFLAGS.VM = 1, CR4.VME = 1)	16	3	0-2	N/X	N/X	SV/X	N/X	N/X	0/X	S/X	N/X	S/X	S/X	N/X	S/X	S/X	S/X	S/X	S/X	S/X	2
	16	3	3	N	N	N	N	N	0	S	N	S	S	S	S	S	S	S	S	S	
	32	3	0-2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
	32	3	3	S	N	N	S	N	0	S	N	S	S	S	S	S	S	S	S	S	

NOTES:

1. #GP fault - no flag update
2. #GP fault with no flag update if VIP=1 in EFLAGS register and IF=1 in FLAGS value on stack

Key	
S	Updated from stack
SV	Updated from IF (bit 9) in FLAGS value on stack
N	No change in value
X	No EFLAGS update
0	Value is cleared

Operation

```
IF VM = 0 (* Not in Virtual-8086 Mode *)
   THEN IF CPL = 0
       THEN
           IF OperandSize = 32;
               THEN
                   EFLAGS ← Pop(); (* 32-bit pop *)
                   (* All non-reserved flags except RF, VIP, VIF, and VM can be modified;
                   VIP, VIF, VM, and all reserved bits are unaffected. RF is cleared. *)
               ELSE IF (Operandsize = 64)
                   RFLAGS = Pop(); (* 64-bit pop *)
                   (* All non-reserved flags except RF, VIP, VIF, and VM can be modified;
                   VIP, VIF, VM, and all reserved bits are unaffected. RF is cleared. *)
               ELSE (* OperandSize = 16 *)
                   EFLAGS[15:0] ← Pop(); (* 16-bit pop *)
                   (* All non-reserved flags can be modified. *)
           FI;
       ELSE (* CPL > 0 *)
           IF OperandSize = 32
               THEN
                   IF CPL > IOPL
                       THEN
                           EFLAGS ← Pop(); (* 32-bit pop *)
                           (* All non-reserved bits except IF, IOPL, VIP, VIF, VM and RF can be modified;
                           IF, IOPL, VIP, VIF, VM and all reserved bits are unaffected; RF is cleared. *)
                       ELSE
                           EFLAGS ← Pop(); (* 32-bit pop *)
                           (* All non-reserved bits except IOPL, VIP, VIF, VM and RF can be modified;
                           IOPL, VIP, VIF, VM and all reserved bits are unaffected; RF is cleared. *)
                   FI;
               ELSE IF (Operandsize = 64)
                   IF CPL > IOPL
                       THEN
                           RFLAGS ← Pop(); (* 64-bit pop *)
                           (* All non-reserved bits except IF, IOPL, VIP, VIF, VM and RF can be modified;
                           IF, IOPL, VIP, VIF, VM and all reserved bits are unaffected; RF is cleared. *)
                       ELSE
                           RFLAGS ← Pop(); (* 64-bit pop *)
                           (* All non-reserved bits except IOPL, VIP, VIF, VM and RF can be modified;
                           IOPL, VIP, VIF, VM and all reserved bits are unaffected; RF is cleared. *)
                   FI;
               ELSE (* OperandSize = 16 *)
                   EFLAGS[15:0] ← Pop(); (* 16-bit pop *)
                   (* All non-reserved bits except IOPL can be modified; IOPL and all
                   reserved bits are unaffected. *)
           FI;
       FI;
   ELSE IF CR4.VME = 1 (* In Virtual-8086 Mode with VME Enabled *)
       IF IOPL = 3
           THEN IF OperandSize = 32
```

```
                THEN
                    EFLAGS ← Pop();
                    (* All non-reserved bits except IOPL, VIP, VIF, VM, and RF can be modified;
                    VIP, VIF, VM, IOPL and all reserved bits are unaffected. RF is cleared. *)
                ELSE
                    EFLAGS[15:0] ← Pop(); FI;
                    (* All non-reserved bits except IOPL can be modified;
                    IOPL and all reserved bits are unaffected. *)
            FI;
        ELSE (* IOPL < 3 *)
            IF (Operandsize = 32)
                THEN
                    #GP(0);  (* Trap to virtual-8086 monitor. *)
                ELSE (* Operandsize = 16 *)
                    tempFLAGS ← Pop();
                    IF EFLAGS.VIP = 1 AND tempFLAGS[9] = 1
                        THEN #GP(0);
                        ELSE
                            EFLAGS.VIF ← tempFLAGS[9];
                            EFLAGS[15:0] ← tempFLAGS;
                        (* All non-reserved bits except IOPL and IF can be modified;
                        IOPL, IF, and all reserved bits are unaffected. *)
                FI;
            FI;
        FI;
    ELSE  (* In Virtual-8086 Mode *)
        IF IOPL = 3
            THEN IF OperandSize = 32
                THEN
                    EFLAGS ← Pop();
                    (* All non-reserved bits except IOPL, VIP, VIF, VM, and RF can be modified;
                    VIP, VIF, VM, IOPL and all reserved bits are unaffected. RF is cleared. *)
                ELSE
                    EFLAGS[15:0] ← Pop(); FI;
                    (* All non-reserved bits except IOPL can be modified;
                    IOPL and all reserved bits are unaffected. *)
        ELSE (* IOPL < 3 *)
            #GP(0);  (* Trap to virtual-8086 monitor. *)
        FI;
    FI;
FI;
```

Flags Affected

All flags may be affected; see the Operation section for details.

Protected Mode Exceptions

#SS(0) If the top of stack is not within the stack segment.

#PF(fault-code) If a page fault occurs.

#AC(0) If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#SS If the top of stack is not within the stack segment.

#UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0) If the I/O privilege level is less than 3.

If an attempt is made to execute the POPF/POPFD instruction with an operand-size override prefix.

#SS(0) If the top of stack is not within the stack segment.

#PF(fault-code) If a page fault occurs.

#AC(0) If an unaligned memory reference is made while alignment checking is enabled.

#UD If the LOCK prefix is used.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

#SS(0) If the stack 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.

...

PREFETCH*h*—Prefetch Data Into Caches

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 18 /1	PREFETCHT0 *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using T0 hint.
0F 18 /2	PREFETCHT1 *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using T1 hint.
0F 18 /3	PREFETCHT2 *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using T2 hint.
0F 18 /0	PREFETCHNTA *m8*	M	Valid	Valid	Move data from *m8* closer to the processor using NTA hint.

Instruction Operand Encoding

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

Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
- T1 (temporal data with respect to first level cache misses)—prefetch data into level 2 cache and higher.
- T2 (temporal data with respect to second level cache misses)—prefetch data into level 3 cache and higher, or an implementation-specific choice.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.

The source operand is a byte memory location. (The locality hints are encoded into the machine level instruction using bits 3 through 5 of the ModR/M byte.)

If the line selected is already present in the cache hierarchy at a level closer to the processor, no data movement occurs. Prefetches from uncacheable or WC memory are ignored.

The PREFETCH*h* instruction is merely a hint and does not affect program behavior. If executed, this instruction moves data closer to the processor in anticipation of future use.

The implementation of prefetch locality hints is implementation-dependent, and can be overloaded or ignored by a processor implementation. The amount of data prefetched is also processor implementation-dependent. It will, however, be a minimum of 32 bytes. Additional details of the implementation-dependent locality hints are described in Section 7.4 of *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

It should be noted that processors are free to speculatively fetch and cache data from system memory regions that are assigned a memory-type that permits speculative reads (that is, the WB, WC, and WT memory types). A PREFETCH*h* instruction is considered a hint to this speculative behavior. Because this speculative fetching can occur at any time and is not tied to instruction execution, a PREFETCH*h* instruction is not ordered with respect to the fence instructions (MFENCE, SFENCE, and LFENCE) or locked memory references. A PREFETCH*h* instruction is also unordered with respect to CLFLUSH and CLFLUSHOPT instructions, other PREFETCH*h* instructions, or any other general instruction. It is ordered with respect to serializing instructions such as CPUID, WRMSR, OUT, and MOV CR.

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

Operation

```
FETCH (m8);
```

Intel C/C++ Compiler Intrinsic Equivalent

```
void _mm_prefetch(char *p, int i)
```

The argument "*p" gives the address of the byte (and corresponding cache line) to be prefetched. The value "i" gives a constant (_MM_HINT_T0, _MM_HINT_T1, _MM_HINT_T2, or _MM_HINT_NTA) that specifies the type of prefetch operation to be performed.

Numeric Exceptions

None.

Exceptions (All Operating Modes)

#UD	If the LOCK prefix is used.

...

PREFETCHW—Prefetch Data into Caches in Anticipation of a Write

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 0D /1 PREFETCHW m8	A	V/V	PRFCHW	Move data from m8 closer to the processor in anticipation of a write.

Instruction Operand Encoding

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

Description

Fetches the cache line of data from memory that contains the byte specified with the source operand to a location in the 1st or 2nd level cache and invalidates other cached instances of the line.

The source operand is a byte memory location. If the line selected is already present in the lowest level cache and is already in an exclusively owned state, no data movement occurs. Prefetches from non-writeback memory are ignored.

The PREFETCHW instruction is merely a hint and does not affect program behavior. If executed, this instruction moves data closer to the processor and invalidates other cached copies in anticipation of the line being written to in the future.

The characteristic of prefetch locality hints is implementation-dependent, and can be overloaded or ignored by a processor implementation. The amount of data prefetched is also processor implementation-dependent. It will, however, be a minimum of 32 bytes. Additional details of the implementation-dependent locality hints are described in Section 7.4 of *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

It should be noted that processors are free to speculatively fetch and cache data with exclusive ownership from system memory regions that permit such accesses (that is, the WB memory type). A PREFETCHW instruction is considered a hint to this speculative behavior. Because this speculative fetching can occur at any time and is not tied to instruction execution, a PREFETCHW instruction is not ordered with respect to the fence instructions (MFENCE, SFENCE, and LFENCE) or locked memory references. A PREFETCHW instruction is also unordered with respect to CLFLUSH and CLFLUSHOPT instructions, other PREFETCHW instructions, or any other general instruction

It is ordered with respect to serializing instructions such as CPUID, WRMSR, OUT, and MOV CR.

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

Operation

```
FETCH_WITH_EXCLUSIVE_OWNERSHIP (m8);
```

Flags Affected

All flags are affected

C/C++ Compiler Intrinsic Equivalent

```
void _m_prefetchw( void * );
```

Protected Mode Exceptions

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#UD If the LOCK prefix is used.

Compatibility Mode Exceptions

#UD If the LOCK prefix is used.

64-Bit Mode Exceptions

#UD If the LOCK prefix is used.

...

PSHUFB — Packed Shuffle Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 00 /r[1] PSHUFB *mm1, mm2/m64*	RM	V/V	SSSE3	Shuffle bytes in *mm1* according to contents of *mm2/m64*.
66 0F 38 00 /r PSHUFB *xmm1, xmm2/m128*	RM	V/V	SSSE3	Shuffle bytes in *xmm1* according to contents of *xmm2/m128*.
VEX.NDS.128.66.0F38.WIG 00 /r VPSHUFB *xmm1, xmm2, xmm3/m128*	RVM	V/V	AVX	Shuffle bytes in *xmm2* according to contents of *xmm3/m128*.
VEX.NDS.256.66.0F38.WIG 00 /r VPSHUFB *ymm1, ymm2, ymm3/m256*	RVM	V/V	AVX2	Shuffle bytes in *ymm2* according to contents of *ymm3/m256*.

NOTES:

1. See note in Section 2.4, "Instruction Exception Specification" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A* and 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 3A*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
RVM	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	NA

Description

PSHUFB performs in-place shuffles of bytes in the destination operand (the first operand) according to the shuffle control mask in the source operand (the second operand). The instruction permutes the data in the destination operand, leaving the shuffle mask unaffected. If the most significant bit (bit[7]) of each byte of the shuffle control mask is set, then constant zero is written in the result byte. Each byte in the shuffle control mask forms an index to permute the corresponding byte in the destination operand. The value of each index is the least significant 4 bits (128-bit operation) or 3 bits (64-bit operation) of the shuffle control byte. When the source operand is a 128-bit memory operand, the operand must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated.

In 64-bit mode, use the REX prefix to access additional registers.

Legacy SSE version: Both operands can be MMX registers.

128-bit Legacy SSE version: The first source operand and the destination operand are the same. Bits (VLMAX-1:128) of the corresponding YMM destination register remain unchanged.

VEX.128 encoded version: The destination operand is the first operand, the first source operand is the second operand, the second source operand is the third operand. Bits (VLMAX-1:128) of the destination YMM register are zeroed.

VEX.256 encoded version: Bits (255:128) of the destination YMM register stores the 16-byte shuffle result of the upper 16 bytes of the first source operand, using the upper 16-bytes of the second source operand as control mask. The value of each index is for the high 128-bit lane is the least significant 4 bits of the respective shuffle control byte. The index value selects a source data element within each 128-bit lane.

Note: VEX.L must be 0, otherwise the instruction will #UD.

Operation

PSHUFB (with 64 bit operands)

```
TEMP ← DEST
for i = 0 to 7 {
    if (SRC[(i * 8)+7] = 1 ) then
        DEST[(i*8)+7...(i*8)+0] ← 0;
    else
        index[2..0] ← SRC[(i*8)+2 .. (i*8)+0];
        DEST[(i*8)+7...(i*8)+0] ← TEMP[(index*8+7)..(index*8+0)];
    endif;
}
```

PSHUFB (with 128 bit operands)

```
TEMP ← DEST
for i = 0 to 15 {
    if (SRC[(i * 8)+7] = 1 ) then
        DEST[(i*8)+7..(i*8)+0] ← 0;
     else
        index[3..0] ← SRC[(i*8)+3 .. (i*8)+0];
        DEST[(i*8)+7..(i*8)+0] ← TEMP[(index*8+7)..(index*8+0)];
    endif
}
```

VPSHUFB (VEX.128 encoded version)

```
for i = 0 to 15 {
    if (SRC2[(i * 8)+7] = 1) then
        DEST[(i*8)+7..(i*8)+0] ← 0;
        else
        index[3..0] ← SRC2[(i*8)+3 .. (i*8)+0];
        DEST[(i*8)+7..(i*8)+0] ← SRC1[(index*8+7)..(index*8+0)];
    endif
}
DEST[VLMAX-1:128] ← 0
```

VPSHUFB (VEX.256 encoded version)

```
for i = 0 to 15 {
    if (SRC2[(i * 8)+7] == 1 ) then
        DEST[(i*8)+7..(i*8)+0] ← 0;
        else
        index[3..0] ← SRC2[(i*8)+3 .. (i*8)+0];
        DEST[(i*8)+7..(i*8)+0] ← SRC1[(index*8+7)..(index*8+0)];
    endif
    if (SRC2[128 + (i * 8)+7] == 1 ) then
        DEST[128 + (i*8)+7..(i*8)+0] ← 0;
        else
        index[3..0] ← SRC2[128 + (i*8)+3 .. (i*8)+0];
        DEST[128 + (i*8)+7..(i*8)+0] ← SRC1[128 + (index*8+7)..(index*8+0)];
    endif
}
```

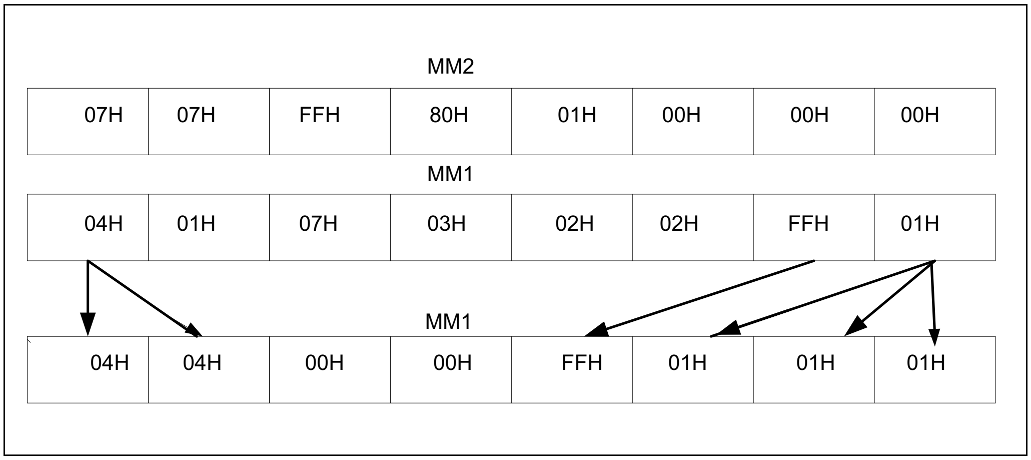

MM2
07H 07H FFH 80H 01H 00H 00H 00H
MM1
04H 01H 07H 03H 02H 02H FFH 01H
MM1
04H 04H 00H 00H FFH 01H 01H 01H

Figure 4-11 PSHUFB with 64-Bit Operands

Intel C/C++ Compiler Intrinsic Equivalent

PSHUFB:	__m64 _mm_shuffle_pi8 (__m64 a, __m64 b)
(V)PSHUFB:	__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
VPSHUFB:	__m256i _mm256_shuffle_epi8(__m256i a, __m256i b)

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally

#UD If VEX.L = 1.

...

RDPID—Read Processor ID

Opcode/ Instruction	Op/ En	64/32- bit Mode	CPUID Feature Flag	Description
F3 0F C7 /7 RDPID r32	M	N.E./V	RDPID	Read IA32_TSC_AUX into r32.
F3 0F C7 /7 RDPID r64	M	V/N.E.	RDPID	Read IA32_TSC_AUX into r64.

Instruction Operand Encoding

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

Description

Reads the value of the IA32_TSC_AUX MSR (address C0000103H) into the destination register. The value of CS.D and operand-size prefixes (66H and REX.W) do not affect the behavior of the RDPID instruction.

Operation

```
DEST ← IA32_TSC_AUX
```

Flags Affected

None.

Protected Mode Exceptions

#UD	If the LOCK prefix is used. If the F2 prefix is used. If CPUID.7H.0:ECX.RDPID[bit 22] = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

RDPKRU—Read Protection Key Rights for User Pages

Opcode*	Instruction	Op/ En	64/32bit Mode Support	CPUID Feature Flag	Description
0F 01 EE	RDPKRU	NP	V/V	OSPKE	Reads PKRU into EAX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Reads the value of PKRU into EAX and clears EDX. ECX must be 0 when RDPKRU is executed; otherwise, a general-protection exception (#GP) occurs.

RDPKRU can be executed only if CR4.PKE = 1; otherwise, an invalid-opcode exception (#UD) occurs. Software can discover the value of CR4.PKE by examining CPUID.(EAX=07H,ECX=0H):ECX.OSPKE [bit 4].

On processors that support the Intel 64 Architecture, the high-order 32-bits of RCX are ignored and the high-order 32-bits of RDX and RAX are cleared.

Operation

```
IF (ECX = 0)
    THEN
         EAX ← PKRU;
         EDX ← 0;
    ELSE #GP(0);
FI;
```

Flags Affected

None.

C/C++ Compiler Intrinsic Equivalent

RDPKRU:	uint32_t _rdpkru_u32(void);

Protected Mode Exceptions

#GP(0)	If ECX ≠ 0
#UD	If the LOCK prefix is used. If CR4.PKE = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

RDPMC—Read Performance-Monitoring Counters

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 33	RDPMC	NP	Valid	Valid	Read performance-monitoring counter specified by ECX into EDX:EAX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

The EAX register is loaded with the low-order 32 bits. The EDX register is loaded with the supported high-order bits of the counter. The number of high-order bits loaded into EDX is implementation specific on processors that do no support architectural performance monitoring. The width of fixed-function and general-purpose performance counters on processors supporting architectural performance monitoring are reported by CPUID 0AH leaf. See below for the treatment of the EDX register for “fast” reads.

The ECX register specifies the counter type (if the processor supports architectural performance monitoring) and counter index. Counter type is specified in ECX[30] to select one of two type of performance counters. If the processor does not support architectural performance monitoring, ECX[30:0] specifies the counter index; otherwise ECX[29:0] specifies the index relative to the base of each counter type. ECX[31] selects “fast” read mode if supported. The two counter types are :

- General-purpose or special-purpose performance counters are specified with ECX[30] = 0: The number of general-purpose performance counters on processor supporting architectural performance monitoring are reported by CPUID 0AH leaf. The number of general-purpose counters is model specific if the processor does not support architectural performance monitoring, see Chapter 18, “Performance Monitoring” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B*. Special-purpose counters are available only in selected processor members, see Table 4-13.
- Fixed-function performance counter are specified with ECX[30] = 1. The number fixed-function performance counters is enumerated by CPUID 0AH leaf. See Chapter 30 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B*. This counter type is selected if ECX[30] is set.

The width of fixed-function performance counters and general-purpose performance counters on processor supporting architectural performance monitoring are reported by CPUID 0AH leaf. The width of general-purpose performance counters are 40-bits for processors that do not support architectural performance monitoring counters. The width of special-purpose performance counters are implementation specific.

Table 4-13 lists valid indices of the general-purpose and special-purpose performance counters according to the DisplayFamily_DisplayModel values of CPUID encoding for each processor family (see CPUID instruction in Chapter 3, “Instruction Set Reference, A-M” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*).

Table 4-13 Valid General and Special Purpose Performance Counter Index Range for RDPMC

Processor Family	DisplayFamily_DisplayModel/ Other Signatures	Valid PMC Index Range	General-purpose Counters
P6	06H_01H, 06H_03H, 06H_05H, 06H_06H, 06H_07H, 06H_08H, 06H_0AH, 06H_0BH	0, 1	0, 1
Processors Based on Intel NetBurst microarchitecture (No L3)	0FH_00H, 0FH_01H, 0FH_02H, 0FH_03H, 0FH_04H, 0FH_06H	≥ 0 and ≤ 17	≥ 0 and ≤ 17
Pentium M processors	06H_09H, 06H_0DH	0, 1	0, 1
Processors Based on Intel NetBurst microarchitecture (No L3)	0FH_03H, 0FH_04H) and (L3 is present)	≥ 0 and ≤ 25	≥ 0 and ≤ 17
Intel® Core™ Solo and Intel® Core™ Duo processors, Dual-core Intel® Xeon® processor LV	06H_0EH	0, 1	0, 1
Intel® Core™2 Duo processor, Intel Xeon processor 3000, 5100, 5300, 7300 Series - general-purpose PMC	06H_0FH	0, 1	0, 1
Intel® Core™2 Duo processor family, Intel Xeon processor 3100, 3300, 5200, 5400 series - general-purpose PMC	06H_17H	0, 1	0, 1
Intel Xeon processors 7400 series	(06H_1DH)	≥ 0 and ≤ 9	0, 1
45 nm and 32 nm Intel® Atom™ processors	06H_1CH, 06_26H, 06_27H, 06_35H, 06_36H	0, 1	0, 1
Intel® Atom™ processors based on Silvermont or Airmont microarchitectures	06H_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_4CH	0, 1	0, 1
Next Generation Intel® Atom™ processors based on Goldmont microarchitecture	06H_5CH, 06_5FH	0-3	0-3
Intel® processors based on the Nehalem, Westmere microarchitectures	06H_1AH, 06H_1EH, 06H_1FH, 06_25H, 06_2CH, 06H_2EH, 06_2FH	0-3	0-3
Intel® processors based on the Sandy Bridge, Ivy Bridge microarchitecture	06H_2AH, 06H_2DH, 06H_3AH, 06H_3EH	0-3 (0-7 if HyperThreading is off)	0-3 (0-7 if HyperThreading is off)
Intel® processors based on the Haswell, Broadwell, SkyLake microarchitectures	06H_3CH, 06H_45H, 06H_46H, 06H_3FH, 06_3DH, 06_47H, 4FH, 06_56H, 06_4EH, 06_5EH	0-3 (0-7 if HyperThreading is off)	0-3 (0-7 if HyperThreading is off)

Processors based on Intel NetBurst microarchitecture support “fast” (32-bit) and “slow” (40-bit) reads on the first 18 performance counters. Selected this option using ECX[31]. If bit 31 is set, RDPMC reads only the low 32 bits of the selected performance counter. If bit 31 is clear, all 40 bits are read. A 32-bit result is returned in EAX and EDX is set to 0. A 32-bit read executes faster on these processors than a full 40-bit read.

On processors based on Intel NetBurst microarchitecture with L3, performance counters with indices 18-25 are 32-bit counters. EDX is cleared after executing RDPMC for these counters.

In Intel Core 2 processor family, Intel Xeon processor 3000, 5100, 5300 and 7400 series, the fixed-function performance counters are 40-bits wide; they can be accessed by RDMPC with ECX between from 4000_0000H and 4000_0002H.

On Intel Xeon processor 7400 series, there are eight 32-bit special-purpose counters addressable with indices 2-9, ECX[30]=0.

When in protected or virtual 8086 mode, the performance-monitoring counters enabled (PCE) flag in register CR4 restricts the use of the RDPMC instruction as follows. When the PCE flag is set, the RDPMC instruction can be executed at any privilege level; when the flag is clear, the instruction can only be executed at privilege level 0. (When in real-address mode, the RDPMC instruction is always enabled.)

The performance-monitoring counters can also be read with the RDMSR instruction, when executing at privilege level 0.

The performance-monitoring counters are event counters that can be programmed to count events such as the number of instructions decoded, number of interrupts received, or number of cache loads. Chapter 19, "Performance Monitoring Events," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, lists the events that can be counted for various processors in the Intel 64 and IA-32 architecture families.

The RDPMC instruction is not a serializing instruction; that is, it does not imply that all the events caused by the preceding instructions have been completed or that events caused by subsequent instructions have not begun. If an exact event count is desired, software must insert a serializing instruction (such as the CPUID instruction) before and/or after the RDPMC instruction.

Performing back-to-back fast reads are not guaranteed to be monotonic. To guarantee monotonicity on back-to-back reads, a serializing instruction must be placed between the two RDPMC instructions.

The RDPMC instruction can execute in 16-bit addressing mode or virtual-8086 mode; however, the full contents of the ECX register are used to select the counter, and the event count is stored in the full EAX and EDX registers. The RDPMC instruction was introduced into the IA-32 Architecture in the Pentium Pro processor and the Pentium processor with MMX technology. The earlier Pentium processors have performance-monitoring counters, but they must be read with the RDMSR instruction.

Operation

```
(* Intel processors that support architectural performance monitoring *)

Most significant counter bit (MSCB) = 47

IF ((CR4.PCE = 1) or (CPL = 0) or (CR0.PE = 0))
   THEN IF (ECX[30] = 1 and ECX[29:0] in valid fixed-counter range)
         EAX ← IA32_FIXED_CTR(ECX)[30:0];
         EDX ← IA32_FIXED_CTR(ECX)[MSCB:32];
   ELSE IF (ECX[30] = 0 and ECX[29:0] in valid general-purpose counter range)
         EAX ← PMC(ECX[30:0])[31:0];
         EDX ← PMC(ECX[30:0])[MSCB:32];
   ELSE (* ECX is not valid or CR4.PCE is 0 and CPL is 1, 2, or 3 and CR0.PE is 1 *)
         #GP(0);
FI;

(* Intel Core 2 Duo processor family and Intel Xeon processor 3000, 5100, 5300, 7400 series*)

Most significant counter bit (MSCB) = 39

IF ((CR4.PCE = 1) or (CPL = 0) or (CR0.PE = 0))
   THEN IF (ECX[30] = 1 and ECX[29:0] in valid fixed-counter range)
         EAX ← IA32_FIXED_CTR(ECX)[30:0];
         EDX ← IA32_FIXED_CTR(ECX)[MSCB:32];
   ELSE IF (ECX[30] = 0 and ECX[29:0] in valid general-purpose counter range)
         EAX ← PMC(ECX[30:0])[31:0];
         EDX ← PMC(ECX[30:0])[MSCB:32];
   ELSE IF (ECX[30] = 0 and ECX[29:0] in valid special-purpose counter range)
         EAX ← PMC(ECX[30:0])[31:0]; (* 32-bit read *)
```

```
    ELSE (* ECX is not valid or CR4.PCE is 0 and CPL is 1, 2, or 3 and CR0.PE is 1 *)
        #GP(0);
FI;

(* P6 family processors and Pentium processor with MMX technology *)

IF (ECX = 0 or 1) and ((CR4.PCE = 1) or (CPL = 0) or (CR0.PE = 0))
    THEN
        EAX ← PMC(ECX)[31:0];
        EDX ← PMC(ECX)[39:32];
    ELSE (* ECX is not 0 or 1 or CR4.PCE is 0 and CPL is 1, 2, or 3 and CR0.PE is 1 *)
        #GP(0);
FI;
(* Processors based on Intel NetBurst microarchitecture *)
IF ((CR4.PCE = 1) or (CPL = 0) or (CR0.PE = 0))
    THEN IF (ECX[30:0] = 0:17)
        THEN IF ECX[31] = 0
            THEN
                EAX ← PMC(ECX[30:0])[31:0]; (* 40-bit read *)
                EDX ← PMC(ECX[30:0])[39:32];
        ELSE (* ECX[31] = 1*)
            THEN
                EAX ← PMC(ECX[30:0])[31:0]; (* 32-bit read *)
                EDX ← 0;
        FI;
    ELSE IF (*64-bit Intel processor based on Intel NetBurst microarchitecture with L3 *)
        THEN IF (ECX[30:0] = 18:25 )
            EAX ← PMC(ECX[30:0])[31:0]; (* 32-bit read *)
            EDX ← 0;
        FI;
    ELSE (* Invalid PMC index in ECX[30:0], see Table 4-16. *)
        GP(0);
    FI;
ELSE (* CR4.PCE = 0 and (CPL = 1, 2, or 3) and CR0.PE = 1 *)
    #GP(0);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0 and the PCE flag in the CR4 register is clear.

If an invalid performance counter index is specified (see Table 4-13).

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP If an invalid performance counter index is specified (see Table 4-13).

#UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If the PCE flag in the CR4 register is clear.
	If an invalid performance counter index is specified (see Table 4-13).
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0 and the PCE flag in the CR4 register is clear.
	If an invalid performance counter index is specified (see Table 4-13).
#UD	If the LOCK prefix is used.

...

RDTSC—Read Time-Stamp Counter

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 31	RDTSC	NP	Valid	Valid	Read time-stamp counter into EDX:EAX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Reads the current value of the processor's time-stamp counter (a 64-bit MSR) into the EDX:EAX registers. The EDX register is loaded with the high-order 32 bits of the MSR and the EAX register is loaded with the low-order 32 bits. (On processors that support the Intel 64 architecture, the high-order 32 bits of each of RAX and RDX are cleared.)

The processor monotonically increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset. See "Time Stamp Counter" in Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, for specific details of the time stamp counter behavior.

The time stamp disable (TSD) flag in register CR4 restricts the use of the RDTSC instruction as follows. When the flag is clear, the RDTSC instruction can be executed at any privilege level; when the flag is set, the instruction can only be executed at privilege level 0.

The time-stamp counter can also be read with the RDMSR instruction, when executing at privilege level 0.

The RDTSC instruction is not a serializing instruction. It does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the read operation is performed. If software requires RDTSC to be executed only after all previous instructions have completed locally, it can either use RDTSCP (if the processor supports that instruction) or execute the sequence LFENCE;RDTSC.

This instruction was introduced by the Pentium processor.

See "Changes to Instruction Behavior in VMX Non-Root Operation" in Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
IF (CR4.TSD = 0) or (CPL = 0) or (CR0.PE = 0)
    THEN EDX:EAX ← TimeStampCounter;
    ELSE (* CR4.TSD = 1 and (CPL = 1, 2, or 3) and CR0.PE = 1 *)
        #GP(0);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the TSD flag in register CR4 is set and the CPL is greater than 0.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If the TSD flag in register CR4 is set.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

RDTSCP—Read Time-Stamp Counter and Processor ID

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 01 F9	RDTSCP	NP	Valid	Valid	Read 64-bit time-stamp counter and IA32_TSC_AUX value into EDX:EAX and ECX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Reads the current value of the processor's time-stamp counter (a 64-bit MSR) into the EDX:EAX registers and also reads the value of the IA32_TSC_AUX MSR (address C0000103H) into the ECX register. The EDX register is loaded with the high-order 32 bits of the IA32_TSC MSR; the EAX register is loaded with the low-order 32 bits of the IA32_TSC MSR; and the ECX register is loaded with the low-order 32-bits of IA32_TSC_AUX MSR. On processors that support the Intel 64 architecture, the high-order 32 bits of each of RAX, RDX, and RCX are cleared.

The processor monotonically increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset. See "Time Stamp Counter" in Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, for specific details of the time stamp counter behavior.

The time stamp disable (TSD) flag in register CR4 restricts the use of the RDTSCP instruction as follows. When the flag is clear, the RDTSCP instruction can be executed at any privilege level; when the flag is set, the instruction can only be executed at privilege level 0.

The RDTSCP instruction waits until all previous instructions have been executed before reading the counter. However, subsequent instructions may begin execution before the read operation is performed.

See "Changes to Instruction Behavior in VMX Non-Root Operation" in Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
IF (CR4.TSD = 0) or (CPL = 0) or (CR0.PE = 0)
    THEN
        EDX:EAX ← TimeStampCounter;
        ECX ← IA32_TSC_AUX[31:0];
    ELSE (* CR4.TSD = 1 and (CPL = 1, 2, or 3) and CR0.PE = 1 *)
        #GP(0);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the TSD flag in register CR4 is set and the CPL is greater than 0.
#UD	If the LOCK prefix is used. If CPUID.80000001H:EDX.RDTSCP[bit 27] = 0.

Real-Address Mode Exceptions

#UD	If the LOCK prefix is used. If CPUID.80000001H:EDX.RDTSCP[bit 27] = 0.

Virtual-8086 Mode Exceptions

#GP(0)	If the TSD flag in register CR4 is set.
#UD	If the LOCK prefix is used. If CPUID.80000001H:EDX.RDTSCP[bit 27] = 0.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

ROUNDPD — Round Packed Double Precision Floating-Point Values

Opcode*/ Instruction	**Op/ En**	**64/32 bit Mode Support**	**CPUID Feature Flag**	**Description**
66 0F 3A 09 /r ib ROUNDPD *xmm1, xmm2/m128, imm8*	RMI	V/V	SSE4_1	Round packed double precision floating-point values in *xmm2/m128* and place the result in *xmm1*. The rounding mode is determined by *imm8*.
VEX.128.66.0F3A.WIG 09 /r ib VROUNDPD *xmm1, xmm2/m128, imm8*	RMI	V/V	AVX	Round packed double-precision floating-point values in *xmm2/m128* and place the result in *xmm1*. The rounding mode is determined by *imm8*.
VEX.256.66.0F3A.WIG 09 /r ib VROUNDPD *ymm1, ymm2/m256, imm8*	RMI	V/V	AVX	Round packed double-precision floating-point values in *ymm2/m256* and place the result in *ymm1*. The rounding mode is determined by *imm8*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMI	ModRM:reg (w)	ModRM:r/m (r)	imm8	NA

Description

Round the 2 double-precision floating-point values in the source operand (second operand) using the rounding mode specified in the immediate operand (third operand) and place the results in the destination operand (first operand). The rounding process rounds each input floating-point value to an integer value and returns the integer result as a double-precision floating-point value.

The immediate operand specifies control fields for the rounding operation, three bit fields are defined and shown in Figure 4-20. Bit 3 of the immediate byte controls processor behavior for a precision exception, bit 2 selects the source of rounding mode control. Bits 1:0 specify a non-sticky rounding-mode value (Table 4-15 lists the encoded values for rounding-mode field).

The Precision Floating-Point Exception is signaled according to the immediate operand. If any source operand is an SNaN then it will be converted to a QNaN. If DAZ is set to ‘1 then denormals will be converted to zero before rounding.

128-bit Legacy SSE version: The second source can be an XMM register or 128-bit memory location. The destination is not distinct from the first source XMM register and the upper bits (VLMAX-1:128) of the corresponding YMM register destination are unmodified.

VEX.128 encoded version: the source operand second source operand or a 128-bit memory location. The destination operand is an XMM register. The upper bits (VLMAX-1:128) of the corresponding YMM register destination are zeroed.

VEX.256 encoded version: The source operand is a YMM register or a 256-bit memory location. The destination operand is a YMM register.

Note: In VEX-encoded versions, VEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

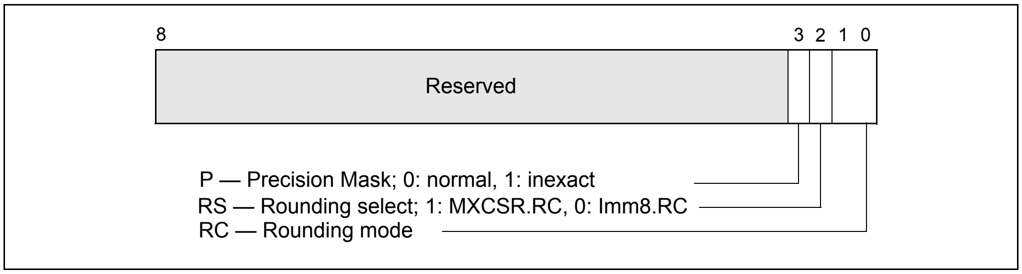

8
3 2 1 0
Reserved
P — Precision Mask; 0: normal, 1: inexact
RS — Rounding select; 1: MXCSR.RC, 0: Imm8.RC
RC — Rounding mode

Figure 4-20 Bit Control Fields of Immediate Byte for ROUNDxx Instruction

Table 4-15 Rounding Modes and Encoding of Rounding Control (RC) Field

Rounding Mode	RC Field Setting	Description
Round to nearest (even)	00B	Rounded result is the closest to the infinitely precise result. If two values are equally close, the result is the even value (i.e., the integer value with the least-significant bit of zero).
Round down (toward –∞)	01B	Rounded result is closest to but no greater than the infinitely precise result.
Round up (toward +∞)	10B	Rounded result is closest to but no less than the infinitely precise result.
Round toward zero (Truncate)	11B	Rounded result is closest to but no greater in absolute value than the infinitely precise result.

Operation

```
IF (imm[2] = ‘1)
    THEN    // rounding mode is determined by MXCSR.RC
        DEST[63:0] ← ConvertDPFPToInteger_M(SRC[63:0]);
        DEST[127:64] ← ConvertDPFPToInteger_M(SRC[127:64]);
    ELSE    // rounding mode is determined by IMM8.RC
        DEST[63:0] ← ConvertDPFPToInteger_Imm(SRC[63:0]);
        DEST[127:64] ← ConvertDPFPToInteger_Imm(SRC[127:64]);
FI
```

ROUNDPD (128-bit Legacy SSE version)

```
DEST[63:0] ← RoundToInteger(SRC[63:0]], ROUND_CONTROL)
DEST[127:64] ← RoundToInteger(SRC[127:64]], ROUND_CONTROL)
DEST[VLMAX-1:128] (Unmodified)
```

VROUNDPD (VEX.128 encoded version)

```
DEST[63:0] ← RoundToInteger(SRC[63:0]], ROUND_CONTROL)
DEST[127:64] ← RoundToInteger(SRC[127:64]], ROUND_CONTROL)
DEST[VLMAX-1:128] ← 0
```

VROUNDPD (VEX.256 encoded version)

```
DEST[63:0] ← RoundToInteger(SRC[63:0], ROUND_CONTROL)
DEST[127:64] ← RoundToInteger(SRC[127:64]], ROUND_CONTROL)
DEST[191:128] ← RoundToInteger(SRC[191:128]], ROUND_CONTROL)
DEST[255:192] ← RoundToInteger(SRC[255:192] ], ROUND_CONTROL)
```

Intel C/C++ Compiler Intrinsic Equivalent

```
__m128 _mm_round_pd(__m128d s1, int iRoundMode);
__m128 _mm_floor_pd(__m128d s1);
__m128 _mm_ceil_pd(__m128d s1)
__m256 _mm256_round_pd(__m256d s1, int iRoundMode);
__m256 _mm256_floor_pd(__m256d s1);
__m256 _mm256_ceil_pd(__m256d s1)
```

SIMD Floating-Point Exceptions

Invalid (signaled only if SRC = SNaN)

Precision (signaled only if imm[3] = ‘0; if imm[3] = ‘1, then the Precision Mask in the MXSCSR is ignored and precision exception is not signaled.)

Note that Denormal is not signaled by ROUNDPD.

Other Exceptions

See Exceptions Type 2; additionally

#UD	If VEX.vvvv ≠ 1111B.

...

SGDT—Store Global Descriptor Table Register

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 01 /0	SGDT *m*	M	Valid	Valid	Store GDTR to *m.*

NOTES:

* See IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

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

Description

Stores the content of the global descriptor table register (GDTR) in the destination operand. The destination operand specifies a memory location.

In legacy or compatibility mode, the destination operand is a 6-byte memory location. If the operand-size attribute is 16 bits, the limit is stored in the low 2 bytes and the 24-bit base address is stored in bytes 3-5, and byte 6 is zero-filled. If the operand-size attribute is 32 bits, the 16-bit limit field of the register is stored in the low 2 bytes of the memory location and the 32-bit base address is stored in the high 4 bytes.

In IA-32e mode, the operand size is fixed at 8+2 bytes. The instruction stores an 8-byte base and a 2-byte limit.

SGDT is useful only by operating-system software. However, it can be used in application programs without causing an exception to be generated if CR4.UMIP = 0. See "LGDT/LIDT—Load Global/Interrupt Descriptor Table Register" in Chapter 3, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 2A*, for information on loading the GDTR and IDTR.

IA-32 Architecture Compatibility

The 16-bit form of the SGDT is compatible with the Intel 286 processor if the upper 8 bits are not referenced. The Intel 286 processor fills these bits with 1s; processor generations later than the Intel 286 processor fill these bits with 0s.

Operation

```
IF instruction is SGDT
        IF OperandSize = 16
            THEN
                DEST[0:15] ← GDTR(Limit);
                DEST[16:39] ← GDTR(Base); (* 24 bits of base address stored *)
                DEST[40:47] ← 0;
            ELSE IF (32-bit Operand Size)
                DEST[0:15] ← GDTR(Limit);
                DEST[16:47] ← GDTR(Base); (* Full 32-bit base address stored *)
                FI;
            ELSE (* 64-bit Operand Size *)
                DEST[0:15] ← GDTR(Limit);
                DEST[16:79] ← GDTR(Base); (* Full 64-bit base address stored *)
        FI;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#UD If the destination operand is a register.

If the LOCK prefix is used.

#GP(0) If the destination is located in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.

If CR4.UMIP = 1 and CPL > 0.

#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 CPL = 3.

Real-Address Mode Exceptions

#UD If the destination operand is a register.

If the LOCK prefix is used.

#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.

Virtual-8086 Mode Exceptions

#UD If the destination operand is a register.

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.

If CR4.UMIP = 1.

#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

#SS(0) If a memory address referencing the SS segment is in a non-canonical form.

#UD If the destination operand is a register.

If the LOCK prefix is used.

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

If CR4.UMIP = 1 and CPL > 0.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while CPL = 3.

...

SHA1RNDS4—Perform Four Rounds of SHA1 Operation

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 3A CC /r ib SHA1RNDS4 xmm1, xmm2/m128, imm8	RMI	V/V	SHA	Performs four rounds of SHA1 operation operating on SHA1 state (A,B,C,D) from xmm1, with a pre-computed sum of the next 4 round message dwords and state variable E from xmm2/m128. The immediate byte controls logic functions and round constants.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	Imm8

Description

The SHA1RNDS4 instruction performs four rounds of SHA1 operation using an initial SHA1 state (A,B,C,D) from the first operand (which is a source operand and the destination operand) and some pre-computed sum of the next 4 round message dwords, and state variable E from the second operand (a source operand). The updated SHA1 state (A,B,C,D) after four rounds of processing is stored in the destination operand.

Operation

SHA1RNDS4

The function f() and Constant K are dependent on the value of the immediate.

```
IF ( imm8[1:0] = 0 )
    THEN f() ← f0(), K ← $K_0$;
ELSE IF ( imm8[1:0] = 1 )
    THEN f() ← f1(), K ← $K_1$;
ELSE IF ( imm8[1:0] = 2 )
    THEN f() ← f2(), K ← $K_2$;
ELSE IF ( imm8[1:0] = 3 )
    THEN f() ← f3(), K ← K3;
FI;

A ← SRC1[127:96];
B ← SRC1[95:64];
C ← SRC1[63:32];
D ← SRC1[31:0];
$W_0$E ← SRC2[127:96];
$W_1$ ← SRC2[95:64];
$W_2$ ← SRC2[63:32];
$W_3$ ← SRC2[31:0];

Round i = 0 operation:
A_1 ← f (B, C, D) + (A ROL 5) +$W_0$E +K;
B_1 ← A;
C_1 ← B ROL 30;
D_1 ← C;
E_1 ← D;
```

```
FOR i = 1 to 3
    A_(i +1) ← f (B_i, C_i, D_i) + (A_i ROL 5) +$W_i$+ E_i +K;
    B_(i +1) ← A_i;
    C_(i +1) ← B_i ROL 30;
    D_(i +1) ← C_i;
    E_(i +1) ← D_i;
ENDFOR

DEST[127:96] ← A_4;
DEST[95:64] ← B_4;
DEST[63:32] ← C_4;
DEST[31:0] ← D_4;
```

Intel C/C++ Compiler Intrinsic Equivalent

SHA1RNDS4: __m128i _mm_sha1rnds4_epu32(__m128i, __m128i, const int);

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

...

SHA1NEXTE—Calculate SHA1 State Variable E after Four Rounds

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 C8 /r SHA1NEXTE xmm1, xmm2/m128	RM	V/V	SHA	Calculates SHA1 state variable E after four rounds of operation from the current SHA1 state variable A in xmm1. The calculated value of the SHA1 state variable E is added to the scheduled dwords in xmm2/m128, and stored with some of the scheduled dwords in xmm1.

Instruction Operand Encoding

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

Description

The SHA1NEXTE calculates the SHA1 state variable E after four rounds of operation from the current SHA1 state variable A in the destination operand. The calculated value of the SHA1 state variable E is added to the source operand, which contains the scheduled dwords.

Operation

SHA1NEXTE

```
TMP ← (SRC1[127:96] ROL 30);

DEST[127:96] ← SRC2[127:96] + TMP;
DEST[95:64] ← SRC2[95:64];
DEST[63:32] ← SRC2[63:32];
DEST[31:0] ← SRC2[31:0];
```

Intel C/C++ Compiler Intrinsic Equivalent

SHA1NEXTE: __m128i _mm_sha1nexte_epu32(__m128i, __m128i);

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

...

SHA1MSG1—Perform an Intermediate Calculation for the Next Four SHA1 Message Dwords

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 C9 /r SHA1MSG1 xmm1, xmm2/m128	RM	V/V	SHA	Performs an intermediate calculation for the next four SHA1 message dwords using previous message dwords from xmm1 and xmm2/m128, storing the result in xmm1.

Instruction Operand Encoding

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

Description

The SHA1MSG1 instruction is one of two SHA1 message scheduling instructions. The instruction performs an intermediate calculation for the next four SHA1 message dwords.

Operation

SHA1MSG1

```
W0 ← SRC1[127:96] ;
W1 ← SRC1[95:64] ;
W2 ← SRC1[63: 32] ;
W3 ← SRC1[31: 0] ;
```

```
W4 ← SRC2[127:96] ;
W5 ← SRC2[95:64] ;

DEST[127:96] ← W2 XOR W0;
DEST[95:64] ← W3 XOR W1;
DEST[63:32] ← W4 XOR W2;
DEST[31:0] ← W5 XOR W3;
```

Intel C/C++ Compiler Intrinsic Equivalent

SHA1MSG1: __m128i _mm_sha1msg1_epu32(__m128i, __m128i);

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

...

SHA1MSG2—Perform a Final Calculation for the Next Four SHA1 Message Dwords

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 CA /r SHA1MSG2 xmm1, xmm2/m128	RM	V/V	SHA	Performs the final calculation for the next four SHA1 message dwords using intermediate results from xmm1 and the previous message dwords from xmm2/m128, storing the result in xmm1.

Instruction Operand Encoding

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

Description

The SHA1MSG2 instruction is one of two SHA1 message scheduling instructions. The instruction performs the final calculation to derive the next four SHA1 message dwords.

Operation

SHA1MSG2

```
W13 ← SRC2[95:64] ;
W14 ← SRC2[63: 32] ;
W15 ← SRC2[31: 0] ;
W16 ← (SRC1[127:96] XOR W13 ) ROL 1;
W17 ← (SRC1[95:64] XOR W14) ROL 1;
W18 ← (SRC1[63: 32] XOR W15) ROL 1;
```

```
W19 ← (SRC1[31: 0] XOR W16) ROL 1;

DEST[127:96] ← W16;
DEST[95:64] ← W17;
DEST[63:32] ← W18;
DEST[31:0] ← W19;
```

Intel C/C++ Compiler Intrinsic Equivalent

SHA1MSG2: __m128i _mm_sha1msg2_epu32(__m128i, __m128i);

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

...

SHA256RNDS2—Perform Two Rounds of SHA256 Operation

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 CB /r SHA256RNDS2 xmm1, xmm2/m128, <XMM0>	RM0	V/V	SHA	Perform 2 rounds of SHA256 operation using an initial SHA256 state (C,D,G,H) from xmm1, an initial SHA256 state (A,B,E,F) from xmm2/m128, and a pre-computed sum of the next 2 round message dwords and the corresponding round constants from the implicit operand XMM0, storing the updated SHA256 state (A,B,E,F) result in xmm1.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	Implicit XMM0 (r)

Description

The SHA256RNDS2 instruction performs 2 rounds of SHA256 operation using an initial SHA256 state (C,D,G,H) from the first operand, an initial SHA256 state (A,B,E,F) from the second operand, and a pre-computed sum of the next 2 round message dwords and the corresponding round constants from the implicit operand xmm0. Note that only the two lower dwords of XMM0 are used by the instruction.

The updated SHA256 state (A,B,E,F) is written to the first operand, and the second operand can be used as the updated state (C,D,G,H) in later rounds.

Operation

SHA256RNDS2

```
A_0 ← SRC2[127:96];
B_0 ← SRC2[95:64];
C_0 ← SRC1[127:96];
D_0 ← SRC1[95:64];
E_0 ← SRC2[63:32];
F_0 ← SRC2[31:0];
G_0 ← SRC1[63:32];
H_0 ← SRC1[31:0];
$WK_0$ ← XMM0[31: 0];
$WK_1$ ← XMM0[63: 32];

FOR i = 0 to 1
    A_(i +1) ← Ch (E_i, F_i, G_i) +$\Sigma_1$( E_i) +$WK_i$+ H_i + Maj(A_i , B_i, C_i) +$\Sigma_0$( A_i);
    B_(i +1) ← A_i;
    C_(i +1) ← B_i ;
    D_(i +1) ← C_i;
    E_(i +1) ← Ch (E_i, F_i, G_i) +$\Sigma_1$( E_i) +$WK_i$+ H_i + D_i;
    F_(i +1) ← E_i ;
    G_(i +1) ← F_i;
    H_(i +1) ← G_i;
ENDFOR

DEST[127:96] ← A_2;
DEST[95:64] ← B_2;
DEST[63:32] ← E_2;
DEST[31:0] ← F_2;
```

Intel C/C++ Compiler Intrinsic Equivalent

SHA256RNDS2: __m128i _mm_sha256rnds2_epu32(__m128i, __m128i, __m128i);

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

…

SHA256MSG1—Perform an Intermediate Calculation for the Next Four SHA256 Message Dwords

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 CC /r SHA256MSG1 xmm1, xmm2/m128	RM	V/V	SHA	Performs an intermediate calculation for the next four SHA256 message dwords using previous message dwords from xmm1 and xmm2/m128, storing the result in xmm1.

Instruction Operand Encoding

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

Description

The SHA256MSG1 instruction is one of two SHA256 message scheduling instructions. The instruction performs an intermediate calculation for the next four SHA256 message dwords.

Operation

SHA256MSG1

```
W4 ← SRC2[31: 0] ;
W3 ← SRC1[127:96] ;
W2 ← SRC1[95:64] ;
W1 ← SRC1[63: 32] ;
W0 ← SRC1[31: 0] ;

DEST[127:96] ← W3 + $\sigma_0$( W4);
DEST[95:64] ← W2 + $\sigma_0$( W3);
DEST[63:32] ← W1 + $\sigma_0$( W2);
DEST[31:0] ← W0 + $\sigma_0$( W1);
```

Intel C/C++ Compiler Intrinsic Equivalent

SHA256MSG1: __m128i _mm_sha256msg1_epu32(__m128i, __m128i);

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

...

SHA256MSG2—Perform a Final Calculation for the Next Four SHA256 Message Dwords

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 CD /r SHA256MSG2 xmm1, xmm2/m128	RM	V/V	SHA	Performs the final calculation for the next four SHA256 message dwords using previous message dwords from xmm1 and xmm2/m128, storing the result in xmm1.

Instruction Operand Encoding

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

Description

The SHA256MSG2 instruction is one of two SHA2 message scheduling instructions. The instruction performs the final calculation for the next four SHA256 message dwords.

Operation

SHA256MSG2

```
W14 ← SRC2[95:64] ;
W15 ← SRC2[127:96] ;
W16 ← SRC1[31: 0] + $\sigma_1$( W14) ;
W17 ← SRC1[63: 32] + $\sigma_1$( W15) ;
W18 ← SRC1[95: 64] + $\sigma_1$( W16) ;
W19 ← SRC1[127: 96] + $\sigma_1$( W17) ;

DEST[127:96] ← W19 ;
DEST[95:64] ← W18 ;
DEST[63:32] ← W17 ;
DEST[31:0] ← W16;
```

Intel C/C++ Compiler Intrinsic Equivalent

SHA256MSG2 : __m128i _mm_sha256msg2_epu32(__m128i, __m128i);

Flags Affected

None

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

...

SIDT—Store Interrupt Descriptor Table Register

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 01 /1	SIDT *m*	M	Valid	Valid	Store IDTR to *m.*

Instruction Operand Encoding

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

Description

Stores the content the interrupt descriptor table register (IDTR) in the destination operand. The destination operand specifies a 6-byte memory location.

In non-64-bit modes, if the operand-size attribute is 32 bits, the 16-bit limit field of the register is stored in the low 2 bytes of the memory location and the 32-bit base address is stored in the high 4 bytes. If the operand-size attribute is 16 bits, the limit is stored in the low 2 bytes and the 24-bit base address is stored in the third, fourth, and fifth byte, with the sixth byte filled with 0s.

In 64-bit mode, the operand size fixed at 8+2 bytes. The instruction stores 8-byte base and 2-byte limit values.

SIDT is only useful in operating-system software; however, it can be used in application programs without causing an exception to be generated if CR4.UMIP = 0. See "LGDT/LIDT—Load Global/Interrupt Descriptor Table Register" in Chapter 3, *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*, for information on loading the GDTR and IDTR.

IA-32 Architecture Compatibility

The 16-bit form of SIDT is compatible with the Intel 286 processor if the upper 8 bits are not referenced. The Intel 286 processor fills these bits with 1s; processor generations later than the Intel 286 processor fill these bits with 0s.

Operation

```
IF instruction is SIDT
   THEN
       IF OperandSize = 16
           THEN
               DEST[0:15] ← IDTR(Limit);
               DEST[16:39] ← IDTR(Base); (* 24 bits of base address stored; *)
               DEST[40:47] ← 0;
           ELSE IF (32-bit Operand Size)
               DEST[0:15] ← IDTR(Limit);
               DEST[16:47] ← IDTR(Base); FI; (* Full 32-bit base address stored *)
           ELSE (* 64-bit Operand Size *)
               DEST[0:15] ← IDTR(Limit);
               DEST[16:79] ← IDTR(Base); (* Full 64-bit base address stored *)
       FI;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.
	If CR4.UMIP = 1 and CPL > 0.
#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 CPL = 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If CR4.UMIP = 1.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#UD	If the destination operand is a register.
	If the LOCK prefix is used.
#GP(0)	If the memory address is in a non-canonical form.
	If CR4.UMIP = 1 and CPL > 0.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while CPL = 3.

...

SLDT—Store Local Descriptor Table Register

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 00 /0	SLDT *r/m16*	M	Valid	Valid	Stores segment selector from LDTR in *r/m16*.
REX.W + 0F 00 /0	SLDT *r64/m16*	M	Valid	Valid	Stores segment selector from LDTR in *r64/ m16*.

Instruction Operand Encoding

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

Description

Stores the segment selector from the local descriptor table register (LDTR) in the destination operand. The destination operand can be a general-purpose register or a memory location. The segment selector stored with this instruction points to the segment descriptor (located in the GDT) for the current LDT. This instruction can only be executed in protected mode.

Outside IA-32e mode, when the destination operand is a 32-bit register, the 16-bit segment selector is copied into the low-order 16 bits of the register. The high-order 16 bits of the register are cleared for the Pentium 4, Intel Xeon, and P6 family processors. They are undefined for Pentium, Intel486, and Intel386 processors. When the destination operand is a memory location, the segment selector is written to memory as a 16-bit quantity, regardless of the operand size.

In compatibility mode, when the destination operand is a 32-bit register, the 16-bit segment selector is copied into the low-order 16 bits of the register. The high-order 16 bits of the register are cleared. When the destination operand is a memory location, the segment selector is written to memory as a 16-bit quantity, regardless of the operand size.

In 64-bit mode, using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). The behavior of SLDT with a 64-bit register is to zero-extend the 16-bit selector and store it in the register. If the destination is memory and operand size is 64, SLDT will write the 16-bit selector to memory as a 16-bit quantity, regardless of the operand size.

Operation

```
DEST ← LDTR(SegmentSelector);
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0) If the destination is located in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.

If CR4.UMIP = 1 and CPL > 0.

#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 CPL = 3.

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

#UD The SLDT instruction is not recognized in virtual-8086 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 CR4.UMIP = 1 and CPL > 0.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while CPL = 3.

#UD If the LOCK prefix is used.

...

SMSW—Store Machine Status Word

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 01 /4	SMSW *r/m16*	M	Valid	Valid	Store machine status word to *r/m16*.
0F 01 /4	SMSW *r32/m16*	M	Valid	Valid	Store machine status word in low-order 16 bits of *r32/m16*; high-order 16 bits of *r32* are undefined.
REX.W + 0F 01 /4	SMSW *r64/m16*	M	Valid	Valid	Store machine status word in low-order 16 bits of *r64/m16*; high-order 16 bits of *r32* are undefined.

Instruction Operand Encoding

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

Description

Stores the machine status word (bits 0 through 15 of control register CR0) into the destination operand. The destination operand can be a general-purpose register or a memory location.

In non-64-bit modes, when the destination operand is a 32-bit register, the low-order 16 bits of register CR0 are copied into the low-order 16 bits of the register and the high-order 16 bits are undefined. When the destination operand is a memory location, the low-order 16 bits of register CR0 are written to memory as a 16-bit quantity, regardless of the operand size.

In 64-bit mode, the behavior of the SMSW instruction is defined by the following examples:

- SMSW r16 operand size 16, store CR0[15:0] in r16
- SMSW r32 operand size 32, zero-extend CR0[31:0], and store in r32
- SMSW r64 operand size 64, zero-extend CR0[63:0], and store in r64
- SMSW m16 operand size 16, store CR0[15:0] in m16
- SMSW m16 operand size 32, store CR0[15:0] in m16 (not m32)
- SMSW m16 operands size 64, store CR0[15:0] in m16 (not m64)

SMSW is only useful in operating-system software. However, it is not a privileged instruction and can be used in application programs if CR4.UMIP = 0. It is provided for compatibility with the Intel 286 processor. Programs and procedures intended to run on IA-32 and Intel 64 processors beginning with the Intel386 processors should use the MOV CR instruction to load the machine status word.

See “Changes to Instruction Behavior in VMX Non-Root Operation” in Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
DEST ← CR0[15:0];
(* Machine status word *)
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0) If the destination is located in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.

If CR4.UMIP = 1 and CPL > 0.

#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 CPL = 3.

#UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

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

#UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If CR4.UMIP = 1.

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

#UD If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0) If a memory address referencing the SS segment is in a non-canonical form.

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

If CR4.UMIP = 1 and CPL > 0.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while CPL = 3.

#UD If the LOCK prefix is used.

...

STR—Store Task Register

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 00 /1	STR *r/m16*	M	Valid	Valid	Stores segment selector from TR in *r/m16*.

Instruction Operand Encoding

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

Description

Stores the segment selector from the task register (TR) in the destination operand. The destination operand can be a general-purpose register or a memory location. The segment selector stored with this instruction points to the task state segment (TSS) for the currently running task.

When the destination operand is a 32-bit register, the 16-bit segment selector is copied into the lower 16 bits of the register and the upper 16 bits of the register are cleared. When the destination operand is a memory location, the segment selector is written to memory as a 16-bit quantity, regardless of operand size.

In 64-bit mode, operation is the same. The size of the memory operand is fixed at 16 bits. In register stores, the 2-byte TR is zero extended if stored to a 64-bit register.

The STR instruction is useful only in operating-system software. It can only be executed in protected mode.

Operation

```
DEST ← TR(SegmentSelector);
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0) If the destination is a memory operand that is located in a non-writable segment or if the 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.

If CR4.UMIP = 1 and CPL > 0.

#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 STR instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The STR instruction is not recognized in virtual-8086 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 CR4.UMIP = 1 and CPL > 0.
#SS(0)	If the stack 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.

...

VEXTRACTI128 — Extract packed Integer Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
VEX.256.66.0F3A.W0 39 /r ib VEXTRACTI128 *xmm1/m128, ymm2, imm8*	MRI	V/V	AVX2	Extract 128 bits of integer data from *ymm2* and store results in *xmm1/mem.*

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MRI	ModRM:r/m (w)	ModRM:reg (r)	Imm8	NA

Description

Extracts 128-bits of packed integer values from the source operand (second operand) at a 128-bit offset from imm8[0] into the destination operand (first operand). The destination may be either an XMM register or a 128-bit memory location.

VEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

The high 7 bits of the immediate are ignored.

An attempt to execute VEXTRACTI128 encoded with VEX.L= 0 will cause an #UD exception.

Operation

VEXTRACTI128 (memory destination form)

```
CASE (imm8[0]) OF
    0: DEST[127:0] ← SRC1[127:0]
    1: DEST[127:0] ← SRC1[255:128]
ESAC.
```

VEXTRACTI128 (register destination form)

```
CASE (imm8[0]) OF
    0: DEST[127:0] ← SRC1[127:0]
    1: DEST[127:0] ← SRC1[255:128]
ESAC.
DEST[VLMAX-1:128] ← 0
```

Intel C/C++ Compiler Intrinsic Equivalent

VEXTRACTI128: __m128i _mm256_extracti128_si256(__m256i a, int offset);

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 6; additionally

#UD	IF VEX.L = 0,
	If VEX.W = 1.

...

VPGATHERDQ/VPGATHERQQ — Gather Packed Qword Values Using Signed Dword/Qword Indices

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.128.66.0F38.W1 90 /r VPGATHERDQ *xmm1, vm32x, xmm2*	RMV	V/V	AVX2	Using dword indices specified in *vm32x*, gather qword values from memory conditioned on mask specified by *xmm2*. Conditionally gathered elements are merged into *xmm1*.
VEX.DDS.128.66.0F38.W1 91 /r VPGATHERQQ *xmm1, vm64x, xmm2*	RMV	V/V	AVX2	Using qword indices specified in *vm64x*, gather qword values from memory conditioned on mask specified by *xmm2*. Conditionally gathered elements are merged into *xmm1*.
VEX.DDS.256.66.0F38.W1 90 /r VPGATHERDQ *ymm1, vm32x, ymm2*	RMV	V/V	AVX2	Using dword indices specified in *vm32x*, gather qword values from memory conditioned on mask specified by *ymm2*. Conditionally gathered elements are merged into *ymm1*.
VEX.DDS.256.66.0F38.W1 91 /r VPGATHERQQ *ymm1, vm64y, ymm2*	RMV	V/V	AVX2	Using qword indices specified in *vm64y*, gather qword values from memory conditioned on mask specified by *ymm2*. Conditionally gathered elements are merged into *ymm1*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:reg (r,w)	BaseReg (R): VSIB:base, VectorReg(R): VSIB:index	VEX.vvvv (r, w)	NA

Description

The instruction conditionally loads up to 2 or 4 qword values from memory addresses specified by the memory operand (the second operand) and using qword indices. The memory operand uses the VSIB form of the SIB byte to specify a general purpose register operand as the common base, a vector register for an array of indices relative to the base and a constant scale factor.

The mask operand (the third operand) specifies the conditional load operation from each memory address and the corresponding update of each data element of the destination operand (the first operand). Conditionality is specified by the most significant bit of each data element of the mask register. If an element's mask bit is not set, the corresponding element of the destination register is left unchanged. The width of data element in the destination register and mask register are identical. The entire mask register will be set to zero by this instruction unless the instruction causes an exception.

Using dword indices in the lower half of the mask register, the instruction conditionally loads up to 2 or 4 qword values from the VSIB addressing memory operand, and updates the destination register.

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 operand 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 size and index size are different, part of the destination register and part of the mask register do not correspond to any elements being gathered. This instruction sets those parts to zero. It may do this 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.

VEX.128 version: The instruction will gather two qword values. For dword indices, only the lower two indices in the vector index register are used.

VEX.256 version: The instruction will gather four qword values. For dword indices, only the lower four indices in the vector index register are used.

Note that:

- If any pair of the index, mask, or destination registers are the same, this instruction results a UD fault.
- 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 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.
- This instruction will cause a #UD if the address size attribute is 16-bit.
- This instruction will cause a #UD if the memory operand is encoded without the SIB byte.
- This instruction should not be used to access memory mapped I/O as the ordering of the individual loads it does is implementation specific, and some implementations may use loads larger than the data element size or load elements an indeterminate number of times.
- 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

```
DEST ← SRC1;
BASE_ADDR: base register encoded in VSIB addressing;
VINDEX: the vector index register encoded by VSIB addressing;
SCALE: scale factor encoded by SIB:[7:6];
DISP: optional 1, 4 byte displacement;
MASK ← SRC3;
```

VPGATHERDQ (VEX.128 version)

```
FOR j← 0 to 1
    i ← j * 64;
    IF MASK[63+i] THEN
        MASK[i +63:i] ← FFFFFFFF_FFFFFFFFH; // extend from most significant bit
    ELSE
        MASK[i +63:i] ← 0;
    FI;
ENDFOR
FOR j← 0 to 1
    k ← j * 32;
    i ← j * 64;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX[k+31:k])*SCALE + DISP;
    IF MASK[63+i] THEN
        DEST[i +63:i] ← FETCH_64BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +63:i] ← 0;
ENDFOR
MASK[VLMAX-1:128] ← 0;
DEST[VLMAX-1:128] ← 0;
(non-masked elements of the mask register have the content of respective element  cleared)
```

VPGATHERQQ (VEX.128 version)

```
FOR j← 0 to 1
    i ← j * 64;
    IF MASK[63+i] THEN
        MASK[i +63:i] ← FFFFFFFF_FFFFFFFFH; // extend from most significant bit
    ELSE
        MASK[i +63:i] ← 0;
    FI;
ENDFOR
FOR j← 0 to 1
    i ←j * 64;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX1[i+63:i])*SCALE + DISP;
    IF MASK[63+i] THEN
        DEST[i +63:i] ← FETCH_64BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +63:i] ← 0;
ENDFOR
MASK[VLMAX-1:128] ← 0;
DEST[VLMAX-1:128] ← 0;
(non-masked elements of the mask register have the content of respective element  cleared)
```

VPGATHERQQ (VEX.256 version)

```
FOR j← 0 to 3
    i ← j * 64;
    IF MASK[63+i] THEN
         MASK[i +63:i] ← FFFFFFFF_FFFFFFFFH; // extend from most significant bit
    ELSE
         MASK[i +63:i] ← 0;
    FI;
ENDFOR
FOR j← 0 to 3
    i ← j * 64;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX1[i+63:i])*SCALE + DISP;
    IF MASK[63+i] THEN
         DEST[i +63:i] ← FETCH_64BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +63:i] ← 0;
ENDFOR
(non-masked elements of the mask register have the content of respective element  cleared)
```

VPGATHERDQ (VEX.256 version)

```
FOR j← 0 to 3
    i ← j * 64;
    IF MASK[63+i] THEN
         MASK[i +63:i] ← FFFFFFFF_FFFFFFFFH; // extend from most significant bit
    ELSE
         MASK[i +63:i] ← 0;
    FI;
ENDFOR
FOR j← 0 to 3
    k ← j * 32;
    i ← j * 64;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX1[k+31:k])*SCALE + DISP;
    IF MASK[63+i] THEN
         DEST[i +63:i] ← FETCH_64BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +63:i] ← 0;
ENDFOR
(non-masked elements of the mask register have the content of respective element  cleared)
```

Intel C/C++ Compiler Intrinsic Equivalent

VPGATHERDQ: __m128i _mm_i32gather_epi64 (__int64 const * base, __m128i index, const int scale);

VPGATHERDQ: __m128i _mm_mask_i32gather_epi64 (__m128i src, __int64 const * base, __m128i index, __m128i mask, const int scale);

VPGATHERDQ: __m256i _mm256_i32gather_epi64 (__int64 const * base, __m128i index, const int scale);

VPGATHERDQ: __m256i _mm256_mask_i32gather_epi64 (__m256i src, __int64 const * base, __m128i index, __m256i mask, const int scale);

VPGATHERQQ: __m128i _mm_i64gather_epi64 (__int64 const * base, __m128i index, const int scale);

VPGATHERQQ: __m128i _mm_mask_i64gather_epi64 (__m128i src, __int64 const * base, __m128i index, __m128i mask, const int scale);

VPGATHERQQ: __m256i _mm256_i64gather_epi64 __(int64 const * base, __m256i index, const int scale);

VPGATHERQQ: __m256i _mm256_mask_i64gather_epi64 (__m256i src, __int64 const * base, __m256i index, __m256i mask, const int scale);

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 12

...

WRPKRU—Write Data to User Page Key Register

Opcode*	Instruction	Op/ En	64/32bit Mode Support	CPUID Feature Flag	Description
0F 01 EF	WRPKRU	NP	V/V	OSPKE	Writes EAX into PKRU.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

Description

Writes the value of EAX into PKRU. ECX and EDX must be 0 when WRPKRU is executed; otherwise, a general-protection exception (#GP) occurs.

WRPKRU can be executed only if CR4.PKE = 1; otherwise, an invalid-opcode exception (#UD) occurs. Software can discover the value of CR4.PKE by examining CPUID.(EAX=07H,ECX=0H):ECX.OSPKE [bit 4].

On processors that support the Intel 64 Architecture, the high-order 32-bits of RCX, RDX and RAX are ignored.

Operation

```
IF (ECX = 0 AND EDX = 0)
    THEN PKRU ← EAX;
    ELSE #GP(0);
FI;
```

Flags Affected

None.

C/C++ Compiler Intrinsic Equivalent

WRPKRU:	void _wrpkru(uint32_t);

Protected Mode Exceptions

#GP(0)	If ECX ≠ 0.
	If EDX ≠ 0.
#UD	If the LOCK prefix is used.
	If CR4.PKE = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

XCHG—Exchange Register/Memory with Register

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
90+*rw*	XCHG AX, *r16*	O	Valid	Valid	Exchange *r16* with AX.
90+*rw*	XCHG *r16*, AX	O	Valid	Valid	Exchange AX with *r16.*
90+*rd*	XCHG EAX, *r32*	O	Valid	Valid	Exchange *r32* with EAX.
REX.W + 90+*rd*	XCHG RAX, *r64*	O	Valid	N.E.	Exchange *r64* with RAX.
90+*rd*	XCHG *r32*, EAX	O	Valid	Valid	Exchange EAX with *r32.*
REX.W + 90+*rd*	XCHG *r64*, RAX	O	Valid	N.E.	Exchange RAX with *r64.*
86 /*r*	XCHG *r/m8, r8*	MR	Valid	Valid	Exchange *r8* (byte register) with byte from *r/m8.*
REX + 86 /*r*	XCHG *r/m8**, r8**	MR	Valid	N.E.	Exchange *r8* (byte register) with byte from *r/m8.*
86 /*r*	XCHG *r8, r/m8*	RM	Valid	Valid	Exchange byte from *r/m8* with *r8* (byte register).
REX + 86 /*r*	XCHG *r8**, r/m8**	RM	Valid	N.E.	Exchange byte from *r/m8* with *r8* (byte register).
87 /*r*	XCHG *r/m16, r16*	MR	Valid	Valid	Exchange *r16* with word from *r/m16.*
87 /*r*	XCHG *r16, r/m16*	RM	Valid	Valid	Exchange word from *r/m16* with *r16.*
87 /*r*	XCHG *r/m32, r32*	MR	Valid	Valid	Exchange *r32* with doubleword from *r/m32.*
REX.W + 87 /*r*	XCHG *r/m64, r64*	MR	Valid	N.E.	Exchange *r64* with quadword from *r/m64.*
87 /*r*	XCHG *r32, r/m32*	RM	Valid	Valid	Exchange doubleword from *r/m32* with *r32.*
REX.W + 87 /*r*	XCHG *r64, r/m64*	RM	Valid	N.E.	Exchange quadword from *r/m64* with *r64.*

NOTES:

* In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
O	AX/EAX/RAX (r, w)	opcode + rd (r, w)	NA	NA
O	opcode + rd (r, w)	AX/EAX/RAX (r, w)	NA	NA
MR	ModRM:r/m (r, w)	ModRM:reg (r)	NA	NA
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

Description

Exchanges the contents of the destination (first) and source (second) operands. The operands can be two general-purpose registers or a register and a memory location. If a memory operand is referenced, the processor's locking protocol is automatically implemented for the duration of the exchange operation, regardless of the presence or absence of the LOCK prefix or of the value of the IOPL. (See the LOCK prefix description in this chapter for more information on the locking protocol.)

This instruction is useful for implementing semaphores or similar data structures for process synchronization. (See "Bus Locking" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3A*, for more information on bus locking.)

The XCHG instruction can also be used instead of the BSWAP instruction for 16-bit operands.

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

NOTE

XCHG (E)AX, (E)AX (encoded instruction byte is 90H) is an alias for NOP regardless of data size prefixes, including REX.W.

Operation

```
TEMP ← DEST;
DEST ← SRC;
SRC ← TEMP;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If either operand is in a non-writable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

...

8. Updates to Appendix A, Volume 2C

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

...

Table A-4. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 38H) *

	pfx	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	vpsignb Vx, Hx, Wx	vpsignw Vx, Hx, Wx	vpsignd Vx, Hx, Wx	vpmulhrsw Vx, Hx, Wx	vpermilps^{V} Vx,Hx,Wx	vpermilpd^{V} Vx,Hx,Wx	vtestps^{V} Vx, Wx	vtestpd^{V} Vx, Wx
1						pabsb Pq, Qq	pabsw Pq, Qq	pabsd Pq, Qq	
	66	vbroadcastss^{V} Vx, Wd	vbroadcastsd^{V} Vqq, Wq	vbroadcastf128^{V} Vqq, Mdq		vpabsb Vx, Wx	vpabsw Vx, Wx	vpabsd Vx, Wx	
2	66	vpmuldq Vx, Hx, Wx	vpcmpeqq Vx, Hx, Wx	vmovntdqa Vx, Mx	vpackusdw Vx, Hx, Wx	vmaskmovps^{V} Vx,Hx,Mx	vmaskmovpd^{V} Vx,Hx,Mx	vmaskmovps^{V} Mx,Hx,Vx	vmaskmovpd^{V} Mx,Hx,Vx
3	66	vpminsb Vx, Hx, Wx	vpminsd Vx, Hx, Wx	vpminuw Vx, Hx, Wx	vpminud Vx, Hx, Wx	vpmaxsb Vx, Hx, Wx	vpmaxsd Vx, Hx, Wx	vpmaxuw Vx, Hx, Wx	vpmaxud Vx, Hx, Wx
4									
5	66	vpbroadcastd^{V} Vx, Wx	vpbroadcastq^{V} Vx, Wx	vbroadcasti128^{V} Vqq, Mdq					
6									
7	66	vpbroadcastb^{V} Vx, Wx	vpbroadcastw^{V} Vx, Wx						
8	66					vpmaskmovd/q^{V} Vx,Hx,Mx		vpmaskmovd/q^{V} Mx,Vx,Hx	
9	66	vfmadd132ps/d^{V} Vx, Hx, Wx	vfmadd132ss/d^{V} Vx, Hx, Wx	vfmsub132ps/d^{V} Vx, Hx, Wx	vfmsub132ss/d^{V} Vx, Hx, Wx	vfnmadd132ps/d^{V} Vx, Hx, Wx	vfnmadd132ss/d^{V} Vx, Hx, Wx	vfnmsub132ps/d^{V} Vx, Hx, Wx	vfnmsub132ss/d^{V} Vx, Hx, Wx
A	66	vfmadd213ps/d^{V} Vx, Hx, Wx	vfmadd213ss/d^{V} Vx, Hx, Wx	vfmsub213ps/d^{V} Vx, Hx, Wx	vfmsub213ss/d^{V} Vx, Hx, Wx	vfnmadd213ps/d^{V} Vx, Hx, Wx	vfnmadd213ss/d^{V} Vx, Hx, Wx	vfnmsub213ps/d^{V} Vx, Hx, Wx	vfnmsub213ss/d^{V} Vx, Hx, Wx
B	66	vfmadd231ps/d^{V} Vx, Hx, Wx	vfmadd231ss/d^{V} Vx, Hx, Wx	vfmsub231ps/d^{V} Vx, Hx, Wx	vfmsub231ss/d^{V} Vx, Hx, Wx	vfnmadd231ps/d^{V} Vx, Hx, Wx	vfnmadd231ss/d^{V} Vx, Hx, Wx	vfnmsub231ps/d^{V} Vx, Hx, Wx	vfnmsub231ss/d^{V} Vx, Hx, Wx
C		sha1nexte Vdq,Wdq	sha1msg1 Vdq,Wdq	sha1msg2 Vdq,Wdq	sha256rnds2 Vdq,Wdq	sha256msg1 Vdq,Wdq	sha256msg2 Vdq,Wdq		
	66								
D	66				VAESIMC Vdq, Wdq	VAESENC Vdq,Hdq,Wdq	VAESENCLAST Vdq,Hdq,Wdq	VAESDEC Vdq,Hdq,Wdq	VAESDECLAST Vdq,Hdq,Wdq
E									
F									
	66								
	F3								
	F2								
	66 & F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

...

Table A-4. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 38H) *

	pfx	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	vpsignb Vx, Hx, Wx	vpsignw Vx, Hx, Wx	vpsignd Vx, Hx, Wx	vpmulhrsw Vx, Hx, Wx	vpermilps^{V} Vx,Hx,Wx	vpermilpd^{V} Vx,Hx,Wx	vtestps^{V} Vx, Wx	vtestpd^{V} Vx, Wx
1						pabsb Pq, Qq	pabsw Pq, Qq	pabsd Pq, Qq	
	66	vbroadcastss^{V} Vx, Wd	vbroadcastsd^{V} Vqq, Wq	vbroadcastf128^{V} Vqq, Mdq		vpabsb Vx, Wx	vpabsw Vx, Wx	vpabsd Vx, Wx	
2	66	vpmuldq Vx, Hx, Wx	vpcmpeqq Vx, Hx, Wx	vmovntdqa Vx, Mx	vpackusdw Vx, Hx, Wx	vmaskmovps^{V} Vx,Hx,Mx	vmaskmovpd^{V} Vx,Hx,Mx	vmaskmovps^{V} Mx,Hx,Vx	vmaskmovpd^{V} Mx,Hx,Vx
3	66	vpminsb Vx, Hx, Wx	vpminsd Vx, Hx, Wx	vpminuw Vx, Hx, Wx	vpminud Vx, Hx, Wx	vpmaxsb Vx, Hx, Wx	vpmaxsd Vx, Hx, Wx	vpmaxuw Vx, Hx, Wx	vpmaxud Vx, Hx, Wx
4									
5	66	vpbroadcastd^{V} Vx, Wx	vpbroadcastq^{V} Vx, Wx	vbroadcasti128^{V} Vqq, Mdq					
6									
7	66	vpbroadcastb^{V} Vx, Wx	vpbroadcastw^{V} Vx, Wx						
8	66					vpmaskmovd/q^{V} Vx,Hx,Mx		vpmaskmovd/q^{V} Mx,Vx,Hx	
9	66	vfmadd132ps/d^{V} Vx, Hx, Wx	vfmadd132ss/d^{V} Vx, Hx, Wx	vfmsub132ps/d^{V} Vx, Hx, Wx	vfmsub132ss/d^{V} Vx, Hx, Wx	vfnmadd132ps/d^{V} Vx, Hx, Wx	vfnmadd132ss/d^{V} Vx, Hx, Wx	vfnmsub132ps/d^{V} Vx, Hx, Wx	vfnmsub132ss/d^{V} Vx, Hx, Wx
A	66	vfmadd213ps/d^{V} Vx, Hx, Wx	vfmadd213ss/d^{V} Vx, Hx, Wx	vfmsub213ps/d^{V} Vx, Hx, Wx	vfmsub213ss/d^{V} Vx, Hx, Wx	vfnmadd213ps/d^{V} Vx, Hx, Wx	vfnmadd213ss/d^{V} Vx, Hx, Wx	vfnmsub213ps/d^{V} Vx, Hx, Wx	vfnmsub213ss/d^{V} Vx, Hx, Wx
B	66	vfmadd231ps/d^{V} Vx, Hx, Wx	vfmadd231ss/d^{V} Vx, Hx, Wx	vfmsub231ps/d^{V} Vx, Hx, Wx	vfmsub231ss/d^{V} Vx, Hx, Wx	vfnmadd231ps/d^{V} Vx, Hx, Wx	vfnmadd231ss/d^{V} Vx, Hx, Wx	vfnmsub231ps/d^{V} Vx, Hx, Wx	vfnmsub231ss/d^{V} Vx, Hx, Wx
C		sha1nexte Vdq,Wdq	sha1msg1 Vdq,Wdq	sha1msg2 Vdq,Wdq	sha256rnds2 Vdq,Wdq	sha256msg1 Vdq,Wdq	sha256msg2 Vdq,Wdq		
	66								
D	66				VAESIMC Vdq, Wdq	VAESENC Vdq,Hdq,Wdq	VAESENCLAST Vdq,Hdq,Wdq	VAESDEC Vdq,Hdq,Wdq	VAESDECLAST Vdq,Hdq,Wdq
E									
F									
	66								
	F3								
	F2								
	66 & F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

...

Table A-5. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 3AH) *

	pfx	8	9	A	B	C	D	E	F
0									palignr Pq, Qq, Ib
	66	vroundps Vx,Wx,Ib	vroundpd Vx,Wx,Ib	vroundss Vss,Wss,Ib	vroundsd Vsd,Wsd,Ib	vblendps Vx,Hx,Wx,Ib	vblendpd Vx,Hx,Wx,Ib	vpblendw Vx,Hx,Wx,Ib	vpalignr Vx,Hx,Wx,Ib
1	66	vinsertf128^{v} Vqq,Hqq,Wqq,Ib	vextractf128^{v} Wdq,Vqq,Ib				vcvtps2ph^{v} Wx, Vx, Ib		
2									
3	66	vinserti128^{v} Vqq,Hqq,Wqq,Ib	vextracti128^{v} Wdq,Vqq,Ib						
4	66			vblendvps^{v} Vx,Hx,Wx,Lx	vblendvpd^{v} Vx,Hx,Wx,Lx	vpblendvb^{v} Vx,Hx,Wx,Lx			
5									
6									
7									
8									
9									
A									
B									
C						sha1rnds4 Vdq,Wdq,Ib			
D	66								VAESKEYGEN Vdq, Wdq, Ib
E									
F									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

...

A.4.2 Opcode Extension Tables

See Table A-6 below.

Table A-6 Opcode Extensions for One- and Two-byte Opcodes by Group Number *

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
80-83	1	mem, 11B		ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
8F	1A	mem, 11B		POP							
C0,C1 reg, imm D0, D1 reg, 1 D2, D3 reg, CL	2	mem, 11B		ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
F6, F7	3	mem, 11B		TEST Ib/Iz		NOT	NEG	MUL AL/rAX	IMUL AL/rAX	DIV AL/rAX	IDIV AL/rAX
FE	4	mem, 11B		INC Eb	DEC Eb						
FF	5	mem, 11B		INC Ev	DEC Ev	near CALL^{f64} Ev	far CALL Ep	near JMP^{f64} Ev	far JMP Mp	PUSH^{d64} Ev	
0F 00	6	mem, 11B		SLDT Rv/Mw	STR Rv/Mw	LLDT Ew	LTR Ew	VERR Ew	VERW Ew		
0F 01	7	mem		SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Mw/Rv		LMSW Ew	INVLPG Mb
		11B		VMCALL (001) VMLAUNCH (010) VMRESUME (011) VMXOFF (100)	MONITOR (000) MWAIT (001) CLAC (010) STAC (011) ENCLS (111)	XGETBV (000) XSETBV (001) VMFUNC (100) XEND (101) XTEST (110) ENCLU(111)					SWAPGS ^{o64}(000) RDTSCP (001)
0F BA	8	mem, 11B						BT	BTS	BTR	BTC
0F C7	9	mem			CMPXCH8B Mq CMPXCHG16B Mdq					VMPTRLD Mq	VMPTRST Mq
			66							VMCLEAR Mq	
			F3							VMXON Mq	
		11B								RDRAND Rv	RDSEED Rv
			F3								RDPID Rv
0F B9	10	mem									
		11B									
C6	11	mem		MOV Eb, Ib							
		11B									XABORT (000) Ib
C7		mem		MOV Ev, Iz							
		11B									XBEGIN (000) Jz
0F 71	12	mem									
		11B				psrlw Nq, Ib		psraw Nq, Ib		psllw Nq, Ib	
			66			vpsrlw Hx,Ux,Ib		vpsraw Hx,Ux,Ib		vpsllw Hx,Ux,Ib	
0F 72	13	mem									
		11B				psrld Nq, Ib		psrad Nq, Ib		pslld Nq, Ib	
			66			vpsrld Hx,Ux,Ib		vpsrad Hx,Ux,Ib		vpslld Hx,Ux,Ib	
0F 73	14	mem									
		11B				psrlq Nq, Ib				psllq Nq, Ib	
			66			vpsrlq Hx,Ux,Ib	vpsrldq Hx,Ux,Ib			vpsllq Hx,Ux,Ib	vpslldq Hx,Ux,Ib

Table A-6 Opcode Extensions for One- and Two-byte Opcodes by Group Number * (Contd.)

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
0F AE	15	mem		fxsave	fxrstor	ldmxcsr	stmxcsr	XSAVE	XRSTOR	XSAVEOPT	clflush
		11B							lfence	mfence	sfence
			F3	RDFSBASE Ry	RDGSBASE Ry	WRFSBASE Ry	WRGSBASE Ry				
0F 18	16	mem		prefetch NTA	prefetch T0	prefetch T1	prefetch T2				
		11B									
VEX.0F38 F3	17	mem			BLSR^{v} By, Ey	BLSMSK^{v} By, Ey	BLSI^{v} By, Ey				
		11B									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

...

9. Updates to Chapter 1, Volume 3A

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

...

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series

- Intel® Core™2 Extreme QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Core™2 Extreme QX9000 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes.
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families
- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel® Atom™ processor X7-Z8000 and X5-Z8000 series
- Intel® Atom™ processor Z3400 series
- Intel® Atom™ processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel® Atom™ processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Intel® microarchitecture code name Nehalem. Intel® microarchitecture code name Westmere is a 32 nm version of Intel® microarchitecture code name Nehalem. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on Intel® microarchitecture code name Westmere. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Intel® microarchitecture code name Sandy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Intel® microarchitecture code name Ivy Bridge and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Intel® microarchitecture code name Ivy Bridge-E and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Intel® microarchitecture code name Haswell and support Intel 64 architecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Intel® microarchitecture code name Broadwell and support Intel 64 architecture.

The Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Intel® microarchitecture code name Skylake and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Intel® microarchitecture code name Haswell-E and support Intel 64 architecture.

The Intel® Atom™ processor Z8000 series is based on the Intel microarchitecture code name Airmont.

The Intel® Atom™ processor Z3400 series and the Intel® Atom™ processor Z3500 series are based on the Intel microarchitecture code name Silvermont.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme processors, Intel Core 2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is a superset of and compatible with IA-32 architecture.

...

10. Updates to Chapter 2, Volume 3A

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-7) determine operating mode of the processor and the characteristics of the currently executing task. These registers are 32 bits in all 32-bit modes and compatibility mode.

In 64-bit mode, control registers are expanded to 64 bits. The MOV CRn instructions are used to manipulate the register bits. Operand-size prefixes for these instructions are ignored. The following is also true:

- The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.
- Bits 63:32 of CR0 and CR4 are reserved and must be written with zeros. Writing a nonzero value to any of the upper 32 bits results in a general-protection exception, #GP(0).
- All 64 bits of CR2 are writable by software.
- Bits 51:40 of CR3 are reserved and must be 0.
- The MOV CRn instructions do not check that addresses written to CR2 and CR3 are within the linear-address or physical-address limitations of the implementation.
- Register CR8 is available in 64-bit mode only.

The control registers are summarized below, and each architecturally defined control field in these control registers is described individually. In Figure 2-7, the width of the register in 64-bit mode is indicated in parenthesis (except for CR0).

- **CR0** — Contains system control flags that control operating mode and states of the processor.
- **CR1** — Reserved.
- **CR2** — Contains the page-fault linear address (the linear address that caused a page fault).
- **CR3** — Contains the physical address of the base of the paging-structure hierarchy and two flags (PCD and PWT). Only the most-significant bits (less the lower 12 bits) of the base address are specified; the lower 12 bits of the address are assumed to be 0. The first paging structure must thus be aligned to a page (4-KByte) boundary. The PCD and PWT flags control caching of that paging structure in the processor's internal data caches (they do not control TLB caching of page-directory information).

  When using the physical address extension, the CR3 register contains the base address of the page-directory-pointer table. In IA-32e mode, the CR3 register contains the base address of the PML4 table.

  See also: Chapter 4, "Paging."
- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities.

- **CR8** — Provides read and write access to the Task Priority Register (TPR). It specifies the priority threshold value that operating systems use to control the priority class of external interrupts allowed to interrupt the processor. This register is available only in 64-bit mode. However, interrupt filtering continues to apply in compatibility mode.

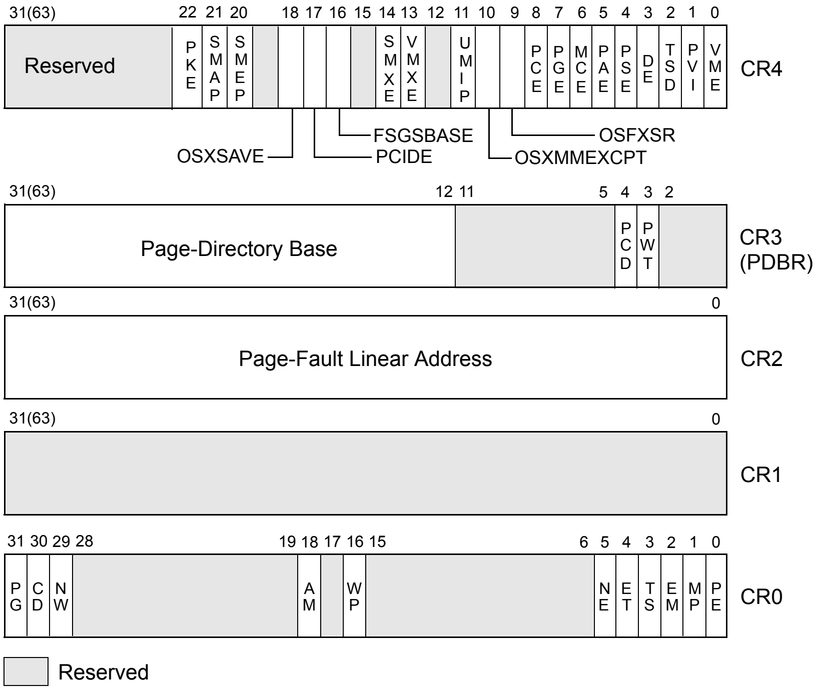


Figure 2-7 Control Registers

When loading a control register, reserved bits should always be set to the values previously read. The flags in control registers are:

PG **Paging (bit 31 of CR0)** — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 4, "Paging."

On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.

CD **Cache Disable (bit 30 of CR0)** — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor's internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 11-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 11.5.3, "Preventing Caching," and Section 11.5, "Cache Control."

NW **Not Write-through (bit 29 of CR0)** — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 11-5 for detailed information about the effect of the NW flag on caching for other settings of the CD and NW flags.

AM **Alignment Mask (bit 18 of CR0)** — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

WP **Write Protect (bit 16 of CR0)** — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting; see Section 4.1.3 and Section 4.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX.

NE **Numeric Error (bit 5 of CR0)** — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.

See also: Section 8.7, "Handling x87 FPU Exceptions in Software" in Chapter 8, "Programming with the x87 FPU," and Appendix A, "EFLAGS Cross-Reference," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

ET **Extension Type (bit 4 of CR0)** — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

TS **Task Switched (bit 3 of CR0)** — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

- If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.
- If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.
- If the EM flag is set, the setting of the TS flag has no effect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

Table 2-2 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 12-1 and 13-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.

The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).

The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction, the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context is never saved.

Table 2-2 Action Taken By x87 FPU Instructions for Different Combinations of EM, MP, and TS

CR0 Flags			x87 FPU Instruction Type	
EM	MP	TS	Floating-Point	WAIT/FWAIT
0	0	0	Execute	Execute.
0	0	1	#NM Exception	Execute.
0	1	0	Execute	Execute.
0	1	1	#NM Exception	#NM exception.
1	0	0	#NM Exception	Execute.
1	0	1	#NM Exception	Execute.
1	1	0	#NM Exception	Execute.
1	1	1	#NM Exception	#NM exception.

EM **Emulation (bit 2 of CR0)** — Indicates that the processor does not have an internal or external x87 FPU when set; indicates an x87 FPU is present when clear. This flag also affects the execution of MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

When the EM flag is set, execution of an x87 FPU instruction generates a device-not-available exception (#NM). This flag must be set when the processor does not have an internal x87 FPU or is not connected to an external math coprocessor. Setting this flag forces all floating-point instructions to be handled by software emulation. Table 9-2 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the EM, MP, and TS flags.

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 12-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see Table 13-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

MP Monitor Coprocessor (bit 1 of CR0) — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CR0). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 9-2 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the MP, EM, and TS flags.

PE **Protection Enable (bit 0 of CR0)** — Enables protected mode when set; enables real-address mode when clear. This flag does not enable paging directly. It only enables segment-level protection. To enable paging, both the PE and PG flags must be set.

See also: Section 9.9, "Mode Switching."

PCD **Page-level Cache Disable (bit 4 of CR3)** — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, "Paging and Memory Typing". This bit is not used if paging is disabled, with PAE paging, or with IA-32e paging if CR4.PCIDE=1.

PWT **Page-level Write-Through (bit 3 of CR3)** — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, "Paging and Memory Typing". This bit is not used if paging is disabled, with PAE paging, or with IA-32e paging if CR4.PCIDE=1.

VME **Virtual-8086 Mode Extensions (bit 0 of CR4)** — Enables interrupt- and exception-handling extensions in virtual-8086 mode when set; disables the extensions when clear. Use of the virtual mode extensions can improve the performance of virtual-8086 applications by eliminating the overhead of calling the virtual-8086 monitor to handle interrupts and exceptions that occur while executing an 8086 program and, instead, redirecting the interrupts and exceptions back to the 8086 program's handlers. It also provides hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multitasking and multiple-processor environments.

See also: Section 20.3, "Interrupt and Exception Handling in Virtual-8086 Mode."

PVI **Protected-Mode Virtual Interrupts (bit 1 of CR4)** — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.

See also: Section 20.4, "Protected-Mode Virtual Interrupts."

TSD **Time Stamp Disable (bit 2 of CR4)** — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported (if CPUID.80000001H:EDX[27] = 1).

DE **Debugging Extensions (bit 3 of CR4)** — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.

See also: Section 17.2.2, "Debug Registers DR4 and DR5."

PSE **Page Size Extensions (bit 4 of CR4)** — Enables 4-MByte pages with 32-bit paging when set; restricts 32-bit paging to pages of 4 KBytes when clear.

See also: Section 4.3, "32-Bit Paging."

PAE **Physical Address Extension (bit 5 of CR4)** — When set, enables paging to produce physical addresses with more than 32 bits. When clear, restricts physical addresses to 32 bits. PAE must be set before entering IA-32e mode.

See also: Chapter 4, "Paging."

MCE **Machine-Check Enable (bit 6 of CR4)** — Enables the machine-check exception when set; disables the machine-check exception when clear.

See also: Chapter 15, "Machine-Check Architecture."

PGE **Page Global Enable (bit 7 of CR4)** — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory or page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.

When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.

See also: Section 4.10, "Caching Translation Information."

PCE **Performance-Monitoring Counter Enable (bit 8 of CR4)** — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.

OSFXSR

Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4) — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX registers, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID feature flag FXSR indicates availability of the FXSAVE/FXRSTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRSTOR to save/restore the contents of the X87 FPU, XMM and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

OSXMMEXCPT

Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4) — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (#XM) is generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

UMIP

User-Mode Instruction Prevention (bit 11 of CR4) — When set, the following instructions cannot be executed if CPL > 0: SGDT, SIDT, SLDT, SMSW, and STR. An attempt at such execution causes a general-protection exception (#GP).

VMXE

VMX-Enable Bit (bit 13 of CR4) — Enables VMX operation when set. See Chapter 23, "Introduction to Virtual-Machine Extensions."

SMXE

SMX-Enable Bit (bit 14 of CR4) — Enables SMX operation when set. See Chapter 5, "Safer Mode Extensions Reference" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C*.

FSGSBASE

FSGSBASE-Enable Bit (bit 16 of CR4) — Enables the instructions RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE.

PCIDE

PCID-Enable Bit (bit 17 of CR4) — Enables process-context identifiers (PCIDs) when set. See Section 4.10.1, "Process-Context Identifiers (PCIDs)". Can be set only in IA-32e mode (if IA32_EFER.LMA = 1).

OSXSAVE

XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4) — When set, this flag: (1) indicates (via CPUID.01H:ECX.OSXSAVE[bit 27]) that the operating system supports the use of the XGETBV, XSAVE and XRSTOR instructions by general software; (2) enables the XSAVE and XRSTOR instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in XCR0; (3) enables the processor to execute XGETBV and XSETBV instructions in order to read and write XCR0. See Section 2.6 and Chapter 13, "System Programming for Instruction Set Extensions and Processor Extended States".

SMEP

SMEP-Enable Bit (bit 20 of CR4) — Enables supervisor-mode execution prevention (SMEP) when set. See Section 4.6, "Access Rights".

SMAP

SMAP-Enable Bit (bit 21 of CR4) — Enables supervisor-mode access prevention (SMAP) when set. See Section 4.6, "Access Rights."

PKE

Protection-Key-Enable Bit (bit 22 of CR4) — Enables IA-32e paging to associate each linear address with a protection key. The PKRU register specifies, for each protection key, whether user-mode linear addresses with that protection key can be read or written. This bit also enables access to the PKRU register using the RDPKRU and WRPKRU instructions.

TPL

Task Priority Level (bit 3:0 of CR8) — This sets the threshold value corresponding to the highest-priority interrupt to be blocked. A value of 0 means all interrupts are enabled. This field is available in 64-bit mode. A value of 15 means all interrupts will be disabled.

...

2.8 SYSTEM INSTRUCTION SUMMARY

System instructions handle system-level functions such as loading system registers, managing the cache, managing interrupts, or setting up the debug registers. Many of these instructions can be executed only by operating-system or executive procedures (that is, procedures running at privilege level 0). Others can be executed at any privilege level and are thus available to application programs.

Table 2-3 lists the system instructions and indicates whether they are available and useful for application programs. These instructions are described in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B & 2C*.

Table 2-3 Summary of System Instructions

Instruction	Description	Useful to Application?	Protected from Application?
LLDT	Load LDT Register	No	Yes
SLDT	Store LDT Register	No	If CR4.UMIP = 1
LGDT	Load GDT Register	No	Yes
SGDT	Store GDT Register	No	If CR4.UMIP = 1
LTR	Load Task Register	No	Yes
STR	Store Task Register	No	If CR4.UMIP = 1
LIDT	Load IDT Register	No	Yes
SIDT	Store IDT Register	No	If CR4.UMIP = 1
MOV CR*n*	Load and store control registers	No	Yes
SMSW	Store MSW	Yes	If CR4.UMIP = 1
LMSW	Load MSW	No	Yes
CLTS	Clear TS flag in CR0	No	Yes
ARPL	Adjust RPL	Yes[1, 5]	No
LAR	Load Access Rights	Yes	No
LSL	Load Segment Limit	Yes	No
VERR	Verify for Reading	Yes	No

Table 2-3 Summary of System Instructions (Contd.)

Instruction	Description	Useful to Application?	Protected from Application?
VERW	Verify for Writing	Yes	No
MOV DR*n*	Load and store debug registers	No	Yes
INVD	Invalidate cache, no writeback	No	Yes
WBINVD	Invalidate cache, with writeback	No	Yes
INVLPG	Invalidate TLB entry	No	Yes
HLT	Halt Processor	No	Yes
LOCK (Prefix)	Bus Lock	Yes	No
RSM	Return from system management mode	No	Yes
RDMSR[3]	Read Model-Specific Registers	No	Yes
WRMSR[3]	Write Model-Specific Registers	No	Yes
RDPMC[4]	Read Performance-Monitoring Counter	Yes	Yes[2]
RDTSC[3]	Read Time-Stamp Counter	Yes	Yes[2]
RDTSCP[7]	Read Serialized Time-Stamp Counter	Yes	Yes[2]
XGETBV	Return the state of XCR0	Yes	No
XSETBV	Enable one or more processor extended states	No[6]	Yes

NOTES:

1. Useful to application programs running at a CPL of 1 or 2.
2. The TSD and PCE flags in control register CR4 control access to these instructions by application programs running at a CPL of 3.
3. These instructions were introduced into the IA-32 Architecture with the Pentium processor.
4. This instruction was introduced into the IA-32 Architecture with the Pentium Pro processor and the Pentium processor with MMX technology.
5. This instruction is not supported in 64-bit mode.
6. Application uses XGETBV to query which set of processor extended states are enabled.
7. RDTSCP is introduced in Intel Core i7 processor.

2.8.1 Loading and Storing System Registers

The GDTR, LDTR, IDTR, and TR registers each have a load and store instruction for loading data into and storing data from the register:

- **LGDT (Load GDTR Register)** — Loads the GDT base address and limit from memory into the GDTR register.
- **SGDT (Store GDTR Register)** — Stores the GDT base address and limit from the GDTR register into memory.
- **LIDT (Load IDTR Register)** — Loads the IDT base address and limit from memory into the IDTR register.
- **SIDT (Store IDTR Register)** — Stores the IDT base address and limit from the IDTR register into memory.
- **LLDT (Load LDTR Register)** — Loads the LDT segment selector and segment descriptor from memory into the LDTR. (The segment selector operand can also be located in a general-purpose register.)
- **SLDT (Store LDTR Register)** — Stores the LDT segment selector from the LDTR register into memory or a general-purpose register.
- **LTR (Load Task Register)** — Loads segment selector and segment descriptor for a TSS from memory into the task register. (The segment selector operand can also be located in a general-purpose register.)

- **STR (Store Task Register)** — Stores the segment selector for the current task TSS from the task register into memory or a general-purpose register.

The LMSW (load machine status word) and SMSW (store machine status word) instructions operate on bits 0 through 15 of control register CR0. These instructions are provided for compatibility with the 16-bit Intel 286 processor. Programs written to run on 32-bit IA-32 processors should not use these instructions. Instead, they should access the control register CR0 using the MOV CR instruction.

The CLTS (clear TS flag in CR0) instruction is provided for use in handling a device-not-available exception (#NM) that occurs when the processor attempts to execute a floating-point instruction when the TS flag is set. This instruction allows the TS flag to be cleared after the x87 FPU context has been saved, preventing further #NM exceptions. See Section 2.5, "Control Registers," for more information on the TS flag.

The control registers (CR0, CR1, CR2, CR3, CR4, and CR8) are loaded using the MOV instruction. The instruction loads a control register from a general-purpose register or stores the content of a control register in a general-purpose register.

...

11. Updates to Chapter 6, Volume 3A

Change bars show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

Interrupt 13—General Protection Exception (#GP)

Exception Class Fault.

Description

Indicates that the processor detected one of a class of protection violations called "general-protection violations." The conditions that cause this exception to be generated comprise all the protection violations that do not cause other exceptions to be generated (such as, invalid-TSS, segment-not-present, stack-fault, or page-fault exceptions). The following conditions cause general-protection exceptions to be generated:

- Exceeding the segment limit when accessing the CS, DS, ES, FS, or GS segments.
- Exceeding the segment limit when referencing a descriptor table (except during a task switch or a stack switch).
- Transferring execution to a segment that is not executable.
- Writing to a code segment or a read-only data segment.
- Reading from an execute-only code segment.
- Loading the SS register with a segment selector for a read-only segment (unless the selector comes from a TSS during a task switch, in which case an invalid-TSS exception occurs).
- Loading the SS, DS, ES, FS, or GS register with a segment selector for a system segment.
- Loading the DS, ES, FS, or GS register with a segment selector for an execute-only code segment.
- Loading the SS register with the segment selector of an executable segment or a null segment selector.
- Loading the CS register with a segment selector for a data segment or a null segment selector.
- Accessing memory using the DS, ES, FS, or GS register when it contains a null segment selector.
- Switching to a busy task during a call or jump to a TSS.

- Using a segment selector on a non-IRET task switch that points to a TSS descriptor in the current LDT. TSS descriptors can only reside in the GDT. This condition causes a #TS exception during an IRET task switch.
- Violating any of the privilege rules described in Chapter 5, “Protection.”
- Exceeding the instruction length limit of 15 bytes (this only can occur when redundant prefixes are placed before an instruction).
- Loading the CR0 register with a set PG flag (paging enabled) and a clear PE flag (protection disabled).
- Loading the CR0 register with a set NW flag and a clear CD flag.
- Referencing an entry in the IDT (following an interrupt or exception) that is not an interrupt, trap, or task gate.
- Attempting to access an interrupt or exception handler through an interrupt or trap gate from virtual-8086 mode when the handler’s code segment DPL is greater than 0.
- Attempting to write a 1 into a reserved bit of CR4.
- Attempting to execute a privileged instruction when the CPL is not equal to 0 (see Section 5.9, “Privileged Instructions,” for a list of privileged instructions).
- Attempting to execute SGDT, SIDT, SLDT, SMSW, or STR when CR4.UMIP = 1 and the CPL is not equal to 0.
- Writing to a reserved bit in an MSR.
- Accessing a gate that contains a null segment selector.
- Executing the INT *n* instruction when the CPL is greater than the DPL of the referenced interrupt, trap, or task gate.
- The segment selector in a call, interrupt, or trap gate does not point to a code segment.
- The segment selector operand in the LLDT instruction is a local type (TI flag is set) or does not point to a segment descriptor of the LDT type.
- The segment selector operand in the LTR instruction is local or points to a TSS that is not available.
- The target code-segment selector for a call, jump, or return is null.
- If the PAE and/or PSE flag in control register CR4 is set and the processor detects any reserved bits in a page-directory-pointer-table entry set to 1. These bits are checked during a write to control registers CR0, CR3, or CR4 that causes a reloading of the page-directory-pointer-table entry.
- Attempting to write a non-zero value into the reserved bits of the MXCSR register.
- Executing an SSE/SSE2/SSE3 instruction that attempts to access a 128-bit memory location that is not aligned on a 16-byte boundary when the instruction requires 16-byte alignment. This condition also applies to the stack segment.

A program or task can be restarted following any general-protection exception. If the exception occurs while attempting to call an interrupt handler, the interrupted program can be restartable, but the interrupt may be lost.

Exception Error Code

The processor pushes an error code onto the exception handler's stack. If the fault condition was detected while loading a segment descriptor, the error code contains a segment selector to or IDT vector number for the descriptor; otherwise, the error code is 0. The source of the selector in an error code may be any of the following:

- An operand of the instruction.
- A selector from a gate which is the operand of the instruction.
- A selector from a TSS involved in a task switch.
- IDT vector number.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

In general, a program-state change does not accompany a general-protection exception, because the invalid instruction or operation is not executed. An exception handler can be designed to correct all of the conditions that cause general-protection exceptions and restart the program or task without any loss of program continuity.

If a general-protection exception occurs during a task switch, it can occur before or after the commit-to-new-task point (see Section 7.3, "Task Switching"). If it occurs before the commit point, no program state change occurs. If it occurs after the commit point, the processor will load all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The general-protection exception handler should thus not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. (See the Program State Change description for "Interrupt 10—Invalid TSS Exception (#TS)" in this chapter for additional information on how to handle this situation.)

General Protection Exception in 64-bit Mode

The following conditions cause general-protection exceptions in 64-bit mode:

- If the memory address is in a non-canonical form.
- If a segment descriptor memory address is in non-canonical form.
- If the target offset in a destination operand of a call or jmp is in a non-canonical form.
- If a code segment or 64-bit call gate overlaps non-canonical space.
- 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 EFLAGS.NT bit is set in IRET.
- If the stack segment selector of IRET is null when going back to compatibility mode.
- If the stack segment selector of IRET is null going back to CPL3 and 64-bit mode.
- If a null stack segment selector RPL of IRET is not equal to CPL going back to non-CPL3 and 64-bit mode.
- If the proposed new code segment descriptor of IRET has both the D-bit and the L-bit set.
- If the segment descriptor pointed to by the segment selector in the destination operand is a code segment and it has both the D-bit and the L-bit set.
- If the segment descriptor from a 64-bit call gate is in non-canonical space.
- 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 type field of the upper 64 bits of a 64-bit call gate is not 0.
- If an attempt is made to load a null selector in the SS register in compatibility mode.
- If an attempt is made to load null selector in the SS register in CPL3 and 64-bit mode.
- If an attempt is made to load a null selector in the SS register in non-CPL3 and 64-bit mode where RPL is not equal to CPL.
- If an attempt is made to clear CR0.PG while IA-32e mode is enabled.
- If an attempt is made to set a reserved bit in CR3, CR4 or CR8.

...

12. Updates to Chapter 7, Volume 3A

Change bars show changes to Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

7.2.4 Task Register

The task register holds the 16-bit segment selector and the entire segment descriptor (32-bit base address (64 bits in IA-32e mode), 16-bit segment limit, and descriptor attributes) for the TSS of the current task (see Figure 2-6). This information is copied from the TSS descriptor in the GDT for the current task. Figure 7-5 shows the path the processor uses to access the TSS (using the information in the task register).

The task register has a visible part (that can be read and changed by software) and an invisible part (maintained by the processor and is inaccessible by software). The segment selector in the visible portion points to a TSS descriptor in the GDT. The processor uses the invisible portion of the task register to cache the segment descriptor for the TSS. Caching these values in a register makes execution of the task more efficient. The LTR (load task register) and STR (store task register) instructions load and read the visible portion of the task register:

The LTR instruction loads a segment selector (source operand) into the task register that points to a TSS descriptor in the GDT. It then loads the invisible portion of the task register with information from the TSS descriptor. LTR is a privileged instruction that may be executed only when the CPL is 0. It's used during system initialization to put an initial value in the task register. Afterwards, the contents of the task register are changed implicitly when a task switch occurs.

The STR (store task register) instruction stores the visible portion of the task register in a general-purpose register or memory. This instruction can be executed by code running at any privilege level in order to identify the currently running task. However, it is normally used only by operating system software. (If CR4.UMIP = 1, STR can be executed only when CPL = 0.)

On power up or reset of the processor, segment selector and base address are set to the default value of 0; the limit is set to FFFFH.

...

13. Updates to Chapter 9, Volume 3A

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

...

9.1.2 Processor Built-In Self-Test (BIST)

Hardware may request that the BIST be performed at power-up. The EAX register is cleared (0H) if the processor passes the BIST. A nonzero value in the EAX register after the BIST indicates that a processor fault was detected. If the BIST is not requested, the contents of the EAX register after a hardware reset is 0H.

The overhead for performing a BIST varies between processor families. For example, the BIST takes approximately 30 million processor clock periods to execute on the Pentium 4 processor. This clock count is model-specific; Intel reserves the right to change the number of periods for any Intel 64 or IA-32 processor, without notification.

Table 9-1 IA-32 Processor States Following Power-up, Reset, or INIT

Register	Pentium 4 and Intel Xeon Processor	P6 Family Processor (Including DisplayFamily = 06H)	Pentium Processor
EFLAGS[1]	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H[2]	60000010H[2]	60000010H[2]
CR2, CR3, CR4	00000000H	00000000H	00000000H

CS	Selector = F000H
	Base = FFFF0000H
	Limit = FFFFH
	AR = Present, R/W, Accessed
		Selector = F000H
		Base = FFFF0000H
		Limit = FFFFH
		AR = Present, R/W, Accessed
			Selector = F000H
			Base = FFFF0000H
			Limit = FFFFH
			AR = Present, R/W, Accessed

SS, DS, ES, FS, GS	Selector = 0000H
	Base = 00000000H
	Limit = FFFFH
	AR = Present, R/W, Accessed
		Selector = 0000H
		Base = 00000000H
		Limit = FFFFH
		AR = Present, R/W, Accessed
			Selector = 0000H
			Base = 00000000H
			Limit = FFFFH
			AR = Present, R/W, Accessed

EDX	00000FxxH	000n06xxH[3]	000005xxH
EAX	0[4]	0[4]	0[4]
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H
ST0 through ST7[5]	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged
x87 FPU Control Word[5]	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH
x87 FPU Status Word[5]	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H
x87 FPU Tag Word[5]	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH
x87 FPU Data Operand and CS Seg. Selectors[5]	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H
x87 FPU Data Operand and Inst. Pointers[5]	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H
MM0 through MM7[5]	Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged	Pentium II and Pentium III Processors Only— Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged	Pentium with MMX Technology Only— Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged
XMM0 through XMM7	Pwr up or Reset: 0H INIT: Unchanged	If CPUID.01H:SSE is 1 — Pwr up or Reset: 0H INIT: Unchanged	NA
MXCSR	Pwr up or Reset: 1F80H INIT: Unchanged	Pentium III processor only- Pwr up or Reset: 1F80H INIT: Unchanged	NA

Table 9-1 IA-32 Processor States Following Power-up, Reset, or INIT (Contd.)

Register	Pentium 4 and Intel Xeon Processor	P6 Family Processor (Including DisplayFamily = 06H)	Pentium Processor

GDTR, IDTR	Base = 00000000H
	Limit = FFFFH
	AR = Present, R/W
		Base = 00000000H
		Limit = FFFFH
		AR = Present, R/W
			Base = 00000000H
			Limit = FFFFH
			AR = Present, R/W

LDTR, Task Register	Selector = 0000H
	Base = 00000000H
	Limit = FFFFH
	AR = Present, R/W
		Selector = 0000H
		Base = 00000000H
		Limit = FFFFH
		AR = Present, R/W
			Selector = 0000H
			Base = 00000000H
			Limit = FFFFH
			AR = Present, R/W

DR0, DR1, DR2, DR3	00000000H	00000000H	00000000H
DR6	FFFF0FF0H	FFFF0FF0H	FFFF0FF0H
DR7	00000400H	00000400H	00000400H
Time-Stamp Counter	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged
Perf. Counters and Event Select	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged
Intel® Processor Trace[6]	Power up or Cold Reset: 0H Warm Reset: Unchanged, but IA32_RTIT_CTL.TraceEn[0] cleared[7] INIT: Unchanged	N.A.	N.A.
All Other MSRs	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged
Data and Code Cache, TLBs	Invalid[8]	Invalid[8]	Invalid[8]
Fixed MTRRs	Pwr up or Reset: Disabled INIT: Unchanged	Pwr up or Reset: Disabled INIT: Unchanged	Not Implemented
Variable MTRRs	Pwr up or Reset: Disabled INIT: Unchanged	Pwr up or Reset: Disabled INIT: Unchanged	Not Implemented
Machine-Check Architecture	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged	Not Implemented
APIC	Pwr up or Reset: Enabled INIT: Unchanged	Pwr up or Reset: Enabled INIT: Unchanged	Pwr up or Reset: Enabled INIT: Unchanged
R8-R15[9]	0000000000000000H	0000000000000000H	N.A.

Table 9-1 IA-32 Processor States Following Power-up, Reset, or INIT (Contd.)

Register	Pentium 4 and Intel Xeon Processor	P6 Family Processor (Including DisplayFamily = 06H)	Pentium Processor
XMM8-XMM15[9]	Pwr up or Reset: 0H INIT: Unchanged	Pwr up or Reset: 0H INIT: Unchanged	N.A.
YMMn[128:VLMAX][10]	N.A.	Pwr up or Reset: 0H INIT: Unchanged	N.A.

NOTES:

1. The 10 most-significant bits of the EFLAGS register are undefined following a reset. Software should not depend on the states of any of these bits.
2. The CD and NW flags are unchanged, bit 4 is set to 1, all other bits are cleared.
3. Where "n" is the Extended Model Value for the respective processor.
4. If Built-In Self-Test (BIST) is invoked on power up or reset, EAX is 0 only if all tests passed. (BIST cannot be invoked during an INIT.)
5. The state of the x87 FPU and MMX registers is not changed by the execution of an INIT.
6. If the processor supports Intel® Processor Trace.
7. If CPUID(EAX=15H, ECX=0):EBX.IPFILT_WRSPRSV[bit 2] = 1.
8. Internal caches are invalid after power-up and RESET, but left unchanged with an INIT.
9. If the processor supports IA-32e mode.
10. If the processor supports AVX.

...

9.11.6 Microcode Update Loader

This section describes an update loader used to load an update into a P6 family or later processors. It also discusses the requirements placed on the BIOS to ensure proper loading. The update loader described contains the minimal instructions needed to load an update. The specific instruction sequence that is required to load an update is dependent upon the loader revision field contained within the update header. This revision is expected to change infrequently (potentially, only when new processor models are introduced).

Example 9-8 below represents the update loader with a loader revision of 00000001H. Note that the microcode update must be aligned on a 16-byte boundary and the size of the microcode update must be 1-KByte granular.

Example 9-8 Assembly Code Example of Simple Microcode Update Loader

```
mov  ecx,79h              ; MSR to write in ECX
xor  eax,eax              ; clear EAX
xor  ebx,ebx              ; clear EBX
mov  ax,cs                ; Segment of microcode update
shl  eax,4
mov  bx,offset Update     ; Offset of microcode update
add  eax,ebx              ; Linear Address of Update in EAX
add  eax,48d              ; Offset of the Update Data within the Update
xor  edx,edx              ; Zero in EDX

WRMSR                    ; microcode update trigger
```

The loader shown in Example 9-8 assumes that *update* is the address of a microcode update (header and data) embedded within the code segment of the BIOS. It also assumes that the processor is operating in real mode. The data may reside anywhere in memory, aligned on a 16-byte boundary, that is accessible by the processor within its current operating mode.

Before the BIOS executes the microcode update trigger (WRMSR) instruction, the following must be true:

- In 64-bit mode, EAX contains the lower 32-bits of the microcode update linear address. In protected mode, EAX contains the full 32-bit linear address of the microcode update.
- In 64-bit mode, EDX contains the upper 32-bits of the microcode update linear address. In protected mode, EDX equals zero.
- ECX contains 79H (address of IA32_BIOS_UPDT_TRIG).

Other requirements are:

- If the update is loaded while the processor is in real mode, then the update data may not cross a segment boundary.
- If the update is loaded while the processor is in real mode, then the update data may not exceed a segment limit.
- If paging is enabled, pages that are currently present must map the update data.
- The microcode update data requires a 16-byte boundary alignment.

...

14. Updates to Chapter 10, Volume 3A

Change bars show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

10.4.6 Local APIC ID

At power up, system hardware assigns a unique APIC ID to each local APIC on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for P6 family and Pentium processors). The hardware assigned APIC ID is based on system topology and includes encoding for socket position and cluster information (see Figure 8-2).

In MP systems, the local APIC ID is also used as a processor ID by the BIOS and the operating system. Some processors permit software to modify the APIC ID. However, the ability of software to modify the APIC ID is processor model specific. Because of this, operating system software should avoid writing to the local APIC ID register. The value returned by bits 31-24 of the EBX register (when the CPUID instruction is executed with a source operand value of 1 in the EAX register) is always the Initial APIC ID (determined by the platform initialization). This is true even if software has changed the value in the Local APIC ID register.

The processor receives the hardware assigned APIC ID (or Initial APIC ID) by sampling pins A11# and A12# and pins BR0# through BR3# (for the Pentium 4, Intel Xeon, and P6 family processors) and pins BE0# through BE3# (for the Pentium processor). The APIC ID latched from these pins is stored in the APIC ID field of the local APIC ID register (see Figure 10-6), and is used as the Initial APIC ID for the processor.

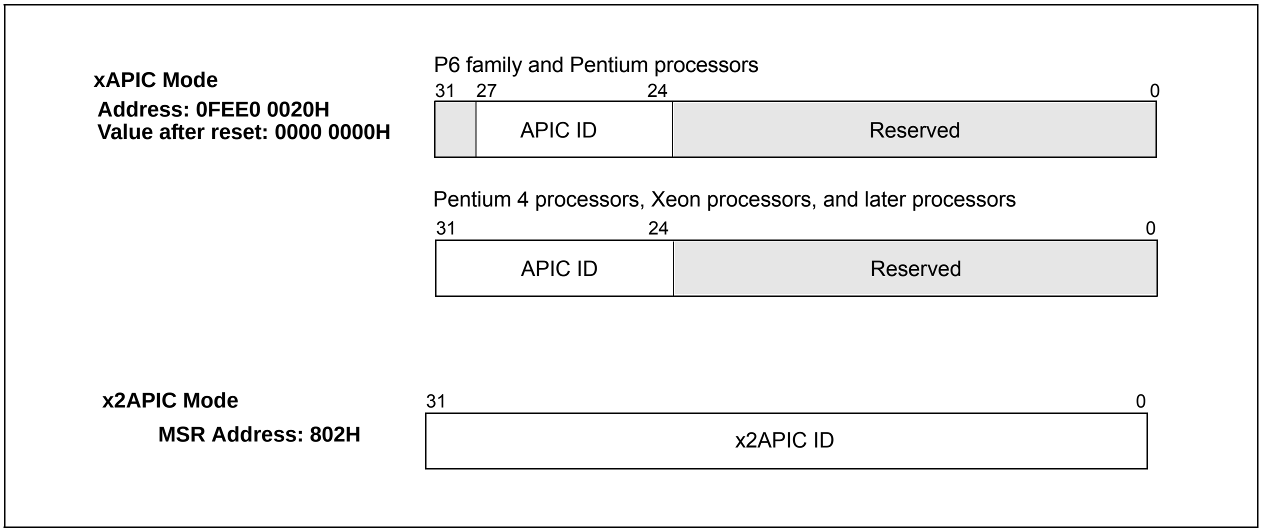

xAPIC Mode
Address: 0FEE0 0020H
Value after reset: 0000 0000H
P6 family and Pentium processors
31
27
24
0
APIC ID
Reserved
Pentium 4 processors, Xeon processors, and later processors
31
24
0
APIC ID
Reserved
x2APIC Mode
MSR Address: 802H
31
0
x2APIC ID

Figure 10-6 Local APIC ID Register

For the P6 family and Pentium processors, the local APIC ID field in the local APIC ID register is 4 bits. Encodings 0H through EH can be used to uniquely identify 15 different processors connected to the APIC bus. For the Pentium 4 and Intel Xeon processors, the xAPIC specification extends the local APIC ID field to 8 bits. These can be used to identify up to 255 processors in the system.

...

10.5.4 APIC Timer

The local APIC unit contains a 32-bit programmable timer that is available to software to time events or operations. This timer is set up by programming four registers: the divide configuration register (see Figure 10-10), the initial-count and current-count registers (see Figure 10-11), and the LVT timer register (see Figure 10-8).

If CPUID.06H:EAX.ARAT[bit 2] = 1, the processor’s APIC timer runs at a constant rate regardless of P-state transitions and it continues to run at the same rate in deep C-states.

If CPUID.06H:EAX.ARAT[bit 2] = 0 or if CPUID 06H is not supported, the APIC timer may temporarily stop while the processor is in deep C-states or during transitions caused by Enhanced Intel SpeedStep® Technology.

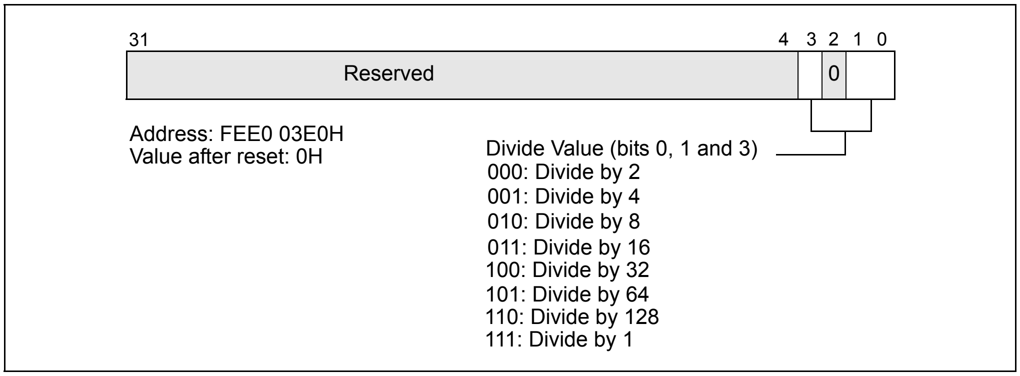


Figure 10-10 Divide Configuration Register

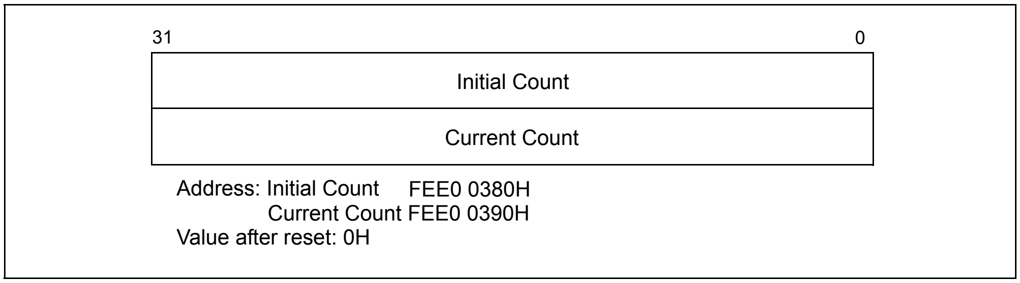


Figure 10-11 Initial Count and Current Count Registers

The APIC timer frequency will be the processor's bus clock or core crystal clock frequency (when TSC/core crystal clock ratio is enumerated in CPUID leaf 0x15) divided by the value specified in the divide configuration register.

The timer can be configured through the timer LVT entry for one-shot or periodic operation. In one-shot mode, the timer is started by programming its initial-count register. The initial count value is then copied into the current-count register and count-down begins. After the timer reaches zero, an timer interrupt is generated and the timer remains at its 0 value until reprogrammed.

In periodic mode, the current-count register is automatically reloaded from the initial-count register when the count reaches 0 and a timer interrupt is generated, and the count-down is repeated. If during the count-down process the initial-count register is set, counting will restart, using the new initial-count value. The initial-count register is a read-write register; the current-count register is read only.

A write of 0 to the initial-count register effectively stops the local APIC timer, in both one-shot and periodic mode.

The LVT timer register determines the vector number that is delivered to the processor with the timer interrupt that is generated when the timer count reaches zero. The mask flag in the LVT timer register can be used to mask the timer interrupt.

...

15. Updates to Chapter 14, Volume 3B

Change bars show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

14.3.2.3 Required Changes to OS Power Management P-state Policy

Intel Dynamic Acceleration (IDA) 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 IDA. 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 IDA, some of these are listed below:

- When engaging ACPI defined passive thermal management, it may be more effective to disable IDA 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 IDA while that policy is in effect.

...

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 SpeedStep® 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 enumerate 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.

...

16. Updates to Chapter 16, Volume 3B

Change bars show changes to Chapter 16 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

16.9 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY WITH CPUID DISPLAYFAMILY_DISPLAYMODEL SIGNATURE 06_55H, MACHINE ERROR CODES FOR MACHINE CHECK

Future Intel Xeon processors with CPUID DisplayFamily_DisplaySignature 06_55H. Incremental error codes for internal machine check error from PCU controller is reported in the register bank IA32_MC4, Table 16-27 in Section 16.9.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

16.9.1 Internal Machine Check Errors

Table 16-27 Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
MCACOD[2]	15:0	internal Errors	0402h - PCU internal Errors 0403h - PCU internal Errors 0406h - Intel TXT Errors 0407h - Other UBOX internal Errors. On an IERR caused by a core 3-strike the IA32_MC3_STATUS (MLC) is copied to the IA32_MC4_STATUS (After a 3-strike, the core MCA banks will be unavailable).
Model specific errors	19:16	Reserved except for the following	0000b - No Error 00xxb - PCU internal error
	23-20	Reserved	Reserved
	31-24	Reserved except for the following	00h - No Error 0Dh - MCA_DMI_TRAINING_TIMEOUT 0Fh - MCA_DMI_CPU_RESET_ACK_TIMEOUT 10h - MCA_MORE_THAN_ONE_LT_AGENT 1Eh - MCA_BIOS_RST_CPL_INVALID_SEQ 1Fh - MCA_BIOS_INVALID_PKG_STATE_CONFIG 25h - MCA_MESSAGE_CHANNEL_TIMEOUT 27h - MCA_MSGCH_PMREQ_CMP_TIMEOUT 30h - MCA_PKGC_DIRECT_WAKE_RING_TIMEOUT 31h - MCA_PKGC_INVALID_RSP_PCH 33h - MCA_PKGC_WATCHDOG_HANG_CBZ_DOWN 34h - MCA_PKGC_WATCHDOG_HANG_CBZ_UP 38h - MCA_PKGC_WATCHDOG_HANG_C3_UP_SF 40h - MCA_SVID_VCCIN_VR_ICC_MAX_FAILURE 41h - MCA_SVID_COMMAND_TIMEOUT 42h - MCA_SVID_VCCIN_VR_VOUT_MAX_FAILURE 43h - MCA_SVID_CPU_VR_CAPABILITY_ERROR 44h - MCA_SVID_CRITICAL_VR_FAILED 45h - MCA_SVID_SA_ITD_ERROR 46h - MCA_SVID_READ_REG_FAILED 47h - MCA_SVID_WRITE_REG_FAILED 48h - MCA_SVID_PKGC_INIT_FAILED

Type	Bit No.	Bit Function	Bit Description
			49h - MCA_SVID_PKGC_CONFIG_FAILED 4Ah - MCA_SVID_PKGC_REQUEST_FAILED 4Bh - MCA_SVID_IMON_REQUEST_FAILED 4Ch - MCA_SVID_ALERT_REQUEST_FAILED 4Dh - MCA_SVID_MCP_VP_ABSENT_OR_RAMP_ERROR 4Eh - MCA_SVID_UNEXPECTED_MCP_VP_DETECTED 51h - MCA_FIVR_CATAS_OVERVOL_FAULT 52h - MCA_FIVR_CATAS_OVERCUR_FAULT 58h - MCA_WATCHDG_TIMEOUT_PKGC_SLAVE 59h - MCA_WATCHDG_TIMEOUT_PKGC_MASTER 5Ah - MCA_WATCHDG_TIMEOUT_PKGS_MASTER 61h - MCA_PKGS_CPD_UNPCD_TIMEOUT 63h - MCA_PKGS_INVALID_REQ_PCH 64h - MCA_PKGS_INVALID_REQ_INTERNAL 65h - MCA_PKGS_INVALID_RSP_INTERNAL 6Bh - MCA_PKGS_SMBUS_VPP_PAUSE_TIMEOUT 81h - MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	52-32	Reserved	Reserved
	54-53	CORR_ERR_STATUS	Reserved
	56-55	Reserved	Reserved
Status register validity indicators[1]	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, “Machine-Check Architecture,” for more information.
2. The internal error codes may be model-specific.

16.9.2 Interconnect Machine Check Errors

MC error codes associated with the link interconnect agents are reported in the MSRs IA32_MC5_STATUS, IA32_MC12_STATUS, IA32_MC19_STATUS. The supported error codes follow the architectural MCACOD definition type 1PPTRRRRIILL (see Chapter 15, “Machine-Check Architecture,”).

Table 16-28 lists model-specific fields to interpret error codes applicable to IA32_MCi_STATUS, i= 5, 12, 19.

Table 16-28 Interconnect MC Error Codes for IA32_MCi_STATUS, i = 5, 12, 19

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	0-15	MCACOD	Bus error format: 1PPTRRRRIILL The two supported compound error codes: - 0x0C0F - Unsupported/Undefined Packet - 0x0E0F - For all other corrected and uncorrected errors
Model specific errors	21-16	MSCOD	The encoding of Uncorrectable (UC) errors are: 00h - UC Phy Initialization Failure. 01h - UC Phy detected drift buffer alarm. 02h - UC Phy detected latency buffer rollover. 10h - UC link layer Rx detected CRC error: unsuccessful LLR entered abort state 11h - UC LL Rx unsupported or undefined packet. 12h - UC LL or Phy control error. 13h - UC LL Rx parameter exchange exception. 1fh - UC LL detected control error from the link-mesh interface The encoding of correctable (COR) errors are: 20h - COR Phy initialization abort 21h - COR Phy reset 22h - COR Phy lane failure, recovery in x8 width. 23h - COR Phy L0c error corrected without Phy reset 24h - COR Phy L0c error triggering Phy reset 25h - COR Phy L0p exit error corrected with Phy reset 30h - COR LL Rx detected CRC error - successful LLR without Phy re-init. 31h - COR LL Rx detected CRC error - successful LLR with Phy re-init. All other values are reserved.

Type	Bit No.	Bit Function	Bit Description
	31-22	MSCOD_SPARE	The definition below applies to MSCOD 12h (UC LL or Phy Control Errors) [Bit 22] : Phy Control Error [Bit 23] : Unexpected Retry.Ack flit [Bit 24] : Unexpected Retry.Req flit [Bit 25] : RF parity error [Bit 26] : Routeback Table error [Bit 27] : unexpected Tx Protocol flit (EOP, Header or Data) [Bit 28] : Rx Header-or-Credit BGF credit overflow/underflow [Bit 29] : Link Layer Reset still in progress when Phy enters L0 (Phy training should not be enabled until after LL reset is complete as indicated by KTILCL.LinkLayerReset going back to 0). [Bit 30] : Link Layer reset initiated while protocol traffic not idle [Bit 31] : Link Layer Tx Parity Error
	37-32	Reserved	Reserved
	52-38	Corrected Error Cnt	
	56-53	Reserved	Reserved
Status register validity indicators[1]	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, “Machine-Check Architecture,” for more information.

16.9.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC13_STATUS-IA32_MC16_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, “Machine-Check Architecture”).

Table 16-29 Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13-16)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	0-15	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	0001H - Address parity error 0002H - HA write data parity error 0004H - HA write byte enable parity error 0008H - Corrected patrol scrub error 0010H - Uncorrected patrol scrub error 0020H - Corrected spare error 0040H - Uncorrected spare error 0080H - Any HA read error 0100H - WDB read parity error 0200H - DDR4 command address parity error 0400H - Uncorrected address parity error 0800H - Unrecognized request type 0801H - Read response to an invalid scoreboard entry 0802H - Unexpected read response 0803H - DDR4 completion to an invalid scoreboard entry 0804H - Completion to an invalid scoreboard entry 0805H - Completion FIFO overflow 0806H - Correctable parity error 0807H - Uncorrectable error 0808H - Interrupt received while outstanding interrupt was not ACKed 0809H - ERID FIFO overflow 080aH - Error on Write credits 080bH - Error on Read credits 080cH - Scheduler error 080dH - Error event
	36-32	Other info	MC logs the first error device. This is an encoded 5-bit value of the device.
	37	Reserved	Reserved
	56-38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.9.4 M2M Machine Check Errors

MC error codes associated with M2M are reported in the MSRs IA32_MC7_STATUS, IA32_MC8_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

Table 16-30 M2M MC Error Codes for IA32_MCi_STATUS (i= 7-8)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	0-15	MCACOD	Compound error format: 0000 0000 1MMM CCCC
Model specific errors	16	MscodDataRdErr	Logged an MC read data error
	17	Reserved	Reserved
	18	MscodPtlWrErr	Logged an MC partial write data error
	19	MscodFullWrErr	Logged a full write data error
	20	MscodBgfErr	Logged an M2M clock-domain-crossing buffer (BGF) error
	21	MscodTimeOut	Logged an M2M time out
	22	MscodParErr	Logged an M2M tracker parity error
	23	MscodBucket1Err	Logged a fatal Bucket1 error
	31-24	Reserved	Reserved
	36-32	Other info	MC logs the first error device. This is an encoded 5-bit value of the device.
	37	Reserved	Reserved
	56-38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

...

17. Updates to Chapter 17, Volume 3B

Change bars show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

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 17.7, “Last Branch, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Nehalem”
- Section 17.8, “Last Branch, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Sandy Bridge”
- Section 17.9, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Haswell Microarchitecture”
- Section 17.10, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture”
- Section 17.12, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors)”
- Section 17.13, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors)”
- Section 17.14, “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).

...

17.4.2 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.

...

17.4.8 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	32	FROM_IP, TO_IP, LBR_INFO[1]	0 to 31
06_3DH, 06_47H, 06_4FH, 06_56H	16	FROM_IP, TO_IP	0 to 15
06_3CH, 06_45H, 06_46H, 06_3FH	16	FROM_IP, TO_IP	0 to 15
06_2AH, 06_2DH, 06_3AH, 06_3EH	16	FROM_IP, TO_IP	0 to 15

Table 17-4 LBR Stack Size and TOS Pointer Range (Contd.)

DisplayFamily_DisplayModel	Size of LBR Stack	Component of an LBR Entry	Range of TOS Pointer
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	4	FROM_IP, TO_IP	0 to 3
06_0FH	4	FROM_IP, TO_IP	0 to 3
06_37H, 06_4AH, 06_4CH, 06_4DH, 06_5AH, 06_5DH	8	FROM_IP, TO_IP	0 to 7
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	8	FROM_IP, TO_IP	0 to 7

NOTES:

1. See Section 17.10.

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 employs a set of MSRs, referred to as last branch record (LBR) stack. The size and exact locations of the LBR stack are generally model-specific (see Chapter 35, "Model-Specific Registers (MSRs)" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* 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.

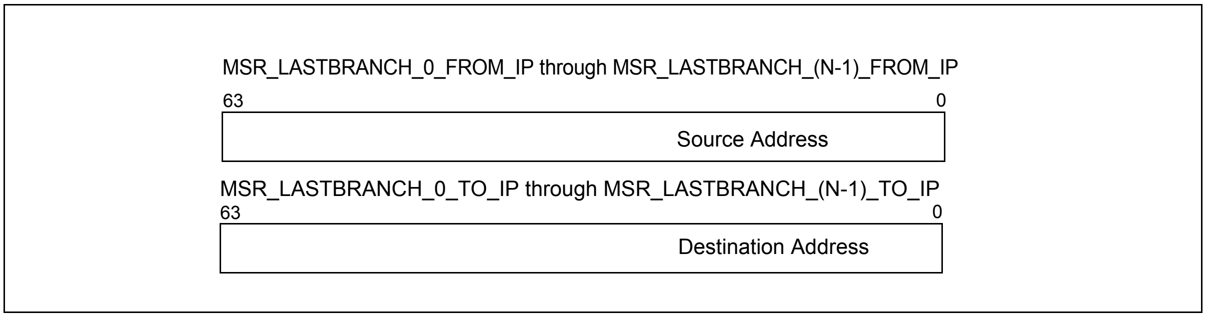

MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_(N-1)_FROM_IP
63
0
Source Address
MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_(N-1)_TO_IP
63
0
Destination Address

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 EIP 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).

Processor’s support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX[PERF_CAPAB_MSR] (bit 15).

...

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 18.4.4, “Processor Event Based Sampling (PEBS),” and the relevant PEBS sub-sections across the core PMU sections in Chapter 18, “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 18.4.4.2 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 18.7.1 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 transition to system-management mode (SMM). Similarly, the recording mechanism is disabled on the generation of a machine-check exception and is cleared on processor RESET and INIT. DS recording is available in real-address mode.

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 35).

The DS save area is divided into three parts (see Figure 17-5): buffer management area, branch trace store (BTS) buffer, and PEBS buffer. 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 40-bit value that the counter is to be reset to after state information has collected following counter overflow. This value allows state information to be collected after a preset number of events have been counted.

...

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 17.11.3.
- **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_EN (bit 14)** — FREEZE_WHILE_SMM_EN 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

Next generation Intel Atom processors are based on the 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. MISPRED bit format is shown in Table 17-8.
- Streamlined Freeze_LBRs_On_PMI operation; see Section 17.10.2.
- LBR MSRs are cleared when software requests C6 or deeper sleep-state; see Section 17.10.3.

Table 17-7 MSR_LASTBRANCH_x_TO_IP for the Goldmont Microarchitecture

Bit Field	Bit Offset	Access	Description
Data	47:0	R/O	This is the “branch to“ address. See Section 17.4.8.1 for address format.
Cycle Count (Saturating)	63:48	R/O	Elapsed core clocks since last update to the LBR stack.

...

17.7.1 LBR Stack

Processors based on Intel microarchitecture code name Nehalem 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/O	This is the “branch from“ address. See Section 17.4.8.1 for address format.
SIGN_EXt	62:48	R/O	Signed extension of bit 47 of this register.
MISPRED	63	R/O	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/O	This is the “branch to“ address. See Section 17.4.8.1 for address format
SIGN_EXt	63:48	R/O	Signed extension of bit 47 of this register.

Processors based on Intel microarchitecture code name Nehalem 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

...

17.9.1 LBR Stack Enhancement

Processors based on Intel microarchitecture code name Haswell 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/O	This is the “branch from“ address. See Section 17.4.8.1 for address format.
SIGN_EXT	60:48	R/O	Signed extension of bit 47 of this register.
TSX_ABORT	61	R/O	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/O	When set, indicates the entry occurred in a TSX region
MISPRED	63	R/O	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.

17.10 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

...

17.10.3 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.

...

17.16 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.

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.

...

17.16.8 Monitoring Programming Considerations

Figure 17-23 illustrates how system software can program IA32_QOSEVTSEL and IA32_QM_CTR to perform resource monitoring.

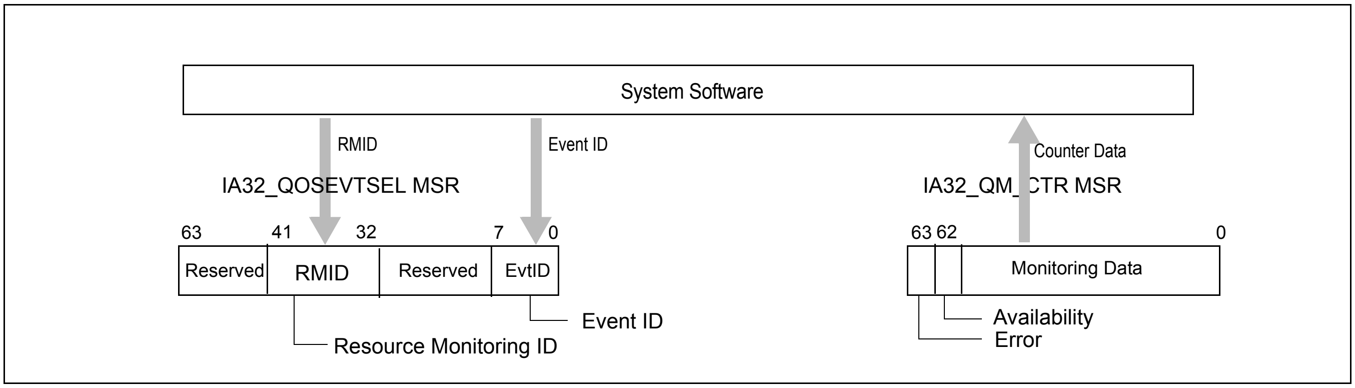


Figure 17-25 Software Usage of Cache Monitoring Resources

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.

...

17.17 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 subset of communication-focused Intel Xeon processors E5 v3 family) introduce capabilities to configure and make use of the Cache Allocation Technology (CAT) mechanisms on the L3 cache. Some future Intel platforms may 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.

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 famility, 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 give 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.

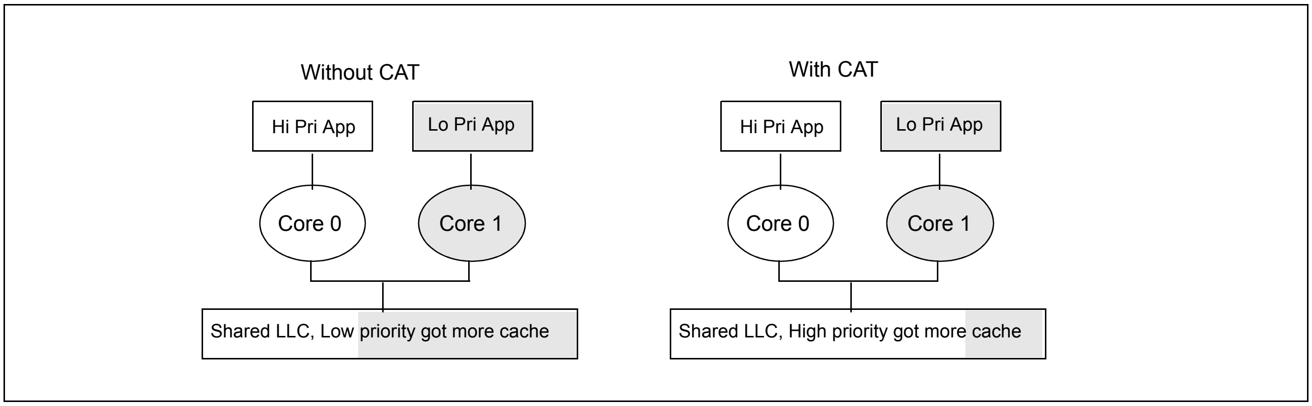


Figure 17-26 Cache Allocation Technology Allocates More Resource to High Priority Applications

17.17.1 Cache Allocation Technology Architecture

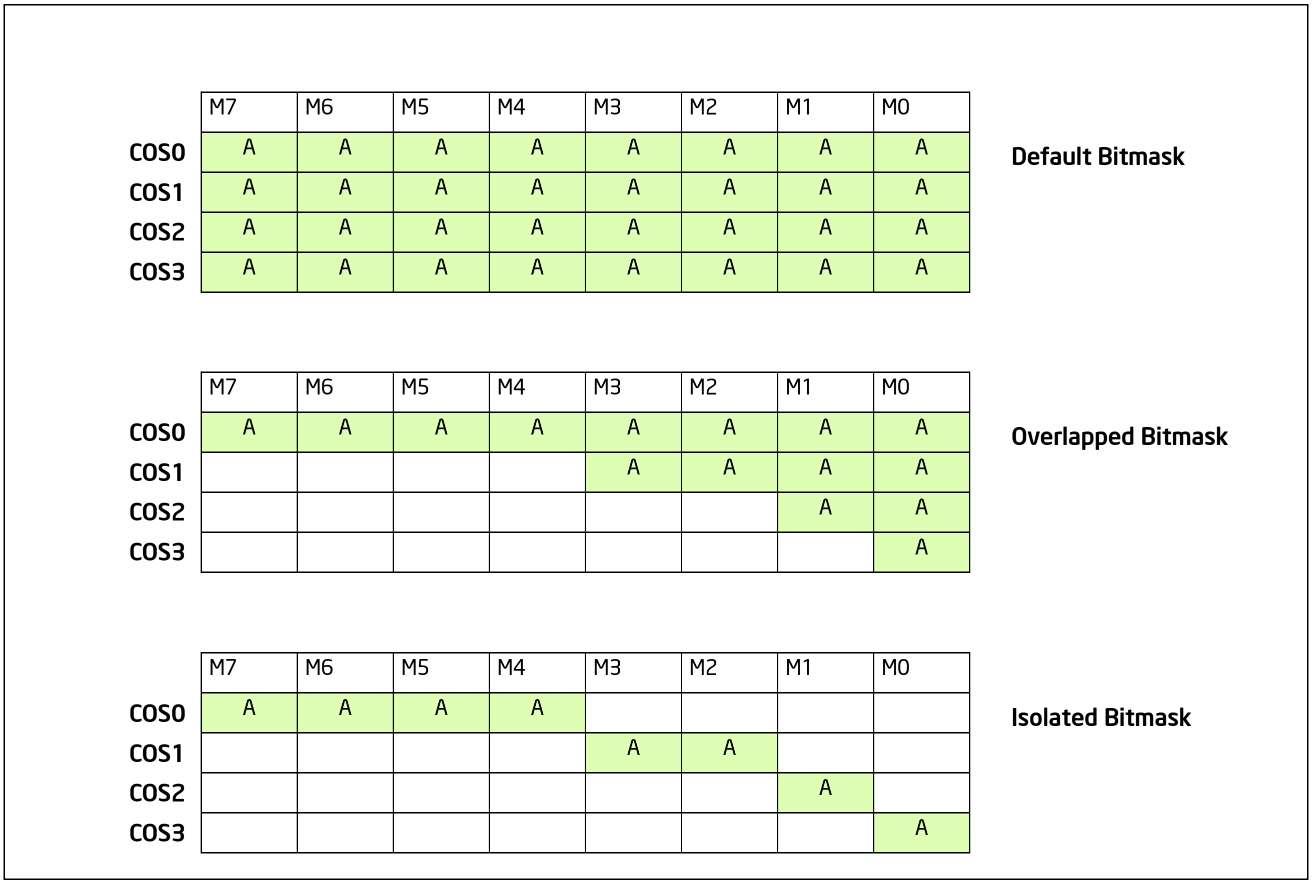
The fundamental goal of Cache Allocation Technology is to enable resource allocation based on application priority or Class of Service (COS or CLOS). The processor exposes a set of Classes of Service into which applications (or individual threads) can be assigned. Cache allocation for the respective 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 re-source 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/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 bitlength 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.



Default Bitmask

	M7	M6	M5	M4	M3	M2	M1	M0
COS0	A	A	A	A	A	A	A	A
COS1	A	A	A	A	A	A	A	A
COS2	A	A	A	A	A	A	A	A
COS3	A	A	A	A	A	A	A	A

Overlapped Bitmask

	M7	M6	M5	M4	M3	M2	M1	M0
COS0	A	A	A	A	A	A	A	A
COS1					A	A	A	A
COS2							A	A
COS3								A

Isolated Bitmask

	M7	M6	M5	M4	M3	M2	M1	M0
COS0	A	A	A	A				
COS1					A	A		
COS2							A	
COS3								A

Figure 17-27 Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bitlength 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.

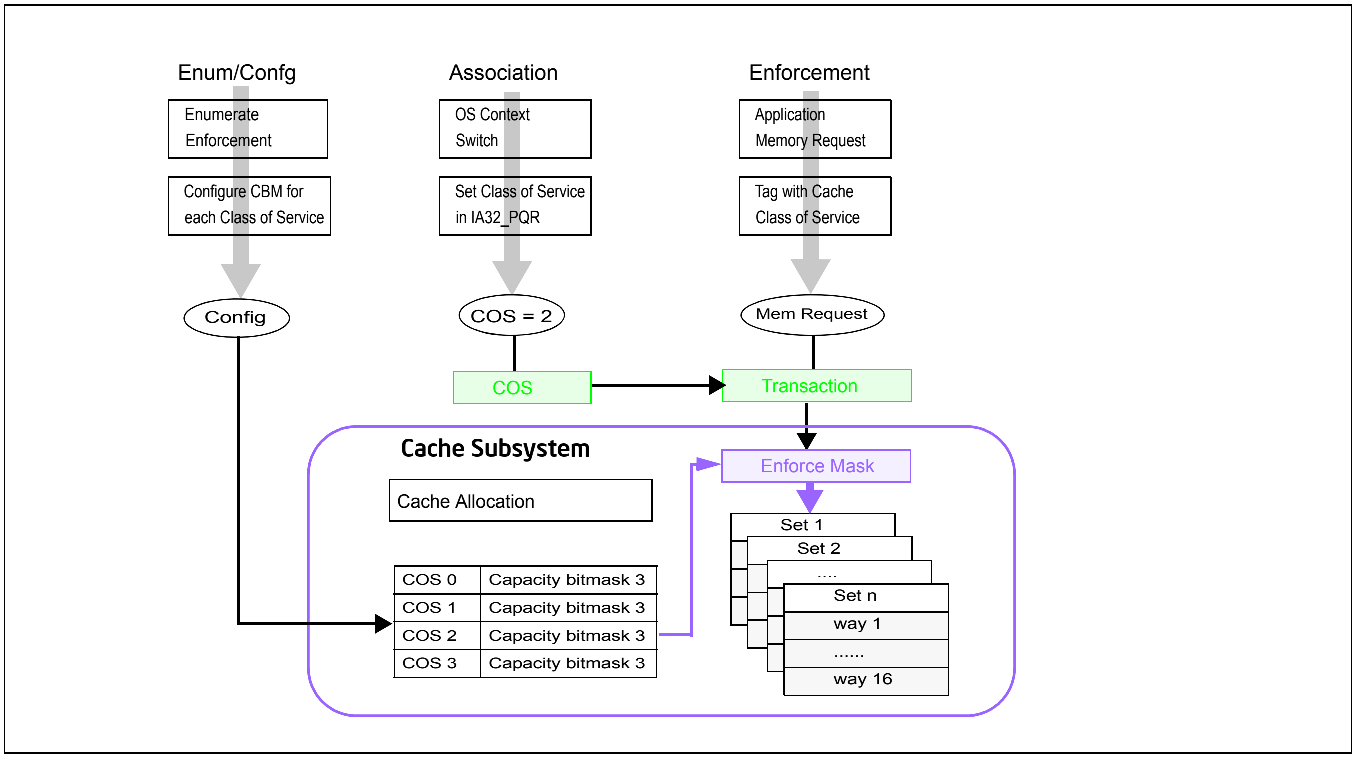


Figure 17-28 Class of Service and Cache Capacity Bitmasks

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 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, and L2 CAT. Depending on the specific processor a mix of features may be supported, and CPUID provides enumeration capabilities to enable software to detect the set of supported features.

...

17.17.3 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.

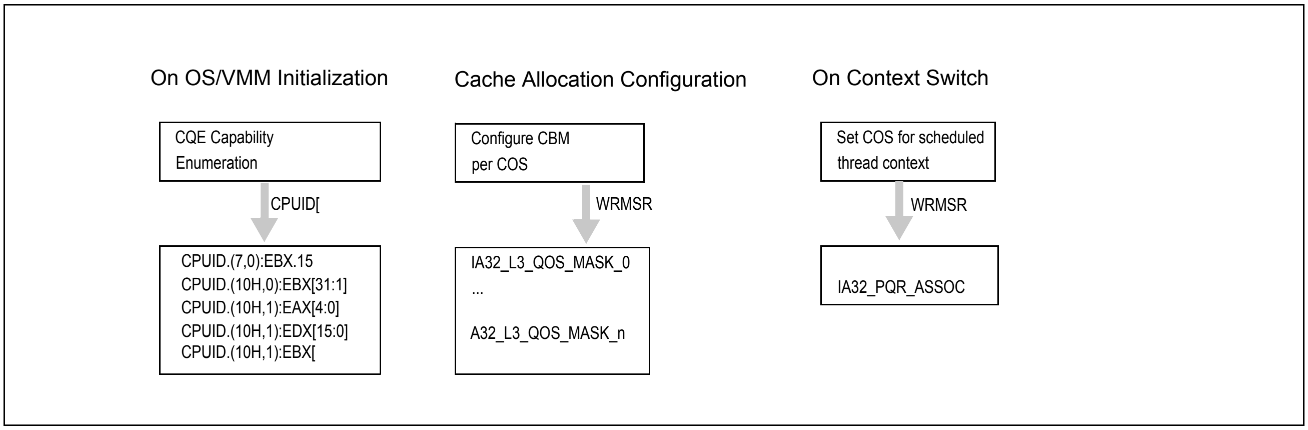


Figure 17-30 Cache Allocation Technology Usage Flow

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.

17.17.3.1 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 17.17.3.2).
- 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 17.17.3.3 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 17.17.3.3 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 CMCI and Machine Check Resources". Software may also use CPUID leaf 4 to determine the maximum number of logical processor IDs that may share a given level of the cache.

- IA32_PQR_ASSOC.CLOS: The IA32_PQR_ASSOC MSR provides a COS field that OS/VMM can use to assign a logical processor to an available COS. The set of COS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software COS management overhead at context swap time. See Section 17.17.3.4 for details.

17.17.3.2 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).

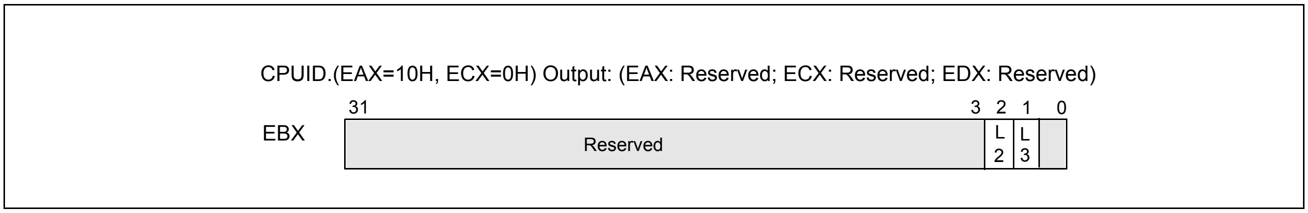

CPUID.(EAX=10H, ECX=0H) Output: (EAX: Reserved; ECX: Reserved; EDX: Reserved)
31
3
2
1
0
EBX
Reserved
L 2
L 3

Figure 17-31 CPUID.(EAX=10H, ECX=0H) Available Resource Type Identification

— EAX[4:0] reports the length of the capacity bitmask length using minus-one notation, i.e. a value of 15 corresponds to the capability 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:

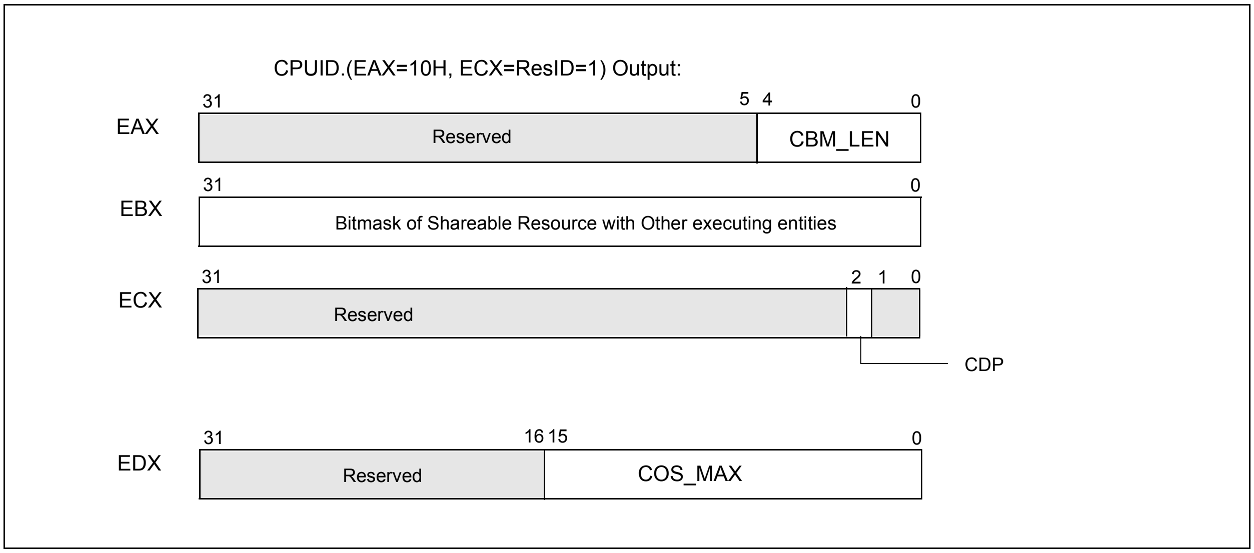

CPUID.(EAX=10H, ECX=ResID=1) Output:
EAX
31
5
4
0
Reserved
CBM_LEN
EBX
31
0
Bitmask of Shareable Resource with Other executing entities
ECX
31
2
1
0
Reserved
CDP
EDX
31
16
15
0
Reserved
COS_MAX

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 length using minus-one notation, i.e. 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 Code and Data Prioritization Technology is supported (see Section 17.17.4). 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:

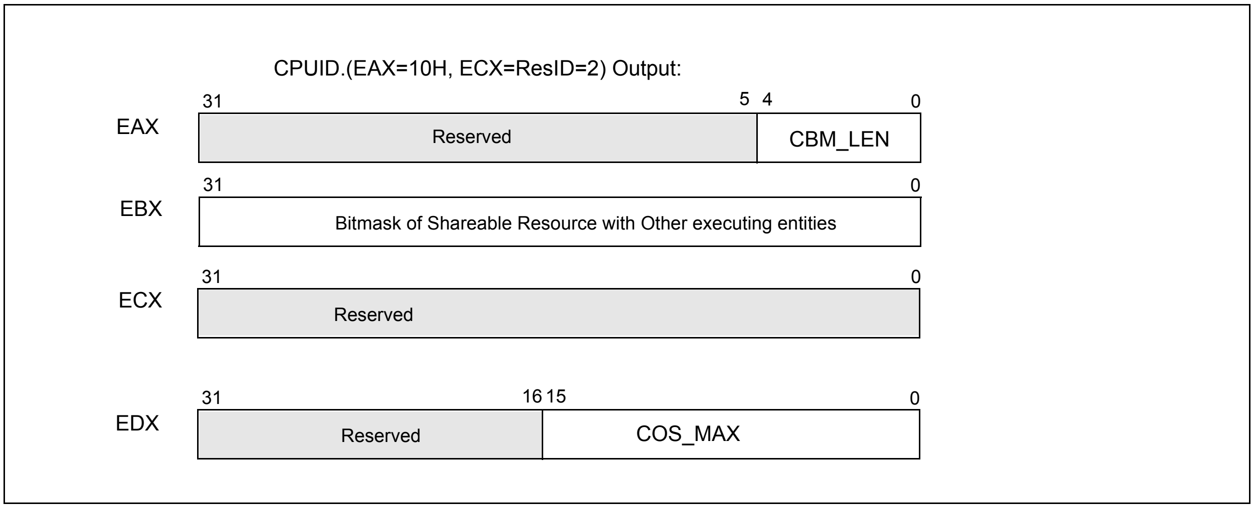

CPUID.(EAX=10H, ECX=ResID=2) Output:
EAX
31
5
4
0
Reserved
CBM_LEN
EBX
31
0
Bitmask of Shareable Resource with Other executing entities
ECX
31
0
Reserved
EDX
31
16
15
0
Reserved
COS_MAX

Figure 17-33 L2 Cache Allocation Technology

— CPUID.(EAX=10H, ECX=ResID=2):EAX[4:0] reports the length of the capacity bitmask length using minus-one notation, i.e. 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: 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.

17.17.3.3 Cache Allocation Technology: Cache Mask Configuration

After determining the length of the capacity bitmasks (CBM) and number of COS supported using CPUID (see Section 17.17.3.2), 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 MSRs is reserved for Cache Allocation Technology registers of the form IA32_resourceType_MASK_n:

- from 0C90H through 0D8FH (inclusive), providing support for multiple sub-ranges to support varying resource types. The first supported resourceType is 'L3', corresponding to the L3 cache in a platform. The MSRs range from 0C90H through 0D0FH (inclusive), enables support for up to 128 L3 CAT Classes of Service.

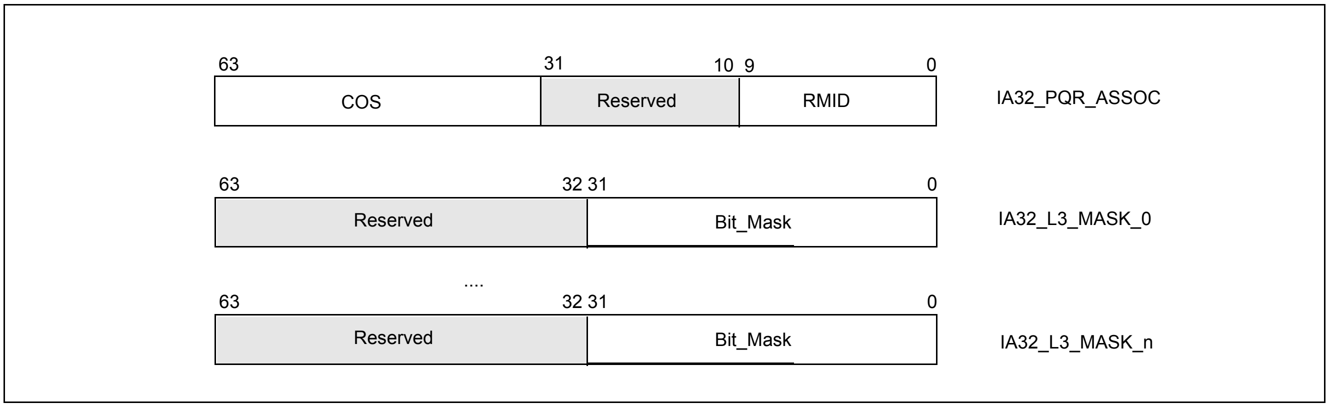


Figure 17-34 IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- 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.

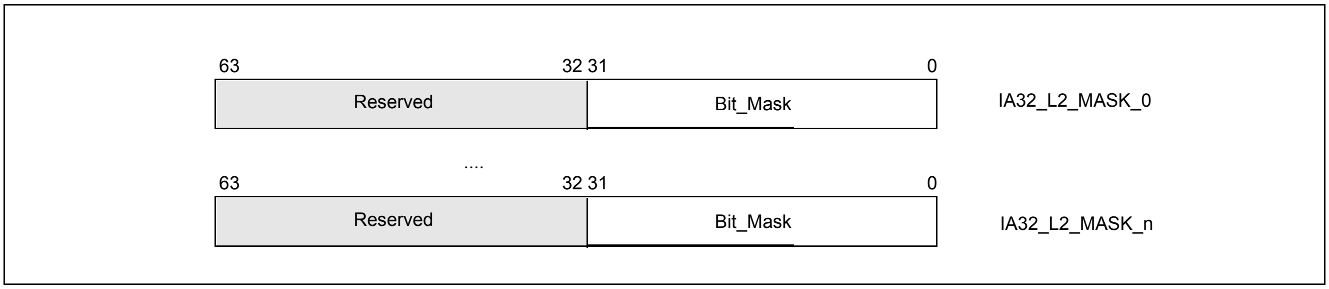


Figure 17-35 IA32_L2_MASK_n MSRs

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 Section 17.17.3.4 (Class of Service (COS) to Cache Mask Association: Common Across Allocation Features).

17.17.3.4 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.

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 17.17.5, "Cache Allocation Technology Programming Considerations" for important COS programming considerations including maximum values when using CAT and CDP.

17.17.4 Code and Data Prioritization (CDP): Enumerating and Enabling L3 CDP Technology

CDP is an extension of CAT. The presence of the 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).

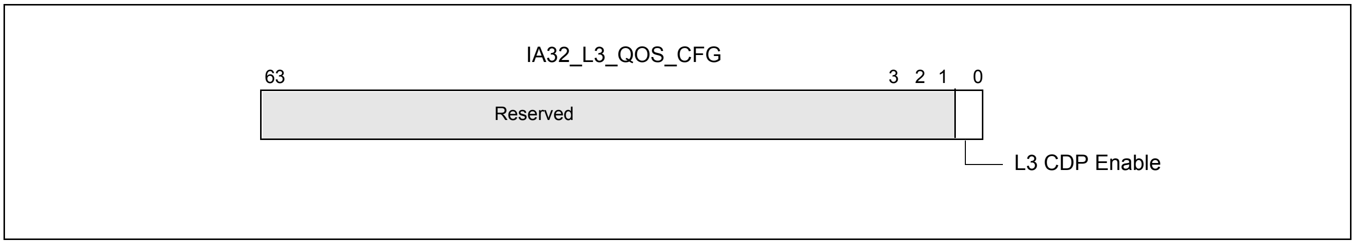


Figure 17-36 Layout of IA32_L3_QOS_CFG

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.

Before enabling or disabling CDP, software should write all 1's to all of the CAT/CDP masks to ensure proper behavior (e.g., the IA32_L3_QOS_Mask_n set of MSRs). 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.

17.17.4.1 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 17-20 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 data 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).

17.17.4.2 L3 CAT: Disabling CDP

Before enabling or disabling CDP, software should write all 1's to all of the CAT/CDP masks to ensure proper behavior (e.g., the IA32_L3_QOS_Mask_n set of MSRs).

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.

...

17.17.5.1 Cache Allocation Technology Dynamic Configuration

Both the CAT masks and CQM registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,

- Accessing a QOS mask register outside the supported COS (the max COS number is specified in CPUID.(EAX=10H, ECX=ResID):EDX[15:0]), or
- Writing a COS greater than the supported maximum (specified as the maximum value of CPUID.(EAX=10H, ECX=ResID):EDX[15:0] for all valid ResID values) is written to the IA32_PQR_ASSOC.CLOS field.

When CDP is enabled, specifying a COS value in IA32_PQR_ASSOC.COS outside of the lower half of the COS space will cause undefined performance impact to code and data fetches due to MSR space re-indexing into code/data masks when CDP is enabled.

When reading the IA32_PQR_ASSOC register the currently programmed COS on the core will be returned.

When reading an IA32_*resourceType*_MASK_*n* register the current capacity bit mask for COS 'n' will be returned.

As noted previously, software should minimize migrations of COS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if COS are migrated frequently. This is aligned with the industry standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and COS migration across processor logical threads and processor cores.

...

18. Updates to Chapter 18, Volume 3B

Change bars show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

Intel 64 and IA-32 architectures provide facilities for monitoring performance via a PMU (Performance Monitoring Unit).

18.1 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 18.3, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)." Non-architectural events for a given microarchitecture can not be enumerated using CPUID; and they are listed in Chapter 19, "Performance-Monitoring Events."

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 18.2.

See also:

— Section 18.2, "Architectural Performance Monitoring"

— Section 18.3, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)"

— Section 18.4, "Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)"

— Section 18.5, "Performance Monitoring (45 nm and 32 nm Intel® Atom™ Processors)"

— Section 18.6, "Performance Monitoring for Silvermont Microarchitecture"

— Section 18.7, "Performance Monitoring for Goldmont Microarchitecture"

— Section 18.8, "Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Nehalem"

— Section 18.8.4, "Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Westmere"

— Section 18.9, "Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Sandy Bridge"

— Section 18.9.8, "Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility"

— Section 18.10, "3rd Generation Intel® Core™ Processor Performance Monitoring Facility"

— Section 18.11, "4th Generation Intel® Core™ Processor Performance Monitoring Facility"

— Section 18.12, "Intel® Core™ M Processor Performance Monitoring Facility"

— Section 18.13, "6th Generation Intel® Core™ Processor Performance Monitoring Facility"

— Section 18.14, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)"

— Section 18.15, "Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture"

— Section 18.18, "Performance Monitoring and Dual-Core Technology"

— Section 18.19, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache"

— Section 18.21, "Performance Monitoring (P6 Family Processor)"

— Section 18.22, "Performance Monitoring (Pentium Processors)"

18.2 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 Core processors and related Intel Xeon processor families based on the Nehalem through Broadwell microarchitectures support version ID 1, 2, and 3. Intel processors based on the Skylake microarchitecture support versionID 4.

Next generation Intel Atom processors is based on the Goldmont microarchitecture. Intel processors based on the Goldmont microarchitecture support versionID 4.

...

18.2.1.1 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].
- 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.
- 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 18-1 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 18-1, 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.

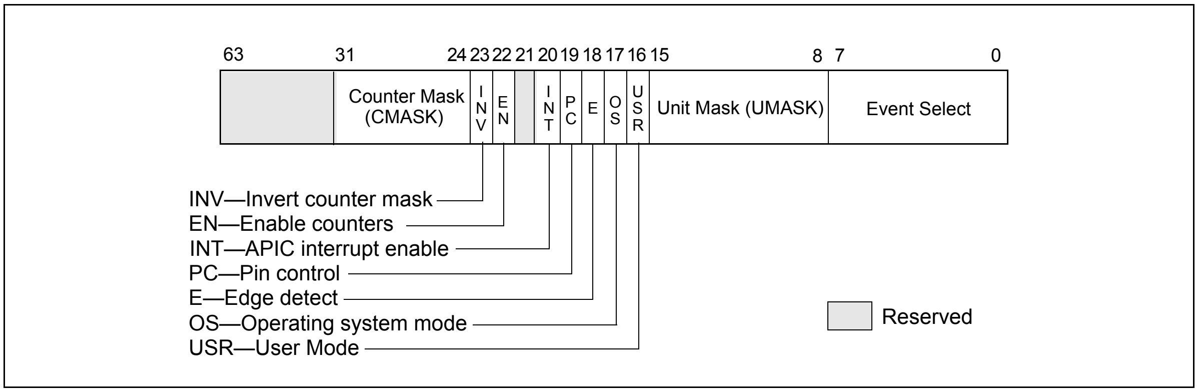


Figure 18-1 Layout of IA32_PERFEVTSELx MSRs

- **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 18-1; support for pre-defined architectural events is enumerated using CPUID.0AH:EBX. Architectural performance events available in the initial implementation are listed in Table 19-1.
- **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) —** When set, the logical processor toggles the PM*i* pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PM*i* 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.

...

18.2.2 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_PERFEVTSELx), 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 18-2 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:

- **Enable field (lowest 2 bits within each 4-bit control)** — When bit 0 is set, performance counting is

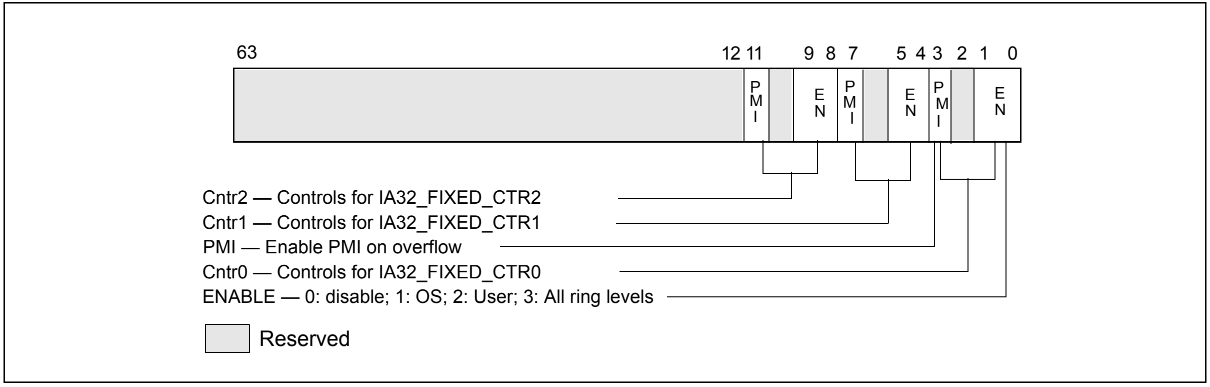

63
12
11
9
8
7
5
4
3
2
1
0
PMI
EN
PMI
EN
PMI
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow
Cntr0 — Controls for IA32_FIXED_CTR0
ENABLE — 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

Figure 18-2 Layout of IA32_FIXED_CTR_CTRL MSR

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 18-3 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.

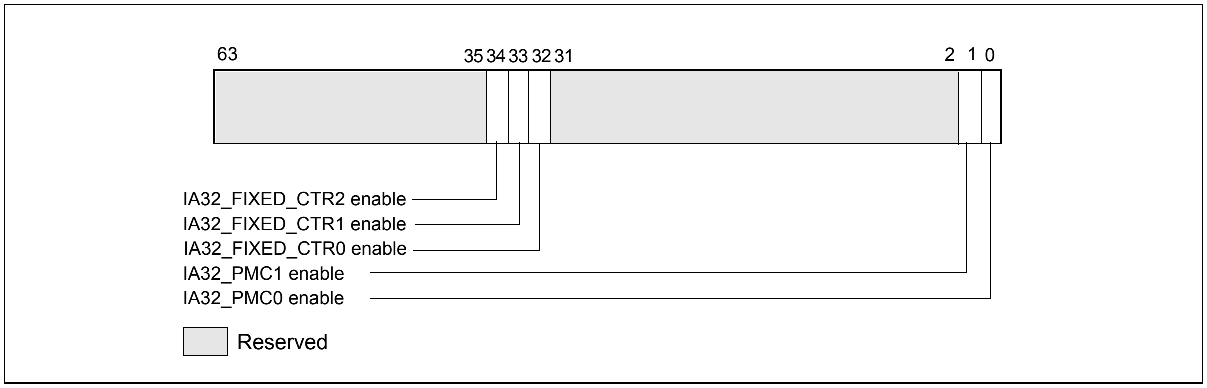


Figure 18-3 Layout of IA32_PERF_GLOBAL_CTRL MSR

The fixed-function performance counters supported by architectural performance version 2 is listed in Table 18-8, the pairing between each fixed-function performance counter to an architectural performance event is also shown.

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 18-4 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.

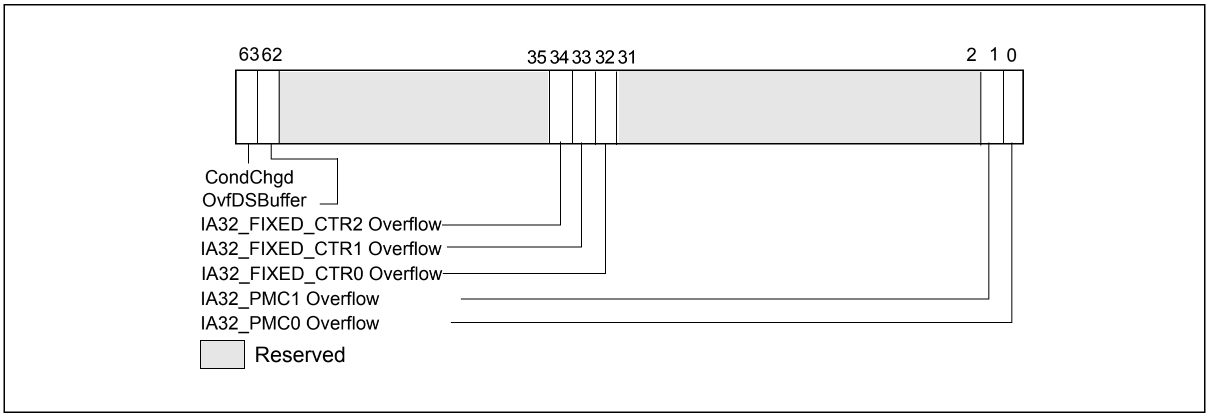


Figure 18-4 Layout of IA32_PERF_GLOBAL_STATUS MSR

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 18-5.

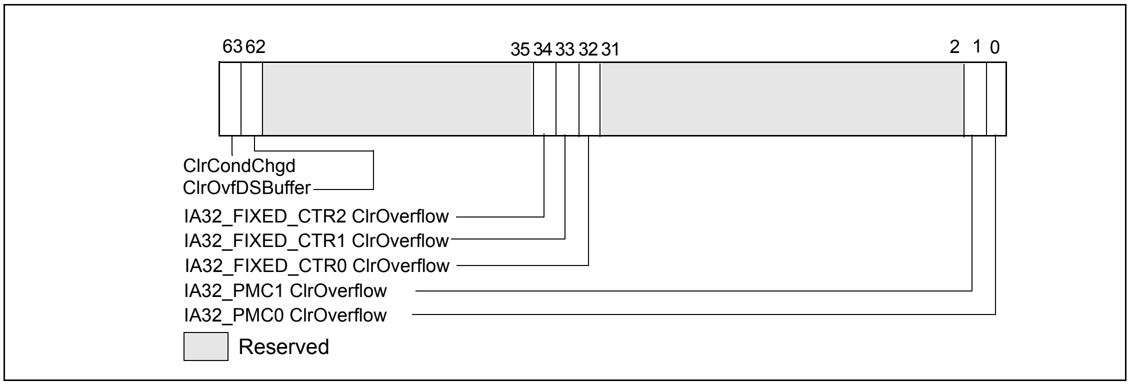


Figure 18-5 Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

...

18.2.5 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 18-49.

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
```

...

18.4.2 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 (MSR_PERF_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 (MSR_PERF_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 (MSR_PERF_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 18-15). 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 MSR_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.

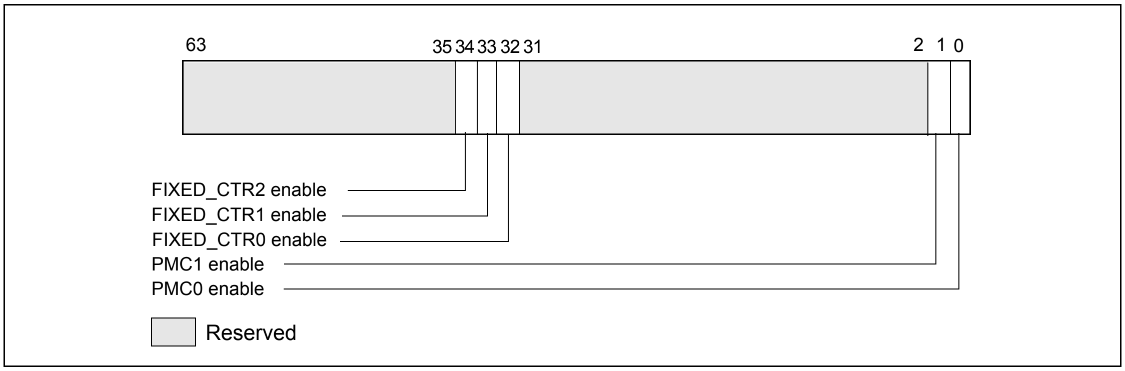


Figure 18-15 Layout of MSR_PERF_GLOBAL_CTRL MSR

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 18-16). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

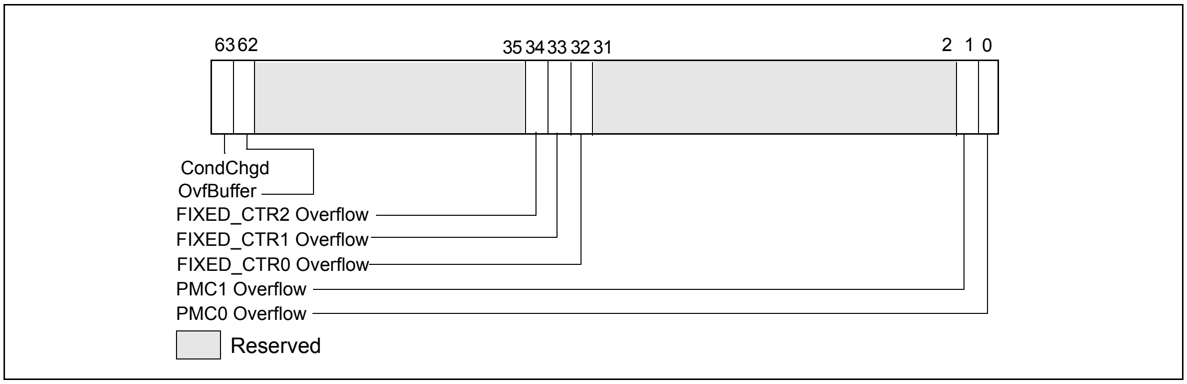


Figure 18-16 Layout of MSR_PERF_GLOBAL_STATUS MSR

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 18-17). 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.

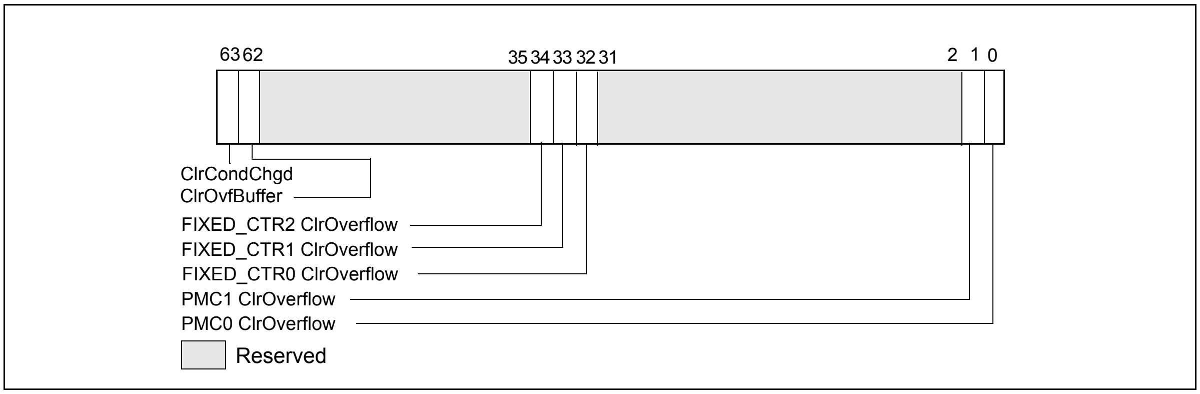


Figure 18-17 Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

...

18.4.4 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 18.4.4.2 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 18-10. The procedure for detecting availability of PEBS is the same as described in Section 18.14.7.1.

Table 18-10 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

...

18.4.4.3 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 18-11.

...

18.6.1.1 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 18.4.4).

The list of precise events supported in the Silvermont microarchitecture is shown in Table 18-12.

Table 18-12 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 18-13, and each field in the PEBS record is 64 bits long.

Table 18-13 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

18.6.2 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 18-14 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 18-14 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 18-18 and Figure 18-19. 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 18.6.3 for details.

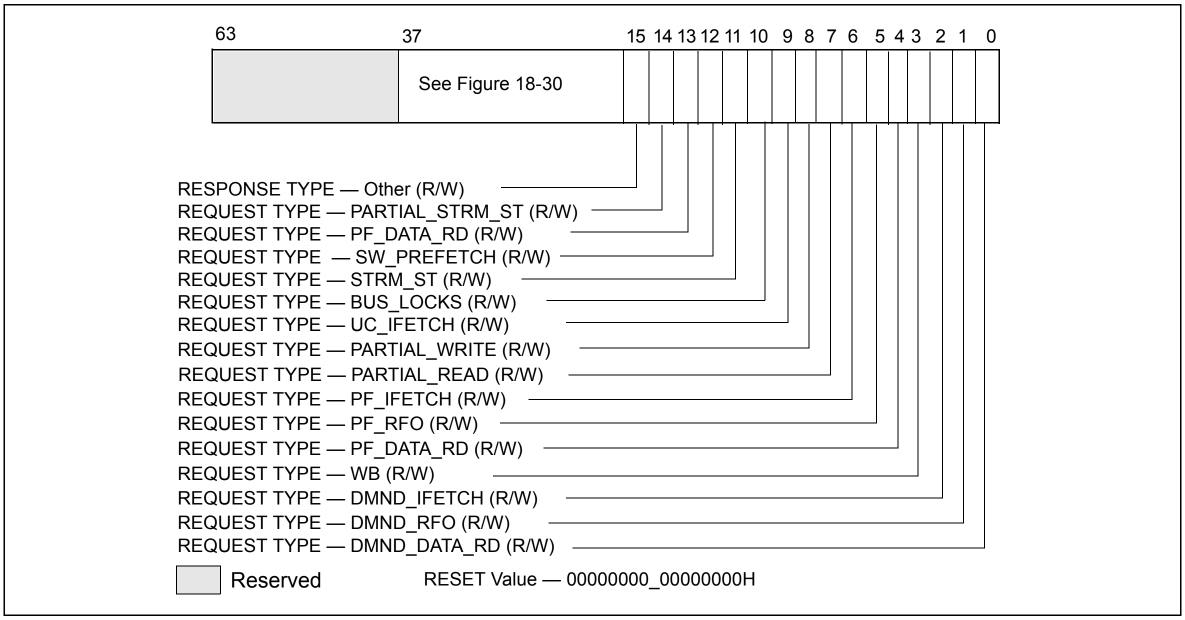

63
37
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
See Figure 18-30
RESPONSE TYPE — Other (R/W)
REQUEST TYPE — PARTIAL_STRM_ST (R/W)
REQUEST TYPE — PF_DATA_RD (R/W)
REQUEST TYPE — SW_PREFETCH (R/W)
REQUEST TYPE — STRM_ST (R/W)
REQUEST TYPE — BUS_LOCKS (R/W)
REQUEST TYPE — UC_IFETCH (R/W)
REQUEST TYPE — PARTIAL_WRITE (R/W)
REQUEST TYPE — PARTIAL_READ (R/W)
REQUEST TYPE — PF_IFETCH (R/W)
REQUEST TYPE — PF_RFO (R/W)
REQUEST TYPE — PF_DATA_RD (R/W)
REQUEST TYPE — WB (R/W)
REQUEST TYPE — DMND_IFETCH (R/W)
REQUEST TYPE — DMND_RFO (R/W)
REQUEST TYPE — DMND_DATA_RD (R/W)
Reserved
RESET Value — 00000000_00000000H

Figure 18-18 Request_Type Fields for MSR_OFFCORE_RSPx

Table 18-15 MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	(R/W). 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	(R/W). 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	(R/W). Counts the number of demand and DCU prefetch instruction cacheline reads. Does not count L2 code read prefetches.
WB	3	(R/W). Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	(R/W). Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	(R/W). Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	(R/W). Counts the number of code reads generated by L2 prefetchers.
PARTIAL_READ	7	(R/W). Counts the number of demand reads of partial cache lines (including UC and WC).
PARTIAL_WRITE	8	(R/W). Counts the number of demand RFO requests to write to partial cache lines (includes UC, WT and WP)
UC_IFETCH	9	(R/W). Counts the number of UC instruction fetches.
BUS_LOCKS	10	(R/W). Bus lock and split lock requests
STRM_ST	11	(R/W). Streaming store requests
SW_PREFETCH	12	(R/W). Counts software prefetch requests

Table 18-15 MSR_OFFCORE_RSPx Request_Type Field Definition (Contd.)

Bit Name	Offset	Description
PF_DATA_RD	13	(R/W). Counts DCU hardware prefetcher data read requests
PARTIAL_STRM_ST	14	(R/W). Streaming store requests
ANY	15	(R/W). Any request that crosses IDI, including I/O.

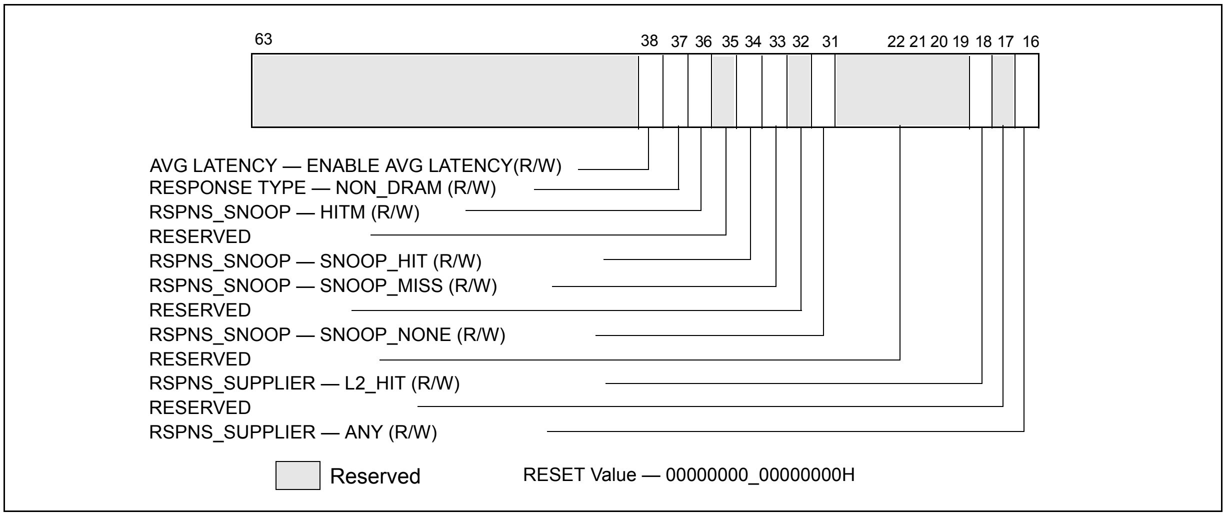

63
38 37 36 35 34 33 32 31
22 21 20 19 18 17 16
AVG LATENCY — ENABLE AVG LATENCY(R/W)
RESPONSE TYPE — NON_DRAM (R/W)
RSPNS_SNOOP — HITM (R/W)
RESERVED
RSPNS_SNOOP — SNOOP_HIT (R/W)
RSPNS_SNOOP — SNOOP_MISS (R/W)
RESERVED
RSPNS_SNOOP — SNOOP_NONE (R/W)
RESERVED
RSPNS_SUPPLIER — L2_HIT (R/W)
RESERVED
RSPNS_SUPPLIER — ANY (R/W)
Reserved
RESET Value — 00000000_00000000H

Figure 18-19 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 18-15) and a valid response type pattern (Table 18-16, Table 18-17). 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 18-16 MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	ANY_RESPONSE	16	(R/W). Catch all value for any response types.
Supplier Info	Reserved	17	Reserved
	L2_HIT	18	(R/W). 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 18-17   MSR_OFFCORE_RSPx Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	(R/W). No details on snoop-related information
	Reserved	32	Reserved
	SNOOP_MISS	33	(R/W). Counts the number of snoop misses when L2 misses
	SNOOP_HIT	34	(R/W). Counts the number of snoops hit in the other module where no modified copies were found
	Reserved	35	Reserved
	HITM	36	(R/W). Counts the number of snoops hit in the other module where modified copies were found in other core's L1 cache.
	NON_DRAM	37	(R/W). Target was non-DRAM system address. This includes MMIO transactions.
	AVG_LATENCY	38	(R/W). 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.

...

18.6.3 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, white 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.

18.7 PERFORMANCE MONITORING FOR GOLDMONT MICROARCHITECTURE

Next generation Intel Atom processors are based on the Goldmont microarchitecture. They report architectural performance monitoring versionID = 4 (see Section 18.2.4) and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 18.2.4.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 18-6 and described in Section 18.2.1.1 and Section 18.2.3. Architectural and non-architectural performance monitoring events in the Goldmont microarchitecture ignore the AnyThread qualification regardless of its setting in the IA32_PERFEVTSELx MSR.

The core PMU's capability is similar to that of the Silvermont microarchitecture described in Section 18.6 , with some differences and enhancements summarized in Table 18-18.

Table 18-18 Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	The Goldmont microarchitecture	The Silvermont microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to enumerate # of counters.
# of general-purpose counters per core	4	2	
Counter width (R,W)	R:48, W: 32/48	R:40, W:32	See Section 18.2.2.
Architectural Performance Monitoring version ID	4	3	Use CPUID to enumerate # of counters.
PMI Overhead Mitigation	▪ Freeze_Perfmon_on_PMI with streamlined semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling. ▪ Freeze_while_SMM.	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR 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	▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_STATUS_RESET ▪ Set via IA32_PERF_GLOBAL_STATUS_SET	▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_OVF_CTRL	See Section 18.2.4.
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/ Overhead/Interference	▪ Individual counter overflow ▪ PEBS buffer overflow ▪ ToPA buffer overflow ▪ CTR_Frz, LBR_Frz	▪ Individual counter overflow ▪ PEBS buffer overflow	See Section 18.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	▪ CTR_Frz, ▪ LBR_Frz	No	See Section 18.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	No	See Section 18.2.4.3.
Processor Event Based Sampling (PEBS) Events	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 18-19.	See Section 18.6.1.1. General-Purpose Counter 0 only. Only supports precise events (see Table 18-12).	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 18-20; enhanced fields at offsets 90H- 98H; and TSC record field at C0H.	Table 18-13.	
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.

18.7.1 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 as on the Silvermont microarchitecture. They can capture precise eventingIP associated with a retired instruction that caused the event. The PEBS record provides architectural state of the instruction executed after the instruction that caused the event (See Section 18.4.4 and Section 17.4.9). The PEBS record also provides architectural state of the processor after the instruction that caused the event completes. The list of precise events supported in the Goldmont microarchitecture is shown in Table 18-19.

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. Other examples of a non-precise events are CPU_CLK_UNHALTED.CORE_P and HARDWARE_INTERRUPTS.RECEIVED. There may be many instructions in various stages of execution, multiple or zero instructions being retired each cycle as CPU_CLK_UNHALTED.CORE_P increments. HARDWARE_INTERRUPTS.RECEIVED increments independent of any instructions being executed. The PEBS record will be generated at the next opportunity, capturing the processor state when the machine received the interrupt, even if interrupts are masked. The PEBS facility thus allows for identification of the instructions which were executing when the event overflowed.

After generating a record, 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 18-19 Precise Events Supported by the Goldmont Microarchitecture

Event Name	Event Select	Sub-event	UMask
LD_BLOCKS	03H	DATA_UNKNOWN	01H
		STORE_FORWARD	02H
		4K_ALIAS	04H
		UTLB_MISS	08H
		ALL_BLOCK	10H
MISALIGN_MEM_REF	13H	LOAD_PAGE_SPLIT	02H
		STORE_PAGE_SPLIT	04H
INST_RETIRED	C0H	ANY	00H
UOPS_RETITRED	C2H	ANY	00H
		LD_SPLITSMS	01H
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH

Table 18-19 Precise Events Supported by the Goldmont Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
		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	D0H	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 Intel Silvermont microarchitecture is shown in Table 18-13, and each field in the PEBS record is 64 bits long.

Table 18-20 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

Table 18-20 PEBS Record Format for the Goldmont Microarchitecture

Byte Offset	Field	Byte Offset	Field
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 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.

18.7.1.1 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. The fields are present in the record but are reserved.

For Goldmont, 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).

18.7.1.2 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 18.9.4.4

In the Goldmont microarchitecture, the mechanism applies to all precise events including INST_RETIRED. However, the Reduced Skid mechanism is disabled for any counter when the INV, ANY, E, or CMASK fields are set.

To ensure the Reduced Skid mechanism operates correctly, disable PEBS via the IA32_PEBS_ENABLE or IA32_PERF_GLOBAL_CTRL MSRs before writing to the configuration registers (IA32_PERFEVTSELx) or to the counters (IA32_PMCx and IA32_A_PMCx).

18.7.1.3 Enhancements to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62]

In addition to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62] being set when PEBS_Index reaches the PEBS_Interrupt_Theshold, the bit is also set when PEBS_Index is out of bounds. That is, the bit will be set when PEBS_Index < PEBS_Buffer_Base or PEBS_Index > PEBS_Absolute_Maximum. Note that when an out of bound condition is encountered, the overflow bits in IA32_PERF_GLOBAL_STATUS will be cleared according to Applicable Counters, however the IA32_PMCx values will not be reloaded with the Reset values stored in the DS_AREA.

18.7.2 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 18-14 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 18-21.
- Bits 30:16 specifies common supplier information or an L2 Hit, and is described in Table 18-16.
- If L2 misses, then Bits 37:31 can be used to specify snoop response information and is described in Table 18-22.
- 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 18.6.3 for details.

Table 18-21 MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	(R/W) Counts cacheline read requests due to demand reads (excludes prefetches).
DEMAND_RFO	1	(R/W) Counts cacheline read for ownership (RFO) requests due to demand writes (excludes prefetches).
DEMAND_CODE_RD	2	(R/W) Counts demand instruction cacheline and I-side prefetch requests that miss the instruction cache.
COREWB	3	(R/W) Counts writeback transactions caused by L1 or L2 cache evictions.
PF_L2_DATA_RD	4	(R/W) Counts data cacheline reads generated by hardware L2 cache prefetcher.
PF_L2_RFO	5	(R/W) Counts reads for ownership (RFO) requests generated by L2 prefetcher.
Reserved	6	Reserved.
PARTIAL_READS	7	(R/W) Counts demand data partial reads, including data in uncacheable (UC) or uncacheable (WC) write combining memory types.
PARTIAL_WRITES	8	(R/W) Counts partial writes, including uncacheable (UC), write through (WT) and write protected (WP) memory type writes.
UC_CODE_READS	9	(R/W) Counts code reads in uncacheable (UC) memory region.
BUS_LOCKS	10	(R/W) Counts bus lock and split lock requests.
FULL_STREAMING_STORES	11	(R/W) Counts full cacheline writes due to streaming stores.
SW_PREFETCH	12	(R/W) Counts cacheline requests due to software prefetch instructions.
PF_L1_DATA_RD	13	(R/W) Counts data cacheline reads generated by hardware L1 data cache prefetcher.
PARTIAL_STREAMING_STORES	14	(R/W) Counts partial cacheline writes due to streaming stores.
ANY_REQUEST	15	(R/W) Counts requests to the uncore subsystem.

To properly program this extra register, software must set at least one request type bit (Table 18-15) and a valid response type pattern (either Table 18-16 or Table 18-22). 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 18-22 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	(R/W). 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	(R/W) A snoop hit in the other processor module, but no data forwarding is required.
	Reserved	35	Reserved
	L2_MISS.HITM_OTHER_CORE	36	(R/W) 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	(R/W) Target was a non-DRAM system address. This includes MMIO transactions.
Outstanding requests[1]	OUTSTANDING	38	(R/W) Counts weighted cycles of outstanding offcore requests of the request type specified in bits 15:0, from the time the XQ receives the request and any response is received. Bits 37:16 must be set to 0. This bit is only available in MSR_OFFCORE_RESP0.

NOTES:

[1] See Section 18.6.3, "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 'OR' (L2 Hit)] 'XOR' (Snoop Info Bits) 'XOR' (Avg Latency)

18.7.3 Average Offcore Request Latency Measurement

In Goldmont microarachitecture, measurement of average latency of offcore transaction requests is the same as described in Section 18.6.3.

...

18.8.1 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 Intel microarchitecture code name Nehalem 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 Intel microarchitecture code name Nehalem. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is

reported for retired demand load operations and in core cycles (it accounts for re-dispatches). This facility is used in conjunction with the PEBS facility.

- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.

18.8.1.1 Processor Event Based Sampling (PEBS)

All four 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 Intel microarchitecture code name Nehalem is shown in Figure 18-21.

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.

...

18.8.2.2 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 corresponding MSR_UNCORE_PerfEvtSelx and counting must be enabled using the respective EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL. Figure 18-28 shows the layout of MSR_UNCORE_PERFEVTSELx.

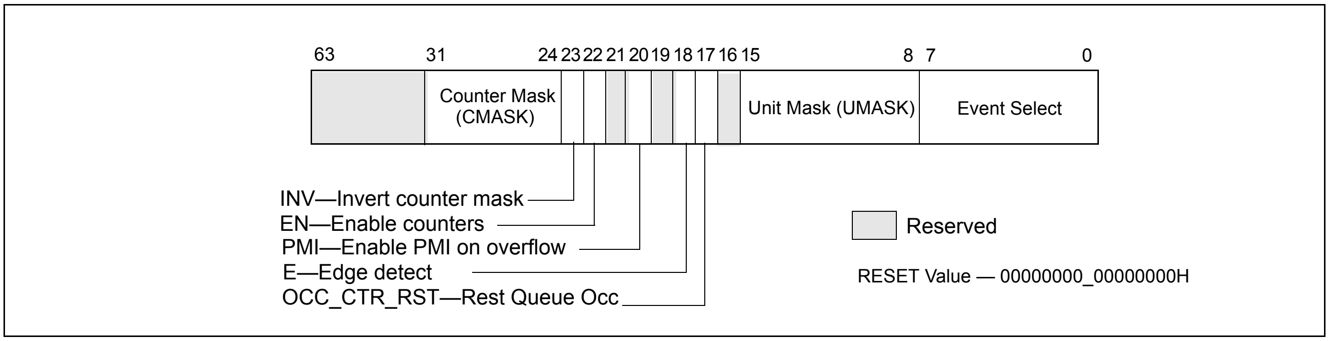


Figure 18-28 Layout of MSR_UNCORE_PERFEVTSELx MSRs

- 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 18-29 shows the layout of MSR_UNCORE_FIXED_CTR_CTRL.

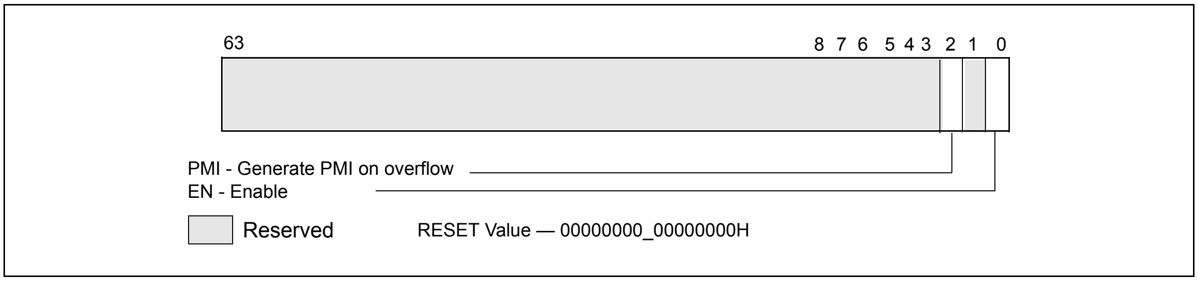

63
8 7 6 5 4 3 2 1 0
PMI - Generate PMI on overflow
EN - Enable
Reserved
RESET Value — 00000000_00000000H

Figure 18-29 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_PerfCntr) and the fixed-function counter (MSR_UNCORE_FixedCntr0) 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.

...

18.9 PERFORMANCE MONITORING FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel® Xeon® processor E3-1200 family are based on Intel microarchitecture code name Sandy Bridge; 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 18.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 18.2.3.

The core PMU's capability is similar to those described in Section 18.8.1 and Section 18.8.4, with some differences and enhancements relative to Intel microarchitecture code name Westmere summarized in Table 18-30.

Table 18-30 Core PMU Comparison

Box	Intel® microarchitecture code name Sandy Bridge	Intel® microarchitecture code name Westmere	Comment
# of Fixed counters per thread	3	3	Use CPUID to enumerate # of counters.
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48, W: 32/48	R:48, W:32	See Section 18.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to enumerate # of counters.
PMI Overhead Mitigation	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling. ▪ Freeze_while_SMM.	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling. ▪ Freeze_while_SMM.	See Section 17.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 18-32.	See Table 18-10.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 18.9.4.2; ▪ Data source encoding ▪ STLB miss encoding ▪ Lock transaction encoding	Data source encoding	
PEBS-Precise Store	Section 18.9.4.3	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.

18.9.1 Global Counter Control Facilities In Intel® Microarchitecture Code Name Sandy Bridge

The number of general-purpose performance counters visible to a logical processor can vary across Processors based on Intel microarchitecture code name Sandy Bridge. Software must use CPUID to determine the number performance counters/event select registers (See Section 18.2.1.1).

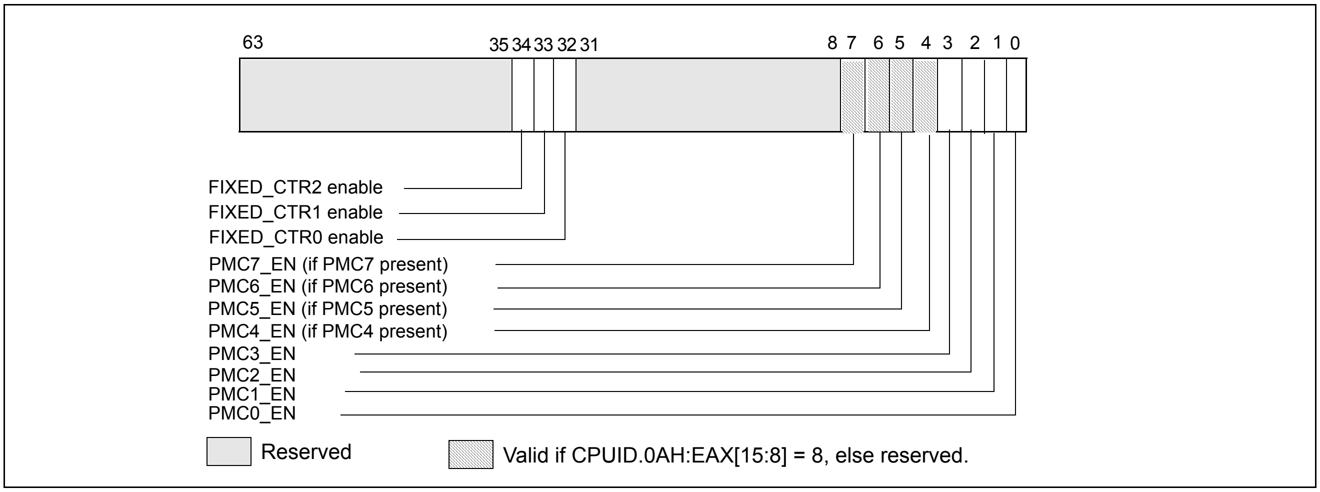

63
35 34 33 32 31
8 7 6 5 4 3 2 1 0
FIXED_CTR2 enable
FIXED_CTR1 enable
FIXED_CTR0 enable
PMC7_EN (if PMC7 present)
PMC6_EN (if PMC6 present)
PMC5_EN (if PMC5 present)
PMC4_EN (if PMC4 present)
PMC3_EN
PMC2_EN
PMC1_EN
PMC0_EN
Reserved
Valid if CPUID.0AH:EAX[15:8] = 8, else reserved.

Figure 18-32 IA32_PERF_GLOBAL_CTRL MSR in Intel® Microarchitecture Code Name Sandy Bridge

Figure 18-15 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_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.
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 18-33). A value of 1 in each bit of the PMCx_OVF field indicates an overflow condition has occurred in the associated counter.

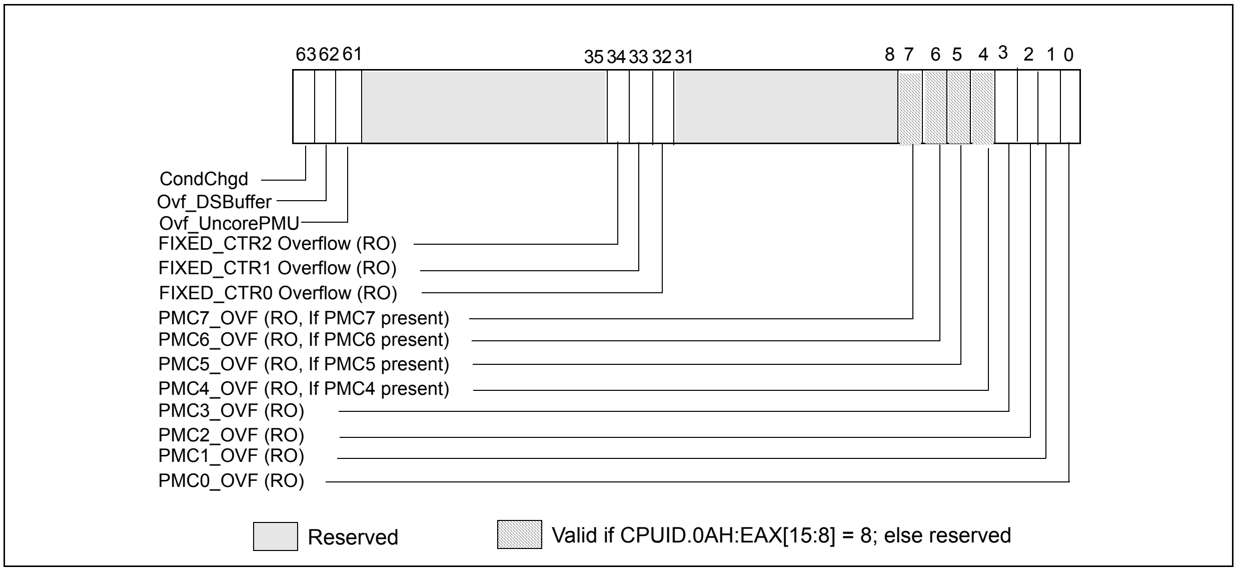


Figure 18-33 IA32_PERF_GLOBAL_STATUS MSR in Intel® Microarchitecture Code Name Sandy Bridge

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 18-34). 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

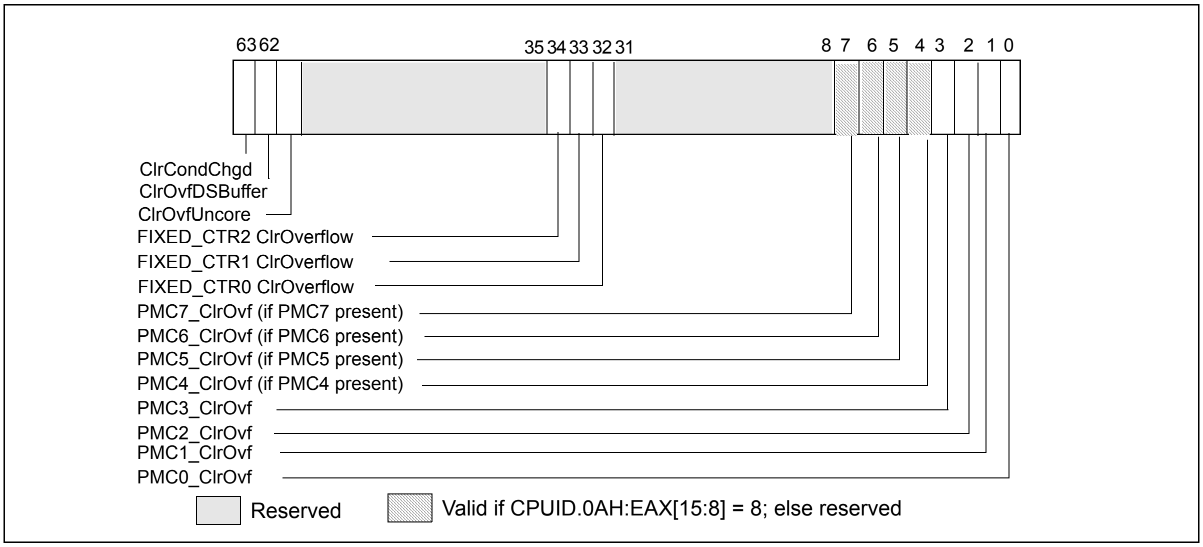


Figure 18-34 IA32_PERF_GLOBAL_OVF_CTRL MSR in Intel microarchitecture code name Sandy Bridge

...

18.11 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 18.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 18.2.3.

The core PMU's capability is similar to those described in Section 18.9 through Section 18.9.5, with some differences and enhancements summarized in Table 18-42. 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 18.11.5. For details of Intel TSX, see Chapter 15, "Programming with Intel® Transactional Synchronization Extensions" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Table 18-42 Core PMU Comparison

Box	Intel® microarchitecture code name Haswell	Intel® microarchitecture code name Sandy Bridge	Comment
# of Fixed counters per thread	3	3	
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 18.2.2.
# 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 enumerate # of counters.
PMI Overhead Mitigation	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling. ▪ Freeze_while_SMM.	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling. ▪ Freeze_while_SMM.	See Section 17.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 18-32 and Section 18.11.5.1.	See Table 18-32.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 18.9.4.2.	See Section 18.9.4.2.	
PEBS-Precise Store	No, replaced by Data Address profiling.	Section 18.9.4.3	
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 17.9.	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 18.11.5.	no	

18.11.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 4th Generation Intel Core processor is similar to those in processors based on Intel microarchitecture code name Sandy Bridge, with several enhanced features. The key components and differences of PEBS facility relative to Intel microarchitecture code name Sandy Bridge is summarized in Table 18-43.

Table 18-43 PEBS Facility Comparison

Box	Intel® microarchitecture code name Haswell	Intel® microarchitecture code name Sandy Bridge	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 18.8.1.1	Section 18.8.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 18-21	Figure 18-35	

Table 18-43 PEBS Facility Comparison

Box	Intel® microarchitecture code name Haswell	Intel® microarchitecture code name Sandy Bridge	Comment
PEBS record layout	Table 18-44; enhanced fields at offsets 98H, A0H, A8H, B0H.	Table 18-23; enhanced fields at offsets 98H, A0H, A8H.	
Precise Events	See Table 18-32.	See Table 18-32.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 18-33.	Table 18-33	
PEBS-Precise Store	No, replaced by data address profiling.	Yes; see Section 18.9.4.3.	
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.

...

18.12 INTEL® CORE™ M PROCESSOR PERFORMANCE MONITORING FACILITY

The Intel® Core™ M processor and the 5th Generation Intel Core processor families are based on the Broadwell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 18.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 18.2.3.

The core PMU has the same capability as those described in Section 18.11. 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.

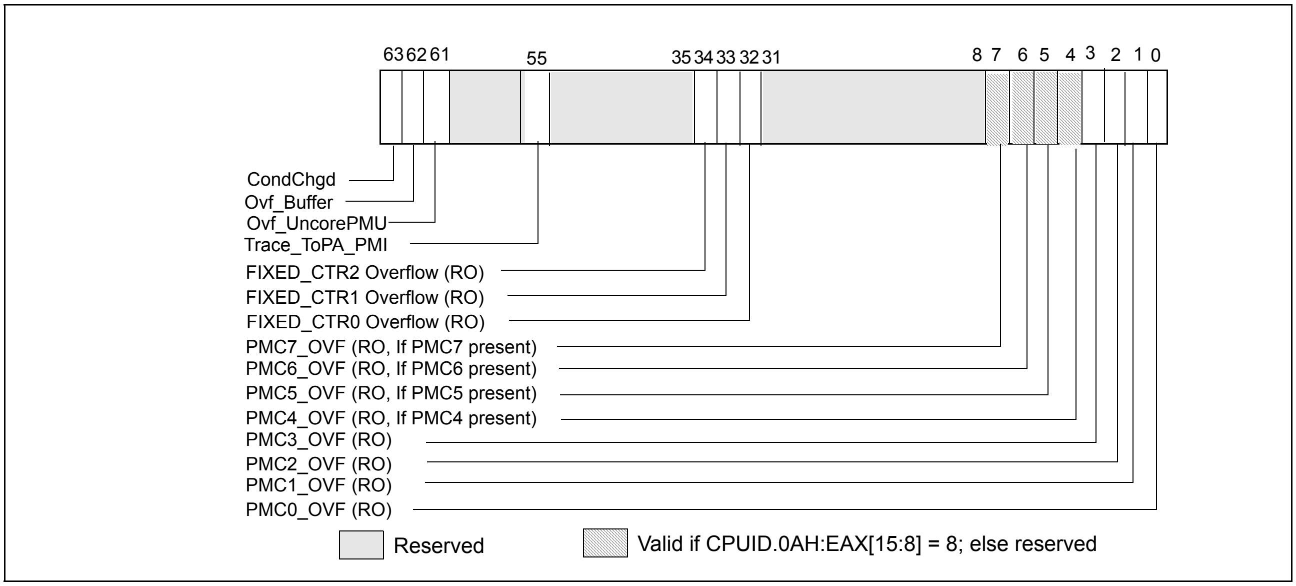


Figure 18-41 IA32_PERF_GLOBAL_STATUS MSR in Broadwell Microarchitecture

Details of Intel Processor Trace is described in Chapter 36, “Intel® Processor Trace”. IA32_PERF_GLOBAL_OVF_CTRL MSR provide a corresponding reset control bit.

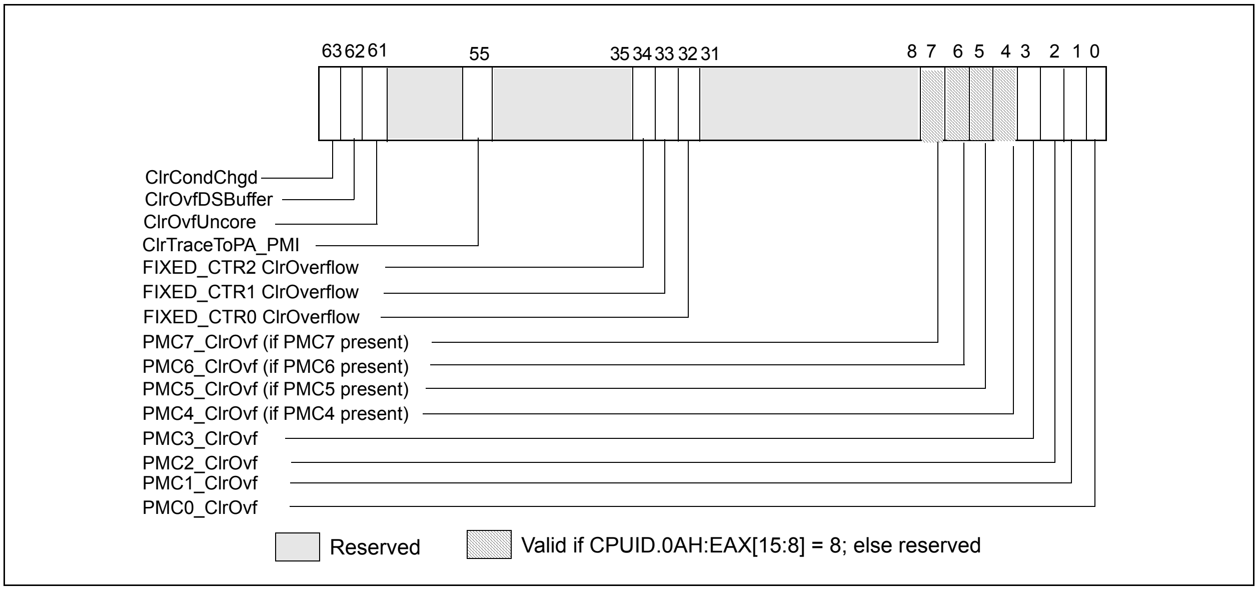


Figure 18-42 IA32_PERF_GLOBAL_OVF_CTRL MSR in Broadwell microarchitecture

The specifics of non-architectural performance events are listed in Chapter 19, “Performance Monitoring Events”.

18.13 6TH GENERATION INTEL® CORE™ PROCESSOR PERFORMANCE MONITORING FACILITY

The 6th generation Intel® Core™ processor is based on the Skylake microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 4 (see Section 18.2.4) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 4 capabilities are described in Section 18.2.4.

The core PMU's capability is similar to those described in Section 18.9 through Section 18.9.5, with some differences and enhancements summarized in Table 18-42. 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 18.11.5. For details of Intel TSX, see Chapter 15, "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 43, "Enclave Code Debug and Profiling".

Table 18-53 Core PMU Comparison

Box	Intel® microarchitecture code name Skylake	Intel® microarchitecture code name Haswell and Broadwell	Comment
# of Fixed counters per thread	3	3	
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 18.2.2.
# 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)	CPUID enumerates # of counters.
Architectural Perfmon version	4	3	See Section 18.2.4
PMI Overhead Mitigation	▪ Freeze_Perfmon_on_PMI with streamlined semantics. ▪ Freeze_on_LBR with streamlined semantics. ▪ Freeze_while_SMM.	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR 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	▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_STATUS_RESET ▪ Set via IA32_PERF_GLOBAL_STATUS_SET	▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_OVF_CTRL	See Section 18.2.4.
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/ Overhead/Interference	▪ Individual counter overflow ▪ PEBS buffer overflow ▪ ToPA buffer overflow ▪ CTR_Frz, LBR_Frz, ASCI	▪ Individual counter overflow ▪ PEBS buffer overflow ▪ ToPA buffer overflow (applicable to Broadwell microarchitecture)	See Section 18.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	▪ CTR_Frz, ▪ LBR_Frz	NA	See Section 18.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	NA	See Section 18.2.4.3.

Table 18-53 Core PMU Comparison (Contd.)

Box	Intel® microarchitecture code name Skylake	Intel® microarchitecture code name Haswell and Broadwell	Comment
Precise Events	See Table 18-56.	See Table 18-32.	IA32_PMC4-PMC7 do not support PEBS.
PEBS for front end events	See Section 18.13.1.4;	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 17.10
LBR Timing	yes	no	Section 17.10.1
Call Stack Profiling	yes, see Section 17.9	yes, see Section 17.9	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 18.11.5;	See Section 18.11.5;	

18.13.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 6th Generation Intel Core processor 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 18-54.

Table 18-54 PEBS Facility Comparison

Box	Intel® microarchitecture code name Skylake	Intel® microarchitecture code name Haswell and Broadwell	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 18.8.1.1	Section 18.8.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 18-21	Figure 18-21	
PEBS-EventingIP	yes	yes	
PEBS record format encoding	0011b	0010b	
PEBS record layout	Table 18-55; enhanced fields at offsets 98H- B8H; and TSC record field at C0H.	Table 18-44; 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 18-56.	See Table 18-32.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-PDIR	yes	yes	IA32_PMC1 only.
PEBS-Load Latency	See Section 18.9.4.2.	See Section 18.9.4.2.	
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.

NOTE

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.

...

18.13.1.2 PEBS Events

The list of precise events supported for PEBS in the Skylake microarchitecture is shown in Table 18-56.

Table 18-56 Precise Events for the Skylake Microarchitecture

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 18-56 Precise Events for the Skylake Microarchitecture (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 18.13.2
4. Instruction with at least one load uop experiencing the condition specified in the UMask.

...

18.14 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 RDPMC instructions is unchanged, the setup mechanism and MSR layouts are incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMC 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 18-63 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 are listed in Chapter 19, “Performance-Monitoring Events.”

Table 18-63 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_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_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

Table 18-63 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_FLAME_COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAAT_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_SAAT_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_SAAT_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_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0[1] MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0[1] MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1[1] MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER3	15	30FH	MSR_IQ_CCCR3	36FH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1[1] MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0[1] MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH

Table 18-63 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_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1[1] MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 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 (see Table 19-28) are events that occur any time during instruction execution (such as bus transactions or cache transactions).
- At-retirement events (see Table 19-29) 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 μops 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 MSRs and data structures used for performance monitoring in the Pentium 4 and Intel Xeon processors.

...

18.14.2 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 18-63). 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 18.14.5.6, "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 18-44). The RDPMC instruction is intended to allow reading of either the full counter-width (40-bits) or 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.

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.

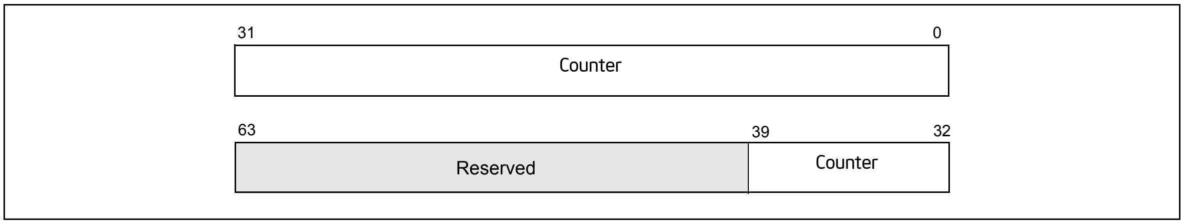


Figure 18-44 Performance Counter (Pentium 4 and Intel Xeon Processors)

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.

18.14.3 CCCR MSRs

Each of the 18 performance counters has one CCCR MSR associated with it (see Table 18-63). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 18-45 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 18.14.5.2, “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 18.14.5.2, “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.

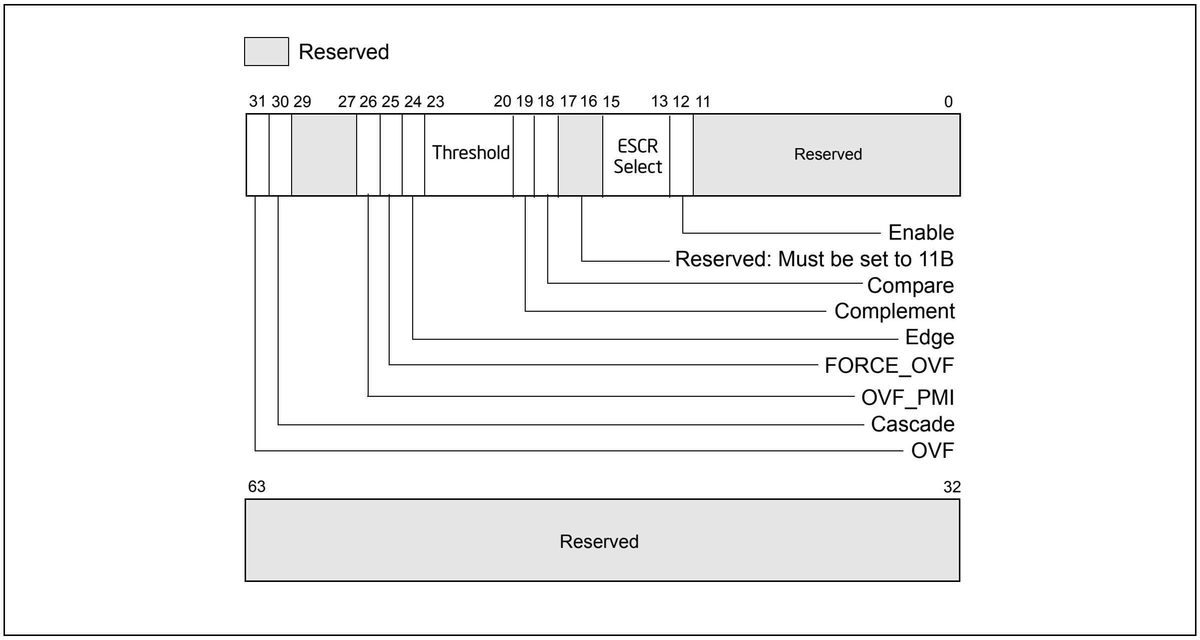


Figure 18-45 Counter Configuration Control Register (CCCR)

- **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 18.14.2, "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 18.14.5, "Programming the Performance Counters for Non-Retirement Events."

18.14.4 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 18.14.7, "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."

...

18.14.5.1 Selecting Events to Count

Table 19-29 in Chapter 19 lists a set of at-retirement events for processors based on Intel NetBurst microarchitecture. For each event listed in Table 19-29, setup information is provided. Table 18-64 gives an example of one of the events.

Table 18-64 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]
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	Requires Additional MSRs for Tagging	No	

For Table 19-28 and Table 19-29, Chapter 19, the name of the event is listed in the Event Name column and parameters that define the event and other information are listed in the Event Parameters column. The Parameter Value and Description columns give specific parameters for the event and additional description information. Entries in the Event Parameters column 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 18-63 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 18-63.
- **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 listed in Table 19-29.)
- **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 listed in Table 19-29.)

NOTE

The performance-monitoring events listed in Chapter 19, "Performance-Monitoring Events," 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.

Using information in Table 19-28, Chapter 19, 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 18-63.
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 18.14.5.2, "Filtering Events."

...

18.14.5.4 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 18.14.2, "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 18.14.5.3, "Starting Event Counting."

7. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 18-63 used as an operand.

This setup procedure is continued in the next section, Section 18.14.5.5, "Halting Event Counting."

...

18.14.5.7 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 18-65.

Table 18-65 CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1|2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INTO0	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INTO3	Allow counter 5 to cascade into counter 3
MSR_IQ_CCCR4:11	CASCNT5INTO4	Allow counter 5 to cascade into counter 4
MSR_IQ_CCCR5:11	CASCNT4INTO5	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 CASCNTxINTOy bits is used.

...

18.14.6 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."

Tables 19-29 through 19-33 list 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 μops 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 μops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and μops 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 in Table 19-29) can count instructions or μops that are retired based on the characterization of bogus" versus non-bogus.
- **Tagging —** Tagging is a means of marking μops 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 μop and a direct count of the event would not provide an indication of how many μops encountered that event.

  The tagging mechanisms allow a μop 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 μop, rather than once per event. For example, a μop may encounter a cache miss more than once during its life time, but a "Miss Retired" metric (that counts the number of retired μops that encountered a cache miss) will increment only once for that μop. 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 μops 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, μops must be reissued. The mechanism that the Pentium 4 and Intel Xeon processors use for this reissuing of μops 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 μops before they begin and are costly.

18.14.6.1 Using At-Retirement Counting

Processors based on Intel NetBurst microarchitecture allow counting both events and μops that encountered a specified event. For a subset of the at-retirement events listed in Table 19-29, a μop 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 μops tagged with that mechanism:

- **Front-end tagging —** This mechanism pertains to the tagging of μops 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 μops that encountered execution events (for example, instruction types) and are counted with the Execution_Event event.
- **Replay tagging —** This mechanism pertains to tagging of μops 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 μop that has been tagged using one mechanism will not be detected with another mechanism's tagged-μop detector. For example, if μops are tagged using the front-end tagging mechanisms, the Replay_event will not count those as tagged μops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of μops 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 μops that cannot be tagged, including I/O, uncacheable and locked accesses, returns, and far transfers.

Table 19-29 lists the 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 μop and count tagged μops.

...

18.14.6.3 Tagging Mechanism For Execution_event

Table 19-29 describes the Execution_event and Table 19-32 describes metrics that are used to set up an Execution_event count.

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 μops 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 μop. 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 μops. 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. This mechanism is summarized in the Table 19-32 metrics that are supported by the execution tagging mechanism. 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 18.14.7.3, "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.

18.14.6.4 Tagging Mechanism for Replay_event

Table 19-29 describes the Replay_event and Table 19-33 describes metrics that are used to set up an Replay_event count.

The replay mechanism enables tagging of μops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of μop 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 Table 19-33 lists the metrics that are support the replay tagging mechanism and the at-retirement events that use the replay tagging mechanism, and specifies how the appropriate MSRs need to be configured. 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 (see Table 19-29) be used to count the tagged μops.

18.14.7 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 18.4.4).

...

19. Updates to Chapter 19, Volume 3B

Change bars show changes to Chapter 19 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

This chapter lists the performance-monitoring events that can be monitored with the Intel 64 or IA-32 processors. The ability to monitor performance events and the events that can be monitored in these processors are mostly model-specific, except for architectural performance events, described in Section 19.1.

Non-architectural performance events (i.e. model-specific events) are listed for each generation of microarchitecture:

- Section 19.2 - Processors based on Skylake microarchitecture
- Section 19.3 - Processors based on Broadwell microarchitecture
- Section 19.4 - Processors based on Haswell microarchitecture
- Section 19.4.1 - Processors based on Haswell-E microarchitecture
- Section 19.5 - Processors based on Ivy Bridge microarchitecture
- Section 19.5.1 - Processors based on Ivy Bridge-E microarchitecture

- Section 19.6 - Processors based on Sandy Bridge microarchitecture
- Section 19.7 - Processors based on Intel® microarchitecture code name Nehalem
- Section 19.8 - Processors based on Intel® microarchitecture code name Westmere
- Section 19.9 - Processors based on Enhanced Intel® Core™ microarchitecture
- Section 19.10 - Processors based on Intel® Core™ microarchitecture
- Section 19.11 - Processors based on the Goldmont microarchitecture
- Section 19.12 - Processors based on the Silvermont microarchitecture
- Section 19.12.1 - Processors based on the Airmont microarchitecture
- Section 19.13 - 45 nm and 32 nm Intel® Atom™ Processors
- Section 19.14 - Intel® Core™ Solo and Intel® Core™ Duo processors
- Section 19.15 - Processors based on Intel NetBurst® microarchitecture
- Section 19.16 - Pentium® M family processors
- Section 19.17 - P6 family processors
- Section 19.18 - Pentium® processors

NOTE

These performance-monitoring events are intended to be used as guides for performance tuning. The counter values reported by the performance-monitoring events are approximate and believed to be useful as relative guides for tuning software. Known discrepancies are documented where applicable.

All performance event encodings not documented in the appropriate tables for the given processor are considered reserved, and their use will result in undefined counter updates with associated overflow actions.

The event tables listed this chapter provide information for tool developers to support architectural and non-architectural performance monitoring events. The tables are up to date at processor launch, but are subject to changes. The most up to date event tables and additional details of performance event implementation for end-user (including additional details beyond event code/umask) can found at the "perfmon" repository provided by The Intel Open Source Technology Center (https://download.01.org/perfmon/).

...

19.2 PERFORMANCE MONITORING EVENTS FOR 6TH GENERATION INTEL® CORE™ PROCESSOR

6th Generation Intel® Core™ processors are based on the Skylake microarchitecture. They support the architectural performance-monitoring events listed in Table 41-56. Fixed counters in the core PMU support the architecture events defined in Table 19-2. Non-architectural performance-monitoring events in the processor core are listed in Table 19-3. The events in Table 19-3 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_4EH and 06_5EH. Table 19-8 lists performance events supporting Intel TSX (see Section 18.11.5) and are applicable to processors based on Skylake microarchitecture. Where Skylake microarchitecture implements TSX-related event semantics that differ from Table 19-8, they are listed in Table 19-4.

The comment column in Table 19-3 uses abbreviated letters to indicate additional conditions applicable to the Event Mask Mnemonic. For event umasks listed in Table 19-3 that do not show "AnyT", users should refrain from programming "AnyThread =1" in IA32_PERF_EVTSELx.

...

19.11 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON THE GOLDMONT MICROARCHITECTURE

Next Generation Intel Atom processors based on the Goldmont microarchitecture support the architectural performance-monitoring events listed in Table 41-56 and fixed-function performance events using a fixed counter. In addition, they also support the following non-architectural performance-monitoring events listed in Table 19-24. These events also apply to processors with CPUID signatures of 06_5CH and 06_5FH.

Performance monitoring event descriptions may refer to terminology described in Section B.2, "Performance Monitoring and Microarchitecture," in Appendix B of the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

In Goldmont microarchitecture, performance monitoring events that support Processor Event Based Sampling (PEBS) and PEBS records that contain processor state information that are associated with at-retirement tagging are marked by "Precise Event".

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture

Event Num.	Umask Value	Event Name	Description	Comment
03H	10H	LD_BLOCKS.ALL_BLOCK	Counts anytime a load that retires is blocked for any reason.	Precise Event
03H	08H	LD_BLOCKS.UTLB_MISS	Counts loads blocked because they are unable to find their physical address in the micro TLB (UTLB).	Precise Event
03H	02H	LD_BLOCKS.STORE_FORWARD	Counts a load blocked from using a store forward because of an address/size mismatch; only one of the loads blocked from each store will be counted.	Precise Event
03H	01H	LD_BLOCKS.DATA_UNKNOWN	Counts a load blocked from using a store forward, but did not occur because the store data was not available at the right time. The forward might occur subsequently when the data is available.	Precise Event
03H	04H	LD_BLOCKS.4K_ALIAS	Counts loads that block because their address modulo 4K matches a pending store.	Precise Event
05H	01H	PAGE_WALKS.D_SIDE_CYCLES	Counts every core cycle when a Data-side (walks due to data operation) page walk is in progress.	
05H	02H	PAGE_WALKS.I_SIDE_CYCLES	Counts every core cycle when an Instruction-side (walks due to an instruction fetch) page walk is in progress.	
05H	03H	PAGE_WALKS.CYCLES	Counts every core cycle a page-walk is in progress due to either a data memory operation, or an instruction fetch.	
0EH	00H	UOPS_ISSUED.ANY	Counts uops issued by the front end and allocated into the back end of the machine. This event counts uops that retire as well as uops that were speculatively executed but didn't retire. The sort of speculative uops that might be counted includes, but is not limited to those uops issued in the shadow of a mispredicted branch, those uops that are inserted during an assist (such as for a denormal floating-point result), and (previously allocated) uops that might be canceled during a machine clear.	
13H	02H	MISALIGN_MEM_REF.LOAD_PAGE_SPLIT	Counts when a memory load of a uop that spans a page boundary (a split) is retired.	Precise Event

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
13H	04H	MISALIGN_MEM_REF.STORE_PAGE_SPLIT	Counts when a memory store of a uop that spans a page boundary (a split) is retired.	Precise Event
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	Counts memory requests originating from the core that reference a cache line in the L2 cache.	
2EH	41H	LONGEST_LAT_CACHE.MISS	Counts memory requests originating from the core that miss in the L2 cache.	
30H	00H	L2_REJECT_XQ.ALL	Counts the number of demand and prefetch transactions that the L2 XQ rejects due to a full or near full condition which likely indicates back pressure from the intra-die interconnect (IDI) fabric. The XQ may reject transactions from the L2Q (non-cacheable requests), L2 misses and L2 write-back victims.	
31H	00H	CORE_REJECT_L2Q.ALL	Counts the number of demand and L1 prefetcher requests rejected by the L2Q due to a full or nearly full condition which likely indicates back pressure from L2Q. It also counts requests that would have gone directly to the XQ, but are rejected due to a full or nearly full condition, indicating back pressure from the IDI link. The L2Q may also reject transactions from a core to ensure fairness between cores, or to delay a core's dirty eviction when the address conflicts with incoming external snoops.	
3CH	00H	CPU_CLK_UNHALTED.CORE_P	Core cycles when core is not halted. This event uses a programmable general purpose performance counter.	
3CH	01H	CPU_CLK_UNHALTED.REF	Reference cycles when core is not halted. This event uses a programmable general purpose performance counter.	
51H	01H	DL1.DIRTY_EVICTION	Counts when a modified (dirty) cache line is evicted from the data L1 cache and needs to be written back to memory. No count will occur if the evicted line is clean, and hence does not require a writeback.	
80H	01H	ICACHE.HIT	Counts requests to the Instruction Cache (ICache) for one or more bytes in an ICache Line and that cache line is in the Icache (hit). The event strives to count on a cache line basis, so that multiple accesses which hit in a single cache line count as one ICACHE.HIT. Specifically, the event counts when straight line code crosses the cache line boundary, or when a branch target is to a new line, and that cache line is in the ICache. This event counts differently than Intel processors based on the Silvermont microarchitecture.	
80H	02H	ICACHE.MISSES	Counts requests to the Instruction Cache (ICache) for one or more bytes in an ICache Line and that cache line is not in the Icache (miss). The event strives to count on a cache line basis, so that multiple accesses which miss in a single cache line count as one ICACHE.MISS. Specifically, the event counts when straight line code crosses the cache line boundary, or when a branch target is to a new line, and that cache line is not in the ICache. This event counts differently than Intel processors based on the Silvermont microarchitecture.	

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
80H	03H	ICACHE.ACCESSES	Counts requests to the Instruction Cache (ICache) for one or more bytes in an ICache Line. The event strives to count on a cache line basis, so that multiple fetches to a single cache line count as one ICACHE.ACCESS. Specifically, the event counts when accesses from straight line code crosses the cache line boundary, or when a branch target is to a new line. This event counts differently than Intel processors based on the Silvermont microarchitecture.	
81H	04H	ITLB.MISS	Counts the number of times the machine was unable to find a translation in the Instruction Translation Lookaside Buffer (ITLB) for a linear address of an instruction fetch. It counts when new translations are filled into the ITLB. The event is speculative in nature, but will not count translations (page walks) that are begun and not finished, or translations that are finished but not filled into the ITLB.	
86H	02H	FETCH_STALL.ICACHE_FILL_PENDING_CYCLES	Counts cycles that an ICache miss is outstanding, and instruction fetch is stalled. That is, the decoder queue is able to accept bytes, but the fetch unit is unable to provide bytes, while an Icache miss is outstanding. Note this event is not the same as cycles to retrieve an instruction due to an Icache miss. Rather, it is the part of the Instruction Cache (ICache) miss time where no bytes are available for the decoder.	

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
9CH	00H	UOPS_NOT_DELIVERED.ANY	This event is used to measure front-end inefficiencies, i.e., when the front end of the machine is not delivering uops to the back end and the back end has not stalled. This event can be used to identify if the machine is truly front-end bound. When this event occurs, it is an indication that the front end of the machine is operating at less than its theoretical peak performance. Background: We can think of the processor pipeline as being divided into 2 broader parts: the front end and the back end. The front end is responsible for fetching the instruction, decoding into uops in machine understandable format and putting them into a uop queue to be consumed by the back end. The back end then takes these uops and allocates the required resources. When all resources are ready, uops are executed. If the back end is not ready to accept uops from the front end, then we do not want to count these as front-end bottlenecks. However, whenever we have bottlenecks in the back end, we will have allocation unit stalls and eventually force the front end to wait until the back end is ready to receive more uops. This event counts only when the back end is requesting more micro-uops and the front end is not able to provide them. When 3 uops are requested and no uops are delivered, the event counts 3. When 3 are requested, and only 1 is delivered, the event counts 2. When only 2 are delivered, the event counts 1. Alternatively stated, the event will not count if 3 uops are delivered, or if the back end is stalled and not requesting any uops at all. Counts indicate missed opportunities for the front end to deliver a uop to the back end. Some examples of conditions that cause front-end efficiencies are: Icache misses, ITLB misses, and decoder restrictions that limit the front-end bandwidth. Known Issues: Some uops require multiple allocation slots. These uops will not be charged as a front end 'not delivered' opportunity, and will be regarded as a back-end problem. For example, the INC instruction has one uop that requires 2 issue slots. A stream of INC instructions will not count as UOPS_NOT_DELIVERED, even though only one instruction can be issued per clock. The low uop issue rate for a stream of INC instructions is considered to be a back-end issue.	
B7H	01H, 02H	OFFCORE_RESPONSE	Requires MSR_OFFCORE_RESP[0,1] to specify request type and response. (Duplicated for both MSRs.)	
C0H	00H	INST_RETIRED.ANY_P	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 event continues counting during hardware interrupts, traps, and inside interrupt handlers. This is an architectural performance event. This event uses a programmable general purpose performance counter. *This event is a Precise Event: the EventingRIP field in the PEBS record is precise to the address of the instruction which caused the event. Note: Because PEBS records can be collected only on IA32_PMC0, only one event can use the PEBS facility at a time.	Precise Event
C2H	00H	UOPS_RETIRED.ANY	Counts uops which have retired.	Precise Event

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
C2H	01H	UOPS_RETIRED.MS	Counts uops retired that are from the complex flows issued by the micro-sequencer (MS). Counts both the uops from a micro-coded instruction, and the uops that might be generated from a micro-coded assist.	Precise Event
C3H	01H	MACHINE_CLEARS.SMC	Counts the number of times that the processor detects that a program is writing to a code section and has to perform a machine clear because of that modification. Self-modifying code (SMC) causes a severe penalty in all Intel architecture processors.	
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Counts machine clears due to memory ordering issues. This occurs when a snoop request happens and the machine is uncertain if memory ordering will be preserved as another core is in the process of modifying the data.	
C3H	04H	MACHINE_CLEARS.FP_ASSIST	Counts machine clears due to floating-point (FP) operations needing assists. For instance, if the result was a floating-point denormal, the hardware clears the pipeline and reissues uops to produce the correct IEEE compliant denormal result.	
C3H	08H	MACHINE_CLEARS.DISAMBIGUATION	Counts machine clears due to memory disambiguation. Memory disambiguation happens when a load which has been issued conflicts with a previous un-retired store in the pipeline whose address was not known at issue time, but is later resolved to be the same as the load address.	
C3H	00H	MACHINE_CLEARS.ALL	Counts machine clears for any reason.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Counts branch instructions retired for all branch types. This is an architectural performance event.	Precise Event
C4H	7EH	BR_INST_RETIRED.JCC	Counts retired Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired, including both when the branch was taken and when it was not taken.	Precise Event
C4H	FEH	BR_INST_RETIRED.TAKEN_JCC	Counts Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired that were taken and does not count when the Jcc branch instruction were not taken.	Precise Event
C4H	F9H	BR_INST_RETIRED.CALL	Counts near CALL branch instructions retired.	Precise Event
C4H	FDH	BR_INST_RETIRED.REL_CALL	Counts near relative CALL branch instructions retired.	Precise Event
C4H	FBH	BR_INST_RETIRED.IND_CALL	Counts near indirect CALL branch instructions retired.	Precise Event
C4H	F7H	BR_INST_RETIRED.RETURN	Counts near return branch instructions retired.	Precise Event
C4H	EBH	BR_INST_RETIRED.NON_RETURN_IND	Counts near indirect call or near indirect jmp branch instructions retired.	Precise Event
C4H	BFH	BR_INST_RETIRED.FAR_BRANCH	Counts far branch instructions retired. This includes far jump, far call and return, and Interrupt call and return.	Precise Event
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Counts mispredicted branch instructions retired including all branch types.	Precise Event

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
C5H	7EH	BR_MISP_RETIRED.JCC	Counts mispredicted retired Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired, including both when the branch was supposed to be taken and when it was not supposed to be taken (but the processor predicted the opposite condition).	Precise Event
C5H	FEH	BR_MISP_RETIRED.TAKEN_JCC	Counts mispredicted retired Jcc (Jump on Conditional Code/Jump if Condition is Met) branch instructions retired that were supposed to be taken but the processor predicted that it would not be taken.	Precise Event
C5H	FBH	BR_MISP_RETIRED.IND_CALL	Counts mispredicted near indirect CALL branch instructions retired, where the target address taken was not what the processor predicted.	Precise Event
C5H	F7H	BR_MISP_RETIRED.RETURN	Counts mispredicted near RET branch instructions retired, where the return address taken was not what the processor predicted.	Precise Event
C5H	EBH	BR_MISP_RETIRED.NON_RETURN_IND	Counts mispredicted branch instructions retired that were near indirect call or near indirect jmp, where the target address taken was not what the processor predicted.	Precise Event
CAH	01H	ISSUE_SLOTS_NOT_CONSUMED.RESOURCE_FULL	Counts the number of issue slots per core cycle that were not consumed because of a full resource in the back end. Including but not limited to resources include the Re-order Buffer (ROB), reservation stations (RS), load/store buffers, physical registers, or any other needed machine resource that is currently unavailable. Note that uops must be available for consumption in order for this event to fire. If a uop is not available (Instruction Queue is empty), this event will not count.	
CAH	02H	ISSUE_SLOTS_NOT_CONSUMED.RECOVERY	Counts the number of issue slots per core cycle that were not consumed by the back end because allocation is stalled waiting for a mispredicted jump to retire or other branch-like conditions (e.g. the event is relevant during certain microcode flows). Counts all issue slots blocked while within this window, including slots where uops were not available in the Instruction Queue.	
CAH	00H	ISSUE_SLOTS_NOT_CONSUMED.ANY	Counts the number of issue slots per core cycle that were not consumed by the back end due to either a full resource in the back end (RESOURCE_FULL), or due to the processor recovering from some event (RECOVERY).	
CBH	01H	HW_INTERRUPTS.RECEIVED	Counts hardware interrupts received by the processor.	
CBH	04H	HW_INTERRUPTS.PENDING_AND_MASKED	Counts core cycles during which there are pending interrupts, but interrupts are masked (EFLAGS.IF = 0).	
CDH	00H	CYCLES_DIV_BUSY.ALL	Counts core cycles if either divide unit is busy.	
CDH	01H	CYCLES_DIV_BUSY.IDIV	Counts core cycles if the integer divide unit is busy.	
CDH	02H	CYCLES_DIV_BUSY.FPDIV	Counts core cycles if the floating point divide unit is busy.	
D0H	81H	MEM_UOPS_RETIRED.ALL_LOADS	Counts the number of load uops retired.	Precise Event
D0H	82H	MEM_UOPS_RETIRED.ALL_STORES	Counts the number of store uops retired.	Precise Event

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
D0H	83H	MEM_UOPS_RETIRED.ALL	Counts the number of memory uops retired that are either a load or a store or both.	Precise Event
D0H	11H	MEM_UOPS_RETIRED.DTLB_MISS_LOADS	Counts load uops retired that caused a DTLB miss.	Precise Event
D0H	12H	MEM_UOPS_RETIRED.DTLB_MISS_STORES	Counts store uops retired that caused a DTLB miss.	Precise Event
D0H	13H	MEM_UOPS_RETIRED.DTLB_MISS	Counts uops retired that had a DTLB miss on load, store or either. Note that when two distinct memory operations to the same page miss the DTLB, only one of them will be recorded as a DTLB miss.	Precise Event
D0H	21H	MEM_UOPS_RETIRED.LOCK_LOADS	Counts locked memory uops retired. This includes 'regular' locks and bus locks. To specifically count bus locks only, see the offcore response event. A locked access is one with a lock prefix, or an exchange to memory.	Precise Event
D0H	41H	MEM_UOPS_RETIRED.SPLIT_LOADS	Counts load uops retired where the data requested spans a 64 byte cache line boundary.	Precise Event
D0H	42H	MEM_UOPS_RETIRED.SPLIT_STORES	Counts store uops retired where the data requested spans a 64 byte cache line boundary.	Precise Event
D0H	43H	MEM_UOPS_RETIRED.SPLIT	Counts memory uops retired where the data requested spans a 64 byte cache line boundary.	Precise Event
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Counts load uops retired that hit the L1 data cache.	Precise Event
D1H	08H	MEM_LOAD_UOPS_RETIRED.L1_MISS	Counts load uops retired that miss the L1 data cache.	Precise Event
D1H	02H	MEM_LOAD_UOPS_RETIRED.L2_HIT	Counts load uops retired that hit in the L2 cache.	Precise Event
0xD1H	10H	MEM_LOAD_UOPS_RETIRED.L2_MISS	Counts load uops retired that miss in the L2 cache.	Precise Event
D1H	20H	MEM_LOAD_UOPS_RETIRED.HITM	Counts load uops retired where the cache line containing the data was in the modified state of another core or modules cache (HITM). More specifically, this means that when the load address was checked by other caching agents (typically another processor) in the system, one of those caching agents indicated that they had a dirty copy of the data. Loads that obtain a HITM response incur greater latency than most that is typical for a load. In addition, since HITM indicates that some other processor had this data in its cache, it implies that the data was shared between processors, or potentially was a lock or semaphore value. This event is useful for locating sharing, false sharing, and contended locks.	Precise Event

Table 19-24 Non-Architectural Performance Events for the Goldmont Microarchitecture (Contd.)

Event Num.	Umask Value	Event Name	Description	Comment
D1H	40H	MEM_LOAD_UOPS_RETIRED.WCB_HIT	Counts memory load uops retired where the data is retrieved from the WCB (or fill buffer), indicating that the load found its data while that data was in the process of being brought into the L1 cache. Typically a load will receive this indication when some other load or prefetch missed the L1 cache and was in the process of retrieving the cache line containing the data, but that process had not yet finished (and written the data back to the cache). For example, consider load X and Y, both referencing the same cache line that is not in the L1 cache. If load X misses cache first, it obtains and WCB (or fill buffer) begins the process of requesting the data. When load Y requests the data, it will either hit the WCB, or the L1 cache, depending on exactly what time the request to Y occurs.	Precise Event
D1H	80H	MEM_LOAD_UOPS_RETIRED.DRAM_HIT	Counts memory load uops retired where the data is retrieved from DRAM. Event is counted at retirment, so the speculative loads are ignored. A memory load can hit (or miss) the L1 cache, hit (or miss) the L2 cache, hit DRAM, hit in the WCB or receive a HITM response.	Precise Event
E6H	01H	BACLEARS.ALL	Counts the number of times a BACLEAR is signaled for any reason, including, but not limited to indirect branch/call, Jcc (Jump on Conditional Code/Jump if Condition is Met) branch, unconditional branch/call, and returns.	
E6H	08H	BACLEARS.RETURN	Counts BACLEARS on return instructions.	
E6H	10H	BACLEARS.COND	Counts BACLEARS on Jcc (Jump on Conditional Code/Jump if Condition is Met) branches.	
E7H	01H	MS_DECODED.MS_ENTRY	Counts the number of times the Microcode Sequencer (MS) starts a flow of uops from the MSROM. It does not count every time a uop is read from the MSROM. The most common case that this counts is when a micro-coded instruction is encountered by the front end of the machine. Other cases include when an instruction encounters a fault, trap, or microcode assist of any sort that initiates a flow of uops. The event will count MS startups for uops that are speculative, and subsequently cleared by branch mispredict or a machine clear.	
E9H	01H	DECODE_RESTRICTION.PREDECODE_WRONG	Counts the number of times the prediction (from the pre-decode cache) for instruction length is incorrect.	

19.12 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON THE SILVERMONT MICROARCHITECTURE

Processors based on the Silvermont microarchitecture support the architectural performance-monitoring events listed in Table 41-56 and fixed-function performance events using fixed counter. In addition, they also support the following non-architectural performance-monitoring events listed in Table 19-25. These processors have the CPUID signatures of 06_37H, 06_4AH, 06_4DH, 06_5AH, and 06_5DH.

Performance monitoring event descriptions may refer to terminology described in Section B.2, "Performance Monitoring and Microarchitecture," in Appendix B of the *Intel® 64 and IA-32 Architectures Optimization Reference Manual*.

...

20. Updates to Chapter 23, Volume 3B

Change bars show changes to Chapter 23 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3B:* System Programming Guide, Part 2.

...

23.8 RESTRICTIONS ON VMX OPERATION

VMX operation places restrictions on processor operation. These are detailed below:

- In VMX operation, processors may fix certain bits in CR0 and CR4 to specific values and not support other values. VMXON fails if any of these bits contains an unsupported value (see "VMXON—Enter VMX Operation" in Chapter 30). Any attempt to set one of these bits to an unsupported value while in VMX operation (including VMX root operation) using any of the CLTS, LMSW, or MOV CR instructions causes a general-protection exception. VM entry or VM exit cannot set any of these bits to an unsupported value. Software should consult the VMX capability MSRs IA32_VMX_CR0_FIXED0 and IA32_VMX_CR0_FIXED1 to determine how bits in CR0 are fixed (see Appendix A.7). For CR4, software should consult the VMX capability MSRs IA32_VMX_CR4_FIXED0 and IA32_VMX_CR4_FIXED1 (see Appendix A.8).

NOTES

The first processors to support VMX operation require that the following bits be 1 in VMX operation: CR0.PE, CR0.NE, CR0.PG, and CR4.VMXE. The restrictions on CR0.PE and CR0.PG imply that VMX operation is supported only in paged protected mode (including IA-32e mode). Therefore, guest software cannot be run in unpaged protected mode or in real-address mode. See Section 31.2, "Supporting Processor Operating Modes in Guest Environments," for a discussion of how a VMM might support guest software that expects to run in unpaged protected mode or in real-address mode.

Later processors support a VM-execution control called "unrestricted guest" (see Section 24.6.2). If this control is 1, CR0.PE and CR0.PG may be 0 in VMX non-root operation (even if the capability MSR IA32_VMX_CR0_FIXED0 reports otherwise).[1] Such processors allow guest software to run in unpaged protected mode or in real-address mode.

- VMXON fails if a logical processor is in A20M mode (see "VMXON—Enter VMX Operation" in Chapter 30). Once the processor is in VMX operation, A20M interrupts are blocked. Thus, it is impossible to be in A20M mode in VMX operation.
- The INIT signal is blocked whenever a logical processor is in VMX root operation. It is not blocked in VMX non-root operation. Instead, INITs cause VM exits (see Section 25.2, "Other Causes of VM Exits").
- Intel® Processor Trace (Intel PT) can be used in VMX operation only if IA32_VMX_MISC[14] is read as 1 (see Appendix A.6). On processors that support Intel PT but which do not allow it to be used in VMX operation, execution of VMXON clears IA32_RTIT_CTL.TraceEn (see "VMXON—Enter VMX Operation" in Chapter 30); any attempt to set that bit while in VMX operation (including VMX root operation) using the WRMSR instruction causes a general-protection exception.

...

[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.

21. Updates to Chapter 24, Volume 3B

Change bars show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3B:* System Programming Guide, Part 2.

...

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

[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
7	P — Segment present
11:8	Reserved
12	AVL — Available for use by system software
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]

- 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.
- The register SMBASE (32 bits). This register contains the base address of the logical processor’s SMRAM image.

[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.

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 *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*. Execution of STI with RFLAGS.IF = 0 blocks interrupts (and, optionally, other events) for one instruction after its execution. Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See the "MOV—Move a Value from the Stack" from Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*, and "POP—Pop a Value from the Stack" from Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*, and Section 6.8.3 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. Execution of a MOV to SS or a POP to SS blocks interrupts for one instruction after its execution. In addition, certain debug exceptions are inhibited between a MOV to SS or a POP to SS and a subsequent instruction. Setting this bit indicates that the blocking of all these events is in effect. This document uses the term "blocking by MOV SS," but it applies equally to POP SS.
2	Blocking by SMI	See Section 34.2. 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 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* and Section 34.8. 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	A VM exit saves this bit as 1 to indicate that the VM exit was incident to enclave mode.
31:5	Reserved	VM entry will fail if these bits are not 0. See Section 26.3.1.5.

...

24.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute two 32-bit 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** and the **secondary processor-based VM-execution controls**.

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.
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP 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
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.
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 28.2.
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 cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpaged protected mode or in real-address mode.
8	APIC-register virtualization	If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 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 25.5.5.
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 28.2.5.
18	EPT-violation #VE	If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 25.5.6.
19	Conceal VMX non-root operation from Intel PT	If this control is 1, Intel Processor Trace suppresses data packets that indicate the use of virtualization (see Chapter 36).

Table 24-7 Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
20	Enable XSAVES/ XRSTORS	If this control is 0, any execution of XSAVES or XRSTORS causes a #UD.
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).

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTLS2 (see Appendix A.3.3) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 26.2.1.1).

...

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 24-9 lists the VM-function controls. See Section 25.5.5 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-9 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 25.5.5.3.

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 25.5.5.3).

...

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 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 28.2.5.

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.

...

24.7.1 VM-Exit Controls

The **VM-exit controls** constitute a 32-bit vector that governs the basic operation of VM exits. Table 24-10 lists the controls supported. See Chapter 27 for complete details of how these controls affect VM exits.

Table 24-10 Definitions of VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit.[1] This control must be 0 on processors that do not support Intel 64 architecture.
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt's vector. The vector is stored in the VM-exit interruption-information field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the VM-exit interruption-information field is marked invalid.
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.

Table 24-10 Definitions of VM-Exit Controls (Contd.)

Bit Position(s)	Name	Description
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 VM exits from Intel PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM exit (see Chapter 36).

NOTES:

1. Since Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX 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.

...

24.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 24-12 lists the controls supported. See Chapter 24 for how these controls affect VM entries.

Table 24-12 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 34.15.7). 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.

Table 24-12 Definitions of VM-Entry Controls (Contd.)

Bit Position(s)	Name	Description
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 VM entries from Intel PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM entry (see Chapter 36).

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.

...

24.9.1 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 24-14.

Table 24-14 Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason
26:16	Reserved (cleared to 0)
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	Reserved (cleared to 0)
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 28 is set only by an SMM VM exit (see Section 34.15.2) 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 34.15.2.3.

- 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 34.15.2.
- Because some VM-entry failures load processor state from the host-state area (see Section 26.7), 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; 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 34.15.2.3 for details of when and how this field is used.
- **Guest-physical address** (64 bits). This field is used VM exits due to EPT violations and EPT misconfigurations. See Section 27.2.1 for details of when and how this field is used.

...

22. Updates to Chapter 25, Volume 3C

Change bars show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

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:[1]

- **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.
- **HLT.** The HLT instruction causes a VM exit if the "HLT exiting" VM-execution control is 1.

[1] Many of the items in this section refer to secondary 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 these controls were all 0. See Section 24.6.2.

- **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 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/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/mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.
- **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. If the CR3-target count in *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.

  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.
- **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]
- **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.
- **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 1and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 24.6.19).
- **XSAVES.** The XSAVES instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 24.6.19).

...

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:

[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 34.15.3.

[1] Some of the items in this section refer to secondary 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 these controls were all 0. See Section 24.6.2.

  - 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 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 28.2.1.
- **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 28.2.1.
  - — 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-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.
- **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.

[1] A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

- If the control is 1, 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.
- **RDTSC.** Behavior of the RDTSC instruction is determined by the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls:
  - If both controls are 0, RDTSC operates normally.
  - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
    - If the control is 0, RDTSC loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
    - If the control is 1, RDTSC first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.
  - If the “RDTSC exiting” VM-execution control is 1, RDTSC causes a VM exit.
- **RDTSCP.** Behavior of the RDTSCP instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
  - If the “enable RDTSCP” VM-execution control is 0, RDTSCP causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
  - If the “enable RDTSCP” VM-execution control is 1, treatment is based on the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls:
    - If both controls are 0, RDTSCP operates normally.
    - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
      - If the control is 0, RDTSCP loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
      - If the control is 1, RDTSCP first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

      In either case, RDTSCP also loads ECX with the value of bits 31:0 of the IA32_TSC_AUX MSR.
    - If the “RDTSC exiting” VM-execution control is 1, RDTSCP causes a VM exit.

- **SMSW.** The behavior of SMSW is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, 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.

  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.
- **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 if it attempts IA32_RTIT_CTL.TraceEn.[1]
  - If ECX contains 808H (indicating the TPR MSR), 80BH (the EOI MSR), or 83FH (self-IPI MSR), instruction behavior may 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 24.6.19):
    - 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 24.6.19):
    - 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.

...

[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).

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 26.5.1), 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 26.5.2), 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 15.3.7, "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 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 43.2 for details.

...

[1] This item includes the cases of an invalid opcode exception—#UD— generated by the UD2 instruction and a BOUND-range exceeded exception—#BR—generated by the BOUND instruction.

23. Updates to Chapter 26, Volume 3C

Change bars show changes to Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

26.2.1.3 VM-Entry Control Fields

VM entries perform the following checks on the VM-entry control fields.

- Reserved bits in the VM-entry controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.5).
- Fields relevant to VM-entry event injection must be set properly. These fields are the VM-entry interruption-information field (see Table 24-13 in Section 24.8.3), the VM-entry exception error code, and the VM-entry instruction length. If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the following must hold:
  - The field's interruption type (bits 10:8) is not set to a reserved value. Value 1 is reserved on all logical processors; value 7 (other event) is reserved on logical processors that do not support the 1-setting of the "monitor trap flag" VM-execution control.
  - The field's vector (bits 7:0) is consistent with the interruption type:
    - If the interruption type is non-maskable interrupt (NMI), the vector is 2.
    - If the interruption type is hardware exception, the vector is at most 31.
    - If the interruption type is other event, the vector is 0 (pending MTF VM exit).
  - The field's deliver-error-code bit (bit 11) is 1 if and only if (1) either (a) the "unrestricted guest" VM-execution control is 0; or (b) bit 0 (corresponding to CR0.PE) is set in the CR0 field in the guest-state area; (2) the interruption type is hardware exception; and (3) the vector indicates an exception that would normally deliver an error code (8 = #DF; 10 = TS; 11 = #NP; 12 = #SS; 13 = #GP; 14 = #PF; or 17 = #AC).
  - Reserved bits in the field (30:12) are 0.
  - If the deliver-error-code bit (bit 11) is 1, bits 31:15 of the VM-entry exception error-code field are 0.
  - If the interruption type is software interrupt, software exception, or privileged software exception, the VM-entry instruction-length field is in the range 0–15. A VM-entry instruction length of 0 is allowed only if IA32_VMX_MISC[30] is read as 1; see Appendix A.6.
- The following checks are performed for the VM-entry MSR-load address if the VM-entry MSR-load count field is non-zero:
  - The lower 4 bits of the VM-entry MSR-load address must be 0. The address should not set any bits beyond the processor's physical-address width.[1]
  - The address of the last byte in the VM-entry MSR-load area should not set any bits beyond the processor's physical-address width. The address of this last byte is VM-entry MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

  If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.

[1] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

- If the processor is not in SMM, the "entry to SMM" and "deactivate dual-monitor treatment" VM-entry controls must be 0.
- The "entry to SMM" and "deactivate dual-monitor treatment" VM-entry controls cannot both be 1.

...

26.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 23.8). The following are exceptions:
  - — Bit 0 (corresponding to CR0.PE) and bit 31 (PG) are not checked if the "unrestricted guest" VM-execution control is 1.[1]
  - — Bit 29 (corresponding to CR0.NW) and bit 30 (CD) are never checked because the values of these bits are not changed by VM entry; see Section 26.3.2.1.
- If bit 31 in the CR0 field (corresponding to PG) is 1, bit 0 in that field (PE) must also be 1.[2]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 23.8).
- If the "load debug controls" VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.
- The following checks are performed on processors that support Intel 64 architecture:
  - — If the "IA-32e mode guest" VM-entry control is 1, bit 31 in the CR0 field (corresponding to CR0.PG) and bit 5 in the CR4 field (corresponding to CR4.PAE) must each be 1.[3]
  - — If the "IA-32e mode guest" VM-entry control is 0, bit 17 in the CR4 field (corresponding to CR4.PCIDE) must be 0.
  - — The CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width are 0.[4,5]
  - — If the "load debug controls" VM-entry control is 1, bits 63:32 in the DR7 field must be 0. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
  - — The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the "load IA32_PERF_GLOBAL_CTRL" VM-entry control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 18-3).

[1] "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "unrestricted guest" VM-execution control were 0. See Section 24.6.2.

[2] If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

[3] If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

[4] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

[5] Bit 63 of the CR3 field in the guest-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

- If the "load IA32_PAT" VM-entry control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the "load IA32_EFER" VM-entry control is 1, the following checks are performed on the field for the IA32_EFER MSR :
  - Bits reserved in the IA32_EFER MSR must be 0.
  - Bit 10 (corresponding to IA32_EFER.LMA) must equal the value of the "IA-32e mode guest" VM-entry control. It must also be identical to bit 8 (LME) if bit 31 in the CR0 field (corresponding to CR0.PG) is 1.[1]
- If the "load IA32_BNDCFGS" VM-entry control is 1, the following checks are performed on the field for the IA32_BNDCFGS MSR :
  - Bits reserved in the IA32_BNDCFGS MSR must be 0.
  - The linear address in bits 63:12 must be canonical.

...

26.3.1.5 Checks on Guest Non-Register State

The following checks are performed on fields in the guest-state area corresponding to non-register state:

- Activity state.
  - The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section 24.4.2). Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.
  - The activity-state field must not indicate the HLT state if the DPL (bits 6:5) in the access-rights field for SS is not 0.[2]
  - The activity-state field must indicate the active state if the interruptibility-state field indicates blocking by either MOV-SS or by STI (if either bit 0 or bit 1 in that field is 1).
  - If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the interruption to be delivered (as defined by interruption type and vector) must not be one that would normally be blocked while a logical processor is in the activity state corresponding to the contents of the activity-state field. The following items enumerate the interruptions (as specified in the VM-entry interruption-information field) whose injection is allowed for the different activity states:
    - Active. Any interruption is allowed.
    - HLT. The only events allowed are the following:
      - Those with interruption type external interrupt or non-maskable interrupt (NMI).
      - Those with interruption type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
      - Those with interruption type other event and vector 0 (pending MTF VM exit).

      See Table 24-13 in Section 24.8.3 for details regarding the format of the VM-entry interruption-information field.
    - Shutdown. Only NMIs and machine-check exceptions are allowed.

[1] If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

[2] As noted in Section 24.4.1, SS.DPL corresponds to the logical processor's current privilege level (CPL).

    - Wait-for-SIPI. No interruptions are allowed.
  - The activity-state field must not indicate the wait-for-SIPI state if the “entry to SMM” VM-entry control is 1.
- Interruptibility state.
  - The reserved bits (bits 31:5) must be 0.
  - The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).
  - Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.
  - Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 0, indicating external interrupt.
  - Bit 1 (blocking by MOV-SS) must be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 2, indicating non-maskable interrupt (NMI).
  - Bit 2 (blocking by SMI) must be 0 if the processor is not in SMM.
  - Bit 2 (blocking by SMI) must be 1 if the “entry to SMM” VM-entry control is 1.
  - A processor may require bit 0 (blocking by STI) to be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 2, indicating NMI. Other processors may not make this requirement.
  - Bit 3 (blocking by NMI) must be 0 if the “virtual NMIs” VM-execution control is 1, the valid bit (bit 31) in the VM-entry interruption-information field is 1, and the interruption type (bits 10:8) in that field has value 2 (indicating NMI).
  - If bit 4 (enclave interruption) is 1, bit 1 (blocking by MOV-SS) must be 0 and the processor must support for SGX by enumerating CPUID.(EAX=07H,ECX=0):EBX.SGX[bit 2] as 1.

NOTE

If the “virtual NMIs” VM-execution control is 0, there is no requirement that bit 3 be 0 if the valid bit in the VM-entry interruption-information field is 1 and the interruption type in that field has value 2.

- Pending debug exceptions.
  - Bits 11:4, bit 13, bit 15, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0.
  - The following checks are performed if any of the following holds: (1) the interruptibility-state field indicates blocking by STI (bit 0 in that field is 1); (2) the interruptibility-state field indicates blocking by MOV SS (bit 1 in that field is 1); or (3) the activity-state field indicates HLT:
    - Bit 14 (BS) must be 1 if the TF flag (bit 8) in the RFLAGS field is 1 and the BTF flag (bit 1) in the IA32_DEBUGCTL field is 0.
    - Bit 14 (BS) must be 0 if the TF flag (bit 8) in the RFLAGS field is 0 or the BTF flag (bit 1) in the IA32_DEBUGCTL field is 1.
  - The following checks are performed if bit 16 (RTM) is 1:
    - Bits 11:0, bits 15:13, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0; bit 12 must be 1.
    - The processor must support for RTM by enumerating CPUID.(EAX=07H,ECX=0):EBX[bit 11] as 1.
    - The interruptibility-state field must not indicate blocking by MOV SS (bit 1 in that field must be 0).

- VMCS link pointer. The following checks apply if the field contains a value other than FFFFFFFF_FFFFFFFFH:
  - — Bits 11:0 must be 0.
  - — Bits beyond the processor's physical-address width must be 0.[1,2]
  - — The 4 bytes located in memory referenced by the value of the field (as a physical address) must satisfy the following:
    - Bits 30:0 must contain the processor's VMCS revision identifier (see Section 24.2).[3]
    - Bit 31 must contain the setting of the "VMCS shadowing" VM-execution control.[4] This implies that the referenced VMCS is a shadow VMCS (see Section 24.10) if and only if the "VMCS shadowing" VM-execution control is 1.
  - — If the processor is not in SMM or the "entry to SMM" VM-entry control is 1, the field must not contain the current VMCS pointer.
  - — If the processor is in SMM and the "entry to SMM" VM-entry control is 0, the field must differ from the executive-VMCS pointer.

26.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG =1, CR4.PAE = 1, and IA32_EFER.LME = 0, the logical processor uses **PAE paging** (see Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3A*).[5] When PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs.

A VM entry is to a guest that uses PAE paging if (1) bit 31 (corresponding to CR0.PG) is set in the CR0 field in the guest-state area; (2) bit 5 (corresponding to CR4.PAE) is set in the CR4 field; and (3) the "IA-32e mode guest" VM-entry control is 0. Such a VM entry checks the validity of the PDPTEs:

- If the "enable EPT" VM-execution control is 0, VM entry checks the validity of the PDPTEs referenced by the CR3 field in the guest-state area if either (1) PAE paging was not in use before the VM entry; or (2) the value of CR3 is changing as a result of the VM entry. VM entry may check their validity even if neither (1) nor (2) hold.[6]
- If the "enable EPT" VM-execution control is 1, VM entry checks the validity of the PDPTE fields in the guest-state area (see Section 24.4.2).

A VM entry to a guest that does not use PAE paging does not check the validity of any PDPTEs.

A VM entry that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use.[7] If MOV to CR3 would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

[1] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

[2] If IA32_VMX_BASIC[48] is read as 1, this field must not set any bits in the range 63:32; see Appendix A.1.

[3] Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.

[4] "VMCS shadowing" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "VMCS shadowing" VM-execution control were 0. See Section 24.6.2.

[5] On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine the number physical-address bits supported by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

[6] "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.

...

26.3.2.1 Loading Guest Control Registers, Debug Registers, and MSRs

The following items describe how guest control registers, debug registers, and MSRs are loaded on VM entry:

- CR0 is loaded from the CR0 field with the exception of the following bits, which are never modified on VM entry: ET (bit 4); reserved bits 15:6, 17, and 28:19; NW (bit 29) and CD (bit 30).[1] The values of these bits in the CR0 field are ignored.
- CR3 and CR4 are loaded from the CR3 field and the CR4 field, respectively.
- If the "load debug controls" VM-entry control is 1, DR7 is loaded from the DR7 field with the exception that bit 12 and bits 15:14 are always 0 and bit 10 is always 1. The values of these bits in the DR7 field are ignored.

  The first processors to support the virtual-machine extensions supported only the 1-setting of the "load debug controls" VM-entry control and thus always loaded DR7 from the DR7 field.
- The following describes how certain MSRs are loaded using fields in the guest-state area:
  - If the "load debug controls" VM-entry control is 1, the IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always loaded the IA32_DEBUGCTL MSR from the IA32_DEBUGCTL field.
  - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since this field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
  - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
  - The following are performed on processors that support Intel 64 architecture:
    - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 26.3.2.2).
    - If the "load IA32_EFER" VM-entry control is 0, bits in the IA32_EFER MSR are modified as follows:
      - IA32_EFER.LMA is loaded with the setting of the "IA-32e mode guest" VM-entry control.
      - If CR0 is being loaded so that CR0.PG = 1, IA32_EFER.LME is also loaded with the setting of the "IA-32e mode guest" VM-entry control.[2] Otherwise, IA32_EFER.LME is unmodified.

      See below for the case in which the "load IA32_EFER" VM-entry control is 1
  - If the "load IA32_PERF_GLOBAL_CTRL" VM-entry control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
  - If the "load IA32_PAT" VM-entry control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
  - If the "load IA32_EFER" VM-entry control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.
  - If the "load IA32_BNDCFGS" VM-entry control is 1, the IA32_BNDCFGS MSR is loaded from the IA32_BNDCFGS field.

[7] This implies that (1) bits 11:9 in each PDPTE are ignored; and (2) if bit 0 (present) is clear in one of the PDPTEs, bits 63:1 of that PDPTE are ignored.

[1] Bits 15:6, bit 17, and bit 28:19 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. Bits 15:6, bit 17, and bit 28:19 of CR0 are always 0 and CR0.ET is always 1.

[2] If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, VM entry must be loading CR0 so that CR0.PG = 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 26.4.

- The SMBASE register is unmodified by all VM entries except those that return from SMM.

...

26.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 26.3.1.6, the logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. A VM entry to a guest that uses PAE paging loads the PDPTEs into internal, non-architectural registers based on the setting of the "enable EPT" VM-execution control:

- If the control is 0, the PDPTEs are loaded from the page-directory-pointer table referenced by the physical address in the value of CR3 being loaded by the VM entry (see Section 26.3.2.1). The values loaded are treated as physical addresses in VMX non-root operation.
- If the control is 1, the PDPTEs are loaded from corresponding fields in the guest-state area (see Section 24.4.2). The values loaded are treated as guest-physical addresses in VMX non-root operation.

...

24. Updates to Chapter 27, Volume 3C

Change bars show changes to Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

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 27.4), EPT violation, EPT misconfiguration, or page-modification log-full event 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.GD, or IA32_DEBUGCTL.LBR. (Information about the nature of the debug exception is saved in the exit qualification field.)
  - — A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)

- An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
- An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the "acknowledge interrupt on exit" VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
- The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
- If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT[1]; or the TR register.
- No last-exception record is made if the event that would do so directly causes a VM exit.
- If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine-check event itself and not the resulting machine-check exception.
- If the logical processor is in an inactive state (see Section 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 26.6.8) 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).

[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.

    - 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.
- 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, 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, or page-modification log-full event 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 VM-exit interruption-information field; see Section 27.2.2.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, or page-modification log-full event 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 virtual-NMI blocking may be recorded in the VM-exit interruption-information field; see Section 27.2.2.
- 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, or page-modification log-full event 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 (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 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 40, "Enclave Exiting Events"). An AEX modifies architectural state (Section 40.3). 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).

...

27.2.1 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 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 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).
  - 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 34.15.2.3).[1]
- **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 retirement of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; 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; MWAIT; accesses to the APIC-access page (see Section 29.4); EPT violations; EOI virtualization (see Section 29.1.4); APIC-write emulation (see Section 29.4.3.3); and page-modification log full (see Section 28.2.5). 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	Reserved (cleared to 0).
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).
63:15	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

[1] Bit 31 of this field is set on certain VM-entry failures; see Section 26.7.

- 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 Switch

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Reserved (cleared to 0)
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
63:32	Reserved (cleared to 0). 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 27.2.4) 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).

— 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
7	Reserved (cleared to 0)

[1] The exit qualification is undefined if the access was part of the logging of a branch record or a precise-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
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	Reserved (cleared to 0)
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

Such a VM exit stores 1 for bit 31 for IDT-vectoring information field (see Section 27.2.3) 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.

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	Reserved (cleared to 0)
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Reserved (cleared to 0)

Table 27-4 Exit Qualification for MOV DR (Contd.)

Bit Position(s)	Contents
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	Reserved (cleared to 0)

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	Reserved (cleared to 0)
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

Bit 12 is undefined in any of the following cases:

- If the "NMI exiting" VM-execution control is 1 and the "virtual NMIs" VM-execution control is 0.
- If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 27.2.3).

Otherwise, bit 12 is defined as follows:

- If the "virtual NMIs" VM-execution control is 0, the EPT violation was caused by a memory access as part of execution of the IRET instruction, and blocking by NMI (see Table 24-3) was in effect before execution of IRET, bit 12 is set to 1.

Table 27-6 Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	▪ If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. ▪ Undefined if the APIC-access VM exit is due a guest-physical access
15:12	Access type: 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 10 = guest-physical access during event delivery 15 = guest-physical access for an instruction fetch or during instruction execution Other values not used
63:16	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

- If the "virtual NMIs" VM-execution control is 1,the EPT violation was caused by a memory access as part of execution of the IRET instruction, and virtual-NMI blocking was in effect before execution of IRET, bit 12 is set to 1.
- For all other relevant VM exits, bit 12 is cleared to 0.

— 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 28.2.5), only bit 12 of the exit qualification is defined, and only in some cases. It is undefined in the following cases:

- If the "NMI exiting" VM-execution control is 1 and the "virtual NMIs" VM-execution control is 0.
- If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 27.2.3).

Otherwise, it is defined as follows:

- If the "virtual NMIs" VM-execution control is 0, the page-modification log-full event was caused by a memory access as part of execution of the IRET instruction, and blocking by NMI (see Table 24-3) was in effect before execution of IRET, bit 12 is set to 1.
- If the "virtual NMIs" VM-execution control is 1,the page-modification log-full event was caused by a memory access as part of execution of the IRET instruction, and virtual-NMI blocking was in effect before execution of IRET, bit 12 is set to 1.
- For all other relevant VM exits, bit 12 is cleared to 0.

For these VM exits, all bits other than bit 12 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.

[1] Execution of WRMSR with ECX = 83FH (self-IPI MSR) can lead to an APIC-write VM exit; the exit qualification for such an APIC-write VM exit is 3F0H.

— VM exits due to attempts to execute INS or OUTS for which the relevant segment is usable (if the relevant segment is not usable, the value is undefined). (ES is always the relevant segment for INS; for OUTS, the relevant segment is DS unless overridden by an instruction prefix.) The linear address is the base address of relevant segment plus (E)DI (for INS) or (E)SI (for OUTS). Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

Table 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 that 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 that 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 (indicates that the guest-physical address was executable).
6	Reserved (cleared to 0).
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.
8	If bit 7 is 1: ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).
11:9	Reserved (cleared to 0).
12	NMI unblocking due to IRET
63:13	Reserved (cleared to 0).

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 28.2.3.2). 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 28.2.2).

— 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). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the EPT violation occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

- — For all other VM exits, the field is undefined.

- **Guest-physical address.** For a VM exit due to an EPT violation or an EPT misconfiguration, 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 INT3, INTO, BOUND, and UD2); external interrupts that occur while the "acknowledge interrupt on exit" VM-exit control is 1; and non-maskable interrupts (NMIs). 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 24-15). 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), or 6 (software exception). Hardware exceptions comprise all exceptions except 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 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 is undefined in any of the following cases:
    - If the "NMI exiting" VM-execution control is 1 and the "virtual NMIs" VM-execution control is 0.
    - If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 27.2.3).
    - If the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).

    Otherwise, bit 12 is defined as follows:
    - If the "virtual NMIs" VM-execution control is 0, the VM exit is due to a fault on the IRET instruction (other than a debug exception for an instruction breakpoint), and blocking by NMI (see Table 24-3) was in effect before execution of IRET, bit 12 is set to 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.

    - If the “virtual NMIs” VM-execution control is 1, the VM exit is due to a fault on the IRET instruction (other than a debug exception for an instruction breakpoint), and virtual-NMI blocking was in effect before execution of IRET, bit 12 is set to 1.
    - For all other relevant VM exits, bit 12 is cleared to 0.[1]
  - Bits 30:13 are always set to 0.
  - Bit 31 is always set to 1.

  For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- VM-exit interruption error code.
  - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally through the IDT. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the IDT-vectoring information field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).
  - For other VM exits, the value of this field is undefined.

27.2.3 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:[2]

- 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, or page-modification log-full 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 26.5.1.2).

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 24-16). The following items detail how this field is established for VM exits that occur during event delivery:

[1] The conditions imply that, if the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1, bit 12 is always cleared to 0 by VM exits due to debug exceptions.

[2] 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).

- 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 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 UD2 are hardware exceptions.

  Bits 10:8 may indicate privileged software interrupt if such an event was injected as part of VM entry.
- 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).[1] If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).
- Bit 12 is undefined.
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- IDT-vectoring error code.
  - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
  - For other VM exits, the value of this field is undefined.

27.2.4 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, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, RDMSR, RDPMC, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WRMSR, XRSTORS, XSETBV, and XSAVES.[2]
  - For VM exits due to software exceptions (those generated by executions of INT3 or INTO).

[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.

[2] 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.

— 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 resulting from accesses (see Section 29.4) during delivery of a software interrupt, privileged software exception, or software exception.[1]

— For VM exits due executions of VMFUNC that fail because one of the following is true:

- EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 25.5.5.2).
- EAX = 0 and either ECX ≥ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 25.5.5.3).

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 26.5.1.2). 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.

[1] 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.

- **VM-exit instruction information**. For VM exits due to attempts to execute INS, INVEPT, INVPCID, INVVPID, LIDT, LGDT, LLDT, 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 or RDSEED, 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.
  - For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 27-14.

  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 34.15.2.3. 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.

[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).

Table 27-9 Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID (Contd.)

Bit Position(s)	Content
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
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)

...

27.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs is 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.

- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 34.15.2.

...

27.3.3 Saving RIP, RSP, and RFLAGS

The contents of the RIP, RSP, and RFLAGS 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), the value saved references the INT3 or INTO 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*) or software exception (due to execution of INT3 or INTO) 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, or INTO).
  - Suppose that the VM exit is due to a task switch that was caused by a task gate in the IDT that was encountered for any reason except the direct access by a software interrupt or software exception. The value saved is that which would have been saved in the old task-state segment had the task switch completed normally.
  - If the VM exit is due to an execution of MOV to CR8 or WRMSR that reduced the value of bits 7:4 of VTPR (see Section 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:

[1] The reference here is to the full value of RIP before any truncation that would occur had the stack width been only 32 bits or 16 bits.

- — If the VM exit occurred in enclave mode, the value saved is 0 (the remaining items do not apply).
- — If the VM exit is caused directly by an event that would normally be delivered through the IDT, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate[1] or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.
- — If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
- — If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
- — If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.[2]
- — For APIC-access VM exits and for VM exits caused by EPT violations EPT misconfigurations, and page-modification log-full 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 27.2.3), 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.

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.
- 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.

[1] The reference here is to the full value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.

[2] This is true even if RFLAGS.RF was 1 before the instruction was executed. If, in response to such a VM exit, a VM monitor re-enters the guest to re-execute the instruction that caused the VM exit (for example, after clearing the VM-execution control that caused the VM exit), the instruction may encounter a code breakpoint that has already been processed. A VM monitor can avoid this by setting the guest value of RFLAGS.RF to 1 before resuming guest software.

[3] If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit that is incident to delivery of an event injected by VM entry leaves this bit unmodified.

- The pending debug exceptions field is saved as clear for all VM exits except the following:
  - A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
  - A VM exit with basic exit reason "TPR below threshold",[1] "virtualized EOI", "APIC write", or "monitor trap flag."
  - 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 26.6.3). 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 26.6.3) 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 15.3.7, "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.
    - 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 26.6.3). 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

[1] This item includes VM exits that occur as a result of certain VM entries (Section 26.6.7).

debug exceptions (see Section 26.6.3) 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.

...

27.5.1 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).[2]
    - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width (they are cleared to 0).[3] (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.

[1] A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.

[2] 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.

[3] 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.

  - 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:
  - 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.
  - IA32_SYSENTER_ESP MSR and IA32_SYSENTER_EIP MSR are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, 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 does support 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 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.

  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.

...

25. Updates to Chapter 29, Volume 3C

Change bars show changes to Chapter 29 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B:* System Programming Guide, Part 2.

...

29.2.2 Virtual-Interrupt Delivery

If a virtual interrupt has been recognized (see Section 29.2.1), it is delivered at an instruction boundary when the following conditions all hold: (1) RFLAGS.IF = 1; (2) there is no blocking by STI; (3) there is no blocking by MOV SS or by POP SS; and (4) the “interrupt-window exiting” VM-execution control is 0.

[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.

Virtual-interrupt delivery has the same priority as that of VM exits due to the 1-setting of the "interrupt-window exiting" VM-execution control.[1] Thus, non-maskable interrupts (NMIs) and higher priority events take priority over delivery of a virtual interrupt; delivery of a virtual interrupt takes priority over external interrupts and lower priority events.

Virtual-interrupt delivery wakes a logical processor from the same inactive activity states as would an external interrupt. Specifically, it wakes a logical processor from the states entered using the HLT and MWAIT instructions. It does not wake a logical processor in the shutdown state or in the wait-for-SIPI state.

Virtual-interrupt delivery updates the guest interrupt status (both RVI and SVI; see Section 24.4.2) and delivers an event within VMX non-root operation without a VM exit. The following pseudocode details the behavior of virtual-interrupt delivery (see Section 29.1.1 for definition of VISR, VIRR, and VPPR):

```
Vector ← RVI;
VISR[Vector] ← 1;
SVI ← Vector;
VPPR ← Vector & F0H;
VIRR[Vector] ← 0;
IF any bits set in VIRR
    THEN RVI ← highest index of bit set in VIRR
    ELSE RVI ← 0;
FI;
deliver interrupt with Vector through IDT;
cease recognition of any pending virtual interrupt;
```

If a logical processor is in enclave mode, an Asynchronous Enclave Exit (AEX) occurs before delivery of a virtual interrupt (see Chapter 40, "Enclave Exiting Events").

...

29.4 VIRTUALIZING MEMORY-MAPPED APIC ACCESSES

When the local APIC is in xAPIC mode, software accesses the local APIC's control registers using a memory-mapped interface. Specifically, software uses linear addresses that translate to physical addresses on page frame indicated by the base address in the IA32_APIC_BASE MSR (see Section 10.4.4, "Local APIC Status and Location"). This section describes how these accesses can be virtualized.

A virtual-machine monitor (VMM) can virtualize these memory-mapped APIC accesses by ensuring that any access to a linear address that would access the local APIC instead causes a VM exit. This could be done using paging or the extended page-table mechanism (EPT). Another way is by using the 1-setting of the "virtualize APIC accesses" VM-execution control.

If the "virtualize APIC accesses" VM-execution control is 1, the logical processor treats specially memory accesses using linear addresses that translate to physical addresses in the 4-KByte **APIC-access page**.[2] (The APIC-access page is identified by the **APIC-access address**, a field in the VMCS; see Section 24.6.8.)

In general, an access to the APIC-access page causes an **APIC-access VM exit**. APIC-access VM exits provide a VMM with information about the access causing the VM exit. Section 29.4.1 discusses the priority of APIC-access VM exits.

[1] A logical processor never recognizes or delivers a virtual interrupt if the "interrupt-window exiting" VM-execution control is 1. Because of this, the relative priority of virtual-interrupt delivery and VM exits due to the 1-setting of that control is not defined.

[2] Even when addresses are translated using EPT (see Section 28.2), the determination of whether an APIC-access VM exit occurs depends on an access's physical address, not its guest-physical address. Even when CR0.PG = 0, ordinary memory accesses by software use linear addresses; the fact that CR0.PG = 0 means only that the identity translation is used to convert linear addresses to physical (or guest-physical) addresses.

Certain VM-execution controls enable the processor to virtualize certain accesses to the APIC-access page without a VM exit. In general, this virtualization causes these accesses to be made to the virtual-APIC page instead of the APIC-access page.

NOTES

Unless stated otherwise, this section characterizes only linear accesses to the APIC-access page; an access to the APIC-access page is a linear access if (1) it results from a memory access using a linear address; and (2) the access's physical address is the translation of that linear address. Section 29.4.6 discusses accesses to the APIC-access page that are not linear accesses.

The distinction between the APIC-access page and the virtual-APIC page allows a VMM to share paging structures or EPT paging structures among the virtual processors of a virtual machine (the shared paging structures referencing the same APIC-access address, which appears in the VMCS of all the virtual processors) while giving each virtual processor its own virtual APIC (the VMCS of each virtual processor will have a unique virtual-APIC address).

Section 29.4.2 discusses when and how the processor may virtualize read accesses from the APIC-access page. Section 29.4.3 does the same for write accesses. When virtualizing a write to the APIC-access page, the processor typically takes actions in addition to passing the write through to the virtual-APIC page.

The discussion in those sections uses the concept of an **operation** within which these memory accesses may occur. For those discussions, an "operation" can be an iteration of a REP-prefixed string instruction, an execution of any other instruction, or delivery of an event through the IDT.

The 1-setting of the "virtualize APIC accesses" VM-execution control may also affect accesses to the APIC-access page that do not result directly from linear addresses. This is discussed in Section 29.4.6.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 42.5.3 for details.

...

29.6 POSTED-INTERRUPT PROCESSING

Posted-interrupt processing is a feature by which a processor processes the virtual interrupts by recording them as pending on the virtual-APIC page.

Posted-interrupt processing is enabled by setting the "process posted interrupts" VM-execution control. The processing is performed in response to the arrival of an interrupt with the **posted-interrupt notification vector**. In response to such an interrupt, the processor processes virtual interrupts recorded in a data structure called a **posted-interrupt descriptor**. The posted-interrupt notification vector and the address of the posted-interrupt descriptor are fields in the VMCS; see Section 24.6.8.

If the "process posted interrupts" VM-execution control is 1, a logical processor uses a 64-byte posted-interrupt descriptor located at the posted-interrupt descriptor address. The posted-interrupt descriptor has the following format:

The notation **PIR** (posted-interrupt requests) refers to the 256 posted-interrupt bits in the posted-interrupt descriptor.

Use of the posted-interrupt descriptor differs from that of other data structures that are referenced by pointers in a VMCS. There is a general requirement that software 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. That requirement does not apply to the posted-interrupt descriptor. There is a requirement, however, that such modifications be done using locked read-modify-write instructions.

Table 29-1 Format of Posted-Interrupt Descriptor

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1
256	Outstanding notification	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0
511:257	Reserved for software and other agents	These bits may be used by software and by other agents in the system (e.g., chipset). The processor does not modify these bits.

If the "external-interrupt exiting" VM-execution control is 1, any unmasked external interrupt causes a VM exit (see Section 25.2). If the "process posted interrupts" VM-execution control is also 1, this behavior is changed and the processor handles an external interrupt as follows:[1]

1. The local APIC is acknowledged; this provides the processor core with an interrupt vector, called here the **physical vector**.
2. If the physical vector equals the posted-interrupt notification vector, the logical processor continues to the next step. Otherwise, a VM exit occurs as it would normally due to an external interrupt; the vector is saved in the VM-exit interruption-information field.
3. The processor clears the outstanding-notification bit in the posted-interrupt descriptor. This is done atomically so as to leave the remainder of the descriptor unmodified (e.g., with a locked AND operation).
4. The processor writes zero to the EOI register in the local APIC; this dismisses the interrupt with the posted-interrupt notification vector from the local APIC.
5. The logical processor performs a logical-OR of PIR into VIRR and clears PIR. No other agent can read or write a PIR bit (or group of bits) between the time it is read (to determine what to OR into VIRR) and when it is cleared.
6. The logical processor sets RVI to be the maximum of the old value of RVI and the highest index of all bits that were set in PIR; if no bit was set in PIR, RVI is left unmodified.
7. The logical processor evaluates pending virtual interrupts as described in Section 29.2.1.

The logical processor performs the steps above in an uninterruptible manner. If step #7 leads to recognition of a virtual interrupt, the processor may deliver that interrupt immediately.

Steps #1 to #7 above occur when the interrupt controller delivers an unmasked external interrupt to the CPU core. This delivery can occur when the logical processor is in the active, HLT, or MWAIT states. If the logical processor had been in the active or MWAIT state before the arrival of the interrupt, it is in the active state following completion of step #7; if it had been in the HLT state, it returns to the HLT state after step #7 (if a pending virtual interrupt was recognized, the logical processor may immediately wake from the HLT state).

If an external interrupt causes a VM exit while the logical processor is in enclave mode, an Asynchronous Enclave Exit (AEX) occurs before the VM exit is delivered (see Chapter 40, "Enclave Exiting Events"). If the "external-interrupt exiting" VM-execution control is 1, any unmasked external interrupt may cause an AEX even if no VM exit occurs (e.g., if the process above does not cause a VM exit at step #2).

...

[1] VM entry ensures that the "process posted interrupts" VM-execution control is 1 only if the "external-interrupt exiting" VM-execution control is also 1. SeeSection 26.2.1.1.

26. Updates to Chapter 30, Volume 3C

Change bars show changes to Chapter 30 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

VMXON—Enter VMX Operation

Opcode	Instruction	Description
F3 0F C7 /6	VMXON m64	Enter VMX root operation.

Description

Puts the logical processor in VMX operation with no current VMCS, blocks INIT signals, disables A20M, and clears any address-range monitoring established by the MONITOR instruction.[1]

The operand of this instruction is a 4KB-aligned physical address (the VMXON pointer) that references the VMXON region, which the logical processor may use to support VMX operation. This operand is always 64 bits and is always in memory.

Operation

```
IF (register operand) or (CR0.PE = 0) or (CR4.VMXE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF not in VMX operation
    THEN
        IF (CPL > 0) or (in A20M mode) or
        (the values of CR0 and CR4 are not supported in VMX operation; see Section 23.8) or
        (bit 0 (lock bit) of IA32_FEATURE_CONTROL MSR is clear) or
        (in SMX operation[2] and bit 1 of IA32_FEATURE_CONTROL MSR is clear) or
        (outside SMX operation and bit 2 of IA32_FEATURE_CONTROL MSR is clear)
            THEN #GP(0);
            ELSE
                addr ← contents of 64-bit in-memory source operand;
                IF addr is not 4KB-aligned or
                addr sets any bits beyond the physical-address width[3]
                    THEN VMfailInvalid;
                    ELSE
                        rev ← 32 bits located at physical address addr;
                        IF rev[30:0] ≠ VMCS revision identifier supported by processor OR rev[31] = 1
                            THEN VMfailInvalid;
                            ELSE
```

[1] See the information on MONITOR/MWAIT in Chapter 8, "Multiple-Processor Management," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

[2] A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference."

[3] If IA32_VMX_BASIC[48] is read as 1, VMfailInvalid occurs if addr sets any bits in the range 63:32; see Appendix A.1.

```
                                        current-VMCS pointer ← FFFFFFFF_FFFFFFFFH;
                                        enter VMX operation;
                                        block INIT signals;
                                        block and disable A20M;
                                        clear address-range monitoring;
                                        IF the processor supports Intel PT but does not allow it to be used in VMX operation[1]
                                            THEN IA32_RTIT_CTL.TraceEn ← 0;
                                        FI;
                                        VMsucceed;
                                FI;
                        FI;
                FI;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE VMfail("VMXON executed in VMX root operation");
FI;
```

Flags Affected

See the operation section and Section 30.2.

Protected Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL>0 or with invalid CR0 or CR4 fixed bits.
	If executed in A20M mode.
	If the memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the memory source operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If operand is a register.
	If executed with CR4.VMXE = 0.

Real-Address Mode Exceptions

#UD	If executed outside VMX root operation.

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

#UD	The VMXON instruction is not recognized in compatibility mode.

[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).

64-Bit Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL > 0 or with invalid CR0 or CR4 fixed bits. If executed in A20M mode. If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register. If executed with CR4.VMXE = 0.

...

27. Updates to Chapter 35, Volume 3C

Change bars show changes to Chapter 35 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3C:* System Programming Guide, Part 3.

...

This chapter lists MSRs across Intel processor families. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions.

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-M" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 2A*). Table 35-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table 35-1 CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_57H	Next Generation Intel® Xeon Phi™ Processor Family
06_8EH, 06_9EH	Future Intel® Core Processors
06_55H	Future Intel® Xeon Processors
06_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family and E3-1200 v5 product family based on Skylake microarchitecture
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture
06_4FH	Intel Xeon processor E5 v4 Family based on Broadwell microarchitecture
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

Table 35-1 CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
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 Intel microarchitecture code name Sandy Bridge, 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_5FH	Future Intel® Atom™ processors based on Goldmont Microarchitecture
06_5CH	Next Generation 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
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

Table 35-1 CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
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

Intel Quark X1000 processor can be identified by the signature of DisplayFamily_DisplayModel = 05_09H and SteppingID = 0

35.1 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 35-2 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 35-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a machine specified 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 35-2 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 35-1) 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 35-2. "MAXPHYADDR" is reported by CPUID.8000_0008H leaf.

MSR address range between 40000000H - 400000FFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

Table 35-2 IA-32 Architectural MSRs

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Section 35.22, "MSRs in Pentium Processors."	**Pentium Processor (05_01H)**
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 35.22, "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
10H	16	IA32_TIME_STAMP_ COUNTER (TSC)	See Section 17.15, "Time-Stamp Counter."	05_01H

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	**Platform ID (RO)** 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 (RO)** 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)		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 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and 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 system executive that do not require SMX. BIOS must set this bit only when the CPUID function 1 returns 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 bit 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 reconfiguration of SGX Launch Control via 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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		19	Reserved	
		20	LMCE On (R/WL): When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.	If IA32_MCG_CAP[27] = 1
		63:21	Reserved	
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.	
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 (RO) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
		31:0	Reserved	
		63:32	It is recommended that this field be pre-loaded with 0 prior to executing CPUID. If the field remains 0 following the execution of CPUID; this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
8CH	140	IA32_SGXLEPUBKEYHASH0	IA32_SGXLEPUBKEYHASH[63:0] (R/W) Bits 63:0 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1, 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.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1, Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1
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.	Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1, Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1
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 34.14.4)	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 (RO, SMM only)	If IA32_VMX_MISC[15]
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
E7H	231	IA32_MPERF	TSC Frequency Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	**C0_MCNT: C0 TSC Frequency Clock Count** Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_APERF.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear).	If CPUID.06H: ECX[0] = 1
		63:0	**C0_ACNT: C0 Actual Frequency Clock Count** Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_MPERF.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) 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.	
		63:12	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR (R/W)	06_01H
		15:0	CS Selector	
		63:16	Reserved.	
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 (RO)	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 support extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH
		63:28	Reserved.	
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		3	LMCE_S.	If IA32_MCG_CAP.LMCE_P[27] =1
		63:4	Reserved.	
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[1]
186H	390	IA32_PERFEVTSEL0 (PERFEVTSEL0)	Performance Event Select Register 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
		7:0	Event Select: Selects a performance event logic unit.	
		15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	
		16	USR: Counts while in privilege level is not ring 0.	
		17	OS: Counts while in privilege level is ring 0.	
		18	Edge: Enables edge detection if set.	
		19	PC: enables pin control.	
		20	INT: enables interrupt on counter overflow.	
		21	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
		22	EN: enables the corresponding performance counter to commence counting when this bit is set.	
		23	INV: invert the CMASK.	
		31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.	
		63:32	Reserved.	
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
189H	393	IA32_PERFEVTSEL3	Performance Event Select Register 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
18AH-197H	394-407	Reserved		06_0EH[2]
198H	408	IA32_PERF_STATUS	(RO)	0F_03H
		15:0	Current performance State Value	
		63:16	Reserved.	
199H	409	IA32_PERF_CTL	(R/W)	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.	
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 14.7.3, “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 14.7.2, “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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 (RO) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 14.7.2, "Thermal Monitor"	If CPUID.01H:EDX[22] = 1
		0	Thermal Status (RO):	If CPUID.01H:EDX[22] = 1
		1	Thermal Status Log (R/W):	If CPUID.01H:EDX[22] = 1
		2	PROCHOT # or FORCEPR# event (RO)	If CPUID.01H:EDX[22] = 1
		3	PROCHOT # or FORCEPR# log (R/WC0)	If CPUID.01H:EDX[22] = 1
		4	Critical Temperature Status (RO)	If CPUID.01H:EDX[22] = 1
		5	Critical Temperature Status log (R/WC0)	If CPUID.01H:EDX[22] = 1
		6	Thermal Threshold #1 Status (RO)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		8	Thermal Threshold #2 Status (RO)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #2 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		10	Power Limitation Status (RO)	If CPUID.06H:EAX[4] = 1
		11	Power Limitation log (R/WC0)	If CPUID.06H:EAX[4] = 1
		12	Current Limit Status (RO)	If CPUID.06H:EAX[7] = 1
		13	Current Limit log (R/WC0)	If CPUID.06H:EAX[7] = 1
		14	Cross Domain Limit Status (RO)	If CPUID.06H:EAX[7] = 1
		15	Cross Domain Limit log (R/WC0)	If CPUID.06H:EAX[7] = 1
		22:16	Digital Readout (RO)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved.	
		30:27	Resolution in Degrees Celsius (RO)	If CPUID.06H:EAX[0] = 1
		31	Reading Valid (RO)	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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	**Fast-Strings Enable** When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default); when clear, fast-strings are disabled.	0F_0H
		2:1	Reserved.	
		3	**Automatic Thermal Control Circuit Enable (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled. Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2 and adaptive thermal throttling will still be activated. The default value of this field varies with product . See respective tables where default value is listed.	0F_0H
		6:4	Reserved	
		7	**Performance Monitoring Available (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled	0F_0H
		10:8	Reserved.	
		11	**Branch Trace Storage Unavailable (RO)** 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported	0F_0H
		12	**Processor Event Based Sampling (PEBS) Unavailable (RO)** 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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		18	**ENABLE MONITOR FSM (R/W)** When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/ MWAIT are not supported. Software attempts to execute MONITOR/ MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
		21:19	Reserved.	
		22	**Limit CPUID Maxval (R/W)** When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 2. Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 2, this bit is supported. Otherwise, this bit is not supported. Setting this bit when the maximum value is not greater than 2 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 2.	0F_03H
		23	**xTPR Message Disable (R/W)** When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	if CPUID.01H:ECX[14] = 1
		33:24	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		34	**XD Bit Disable (R/W)** When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0). When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location, if XD bit is not supported. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	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	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
		63:4	Reserved.	
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package Thermal Status Information (RO) Contains status information about the package's thermal sensor. See Section 14.8, "Package Level Thermal Management."	If CPUID.06H: EAX[6] = 1
		0	Pkg Thermal Status (RO):	
		1	Pkg Thermal Status Log (R/W):	
		2	Pkg PROCHOT # event (RO)	
		3	Pkg PROCHOT # log (R/WC0)	
		4	Pkg Critical Temperature Status (RO)	
		5	Pkg Critical Temperature Status log (R/WC0)	
		6	Pkg Thermal Threshold #1 Status (RO)	
		7	Pkg Thermal Threshold #1 log (R/WC0)	
		8	Pkg Thermal Threshold #2 Status (RO)	
		9	Pkg Thermal Threshold #1 log (R/WC0)	
		10	Pkg Power Limitation Status (RO)	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		11	Pkg Power Limitation log (R/WC0)	
		15:12	Reserved.	
		22:16	Pkg Digital Readout (RO)	
		63:23	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 14.8, "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	
		23	Pkg Threshold #2 Interrupt Enable	
		24	Pkg Power Limit Notification Enable	
		63:25	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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	
1F2H	498	IA32_SMRR_PHYSBASE	**SMRR Base Address (Writeable only in SMM)** Base address of SMM memory range.	If IA32_MTRRCAP.SMRR[11] = 1
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved.	
		31:12	**PhysBase.** SMRR physical Base Address.	
		63:32	Reserved.	
1F3H	499	IA32_SMRR_PHYSMASK	**SMRR Range Mask. (Writeable only in SMM)** Range Mask of SMM memory range.	If IA32_MTRRCAP[SMRR] = 1
		10:0	Reserved.	
		11	**Valid** Enable range mask.	
		31:12	**PhysMask** SMRR address range mask.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 CPUID.01H: EDX.MTRR[12] =1
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	If CPUID.01H: EDX.MTRR[12] =1
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	If CPUID.01H: EDX.MTRR[12] =1
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	If CPUID.01H: EDX.MTRR[12] =1
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	If CPUID.01H: EDX.MTRR[12] =1
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	If CPUID.01H: EDX.MTRR[12] =1
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	If CPUID.01H: EDX.MTRR[12] =1
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	If CPUID.01H: EDX.MTRR[12] =1
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	If CPUID.01H: EDX.MTRR[12] =1

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	If CPUID.01H: EDX.MTRR[12] =1
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	If CPUID.01H: EDX.MTRR[12] =1
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	If CPUID.01H: EDX.MTRR[12] =1
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	If CPUID.01H: EDX.MTRR[12] =1
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	If CPUID.01H: EDX.MTRR[12] =1
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	If CPUID.01H: EDX.MTRR[12] =1
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	If CPUID.01H: EDX.MTRR[12] =1
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
277H	631	IA32_PAT	IA32_PAT (R/W)	If CPUID.01H: EDX.MTRR[16] =1
		2:0	PA0	
		7:3	Reserved.	
		10:8	PA1	
		15:11	Reserved.	
		18:16	PA2	
		23:19	Reserved.	
		26:24	PA3	
		31:27	Reserved.	
		34:32	PA4	
		39:35	Reserved.	
		42:40	PA5	
		47:43	Reserved.	
		50:48	PA6	
		55:51	Reserved.	
		58:56	PA7	
		63:59	Reserved.	
280H	640	IA32_MC0_CTL2	(R/W)	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_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
282H	642	IA32_MC2_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
283H	643	IA32_MC3_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
284H	644	IA32_MC4_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
285H	645	IA32_MC5_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
286H	646	IA32_MC6_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
287H	647	IA32_MC7_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
288H	648	IA32_MC8_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
289H	649	IA32_MC9_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
28AH	650	IA32_MC10_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
28BH	651	IA32_MC11_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
28CH	652	IA32_MC12_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
28DH	653	IA32_MC13_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
28EH	654	IA32_MC14_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
28FH	655	IA32_MC15_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
290H	656	IA32_MC16_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
291H	657	IA32_MC17_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
292H	658	IA32_MC18_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
293H	659	IA32_MC19_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
294H	660	IA32_MC20_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
295H	661	IA32_MC21_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
296H	662	IA32_MC22_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
297H	663	IA32_MC23_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
298H	664	IA32_MC24_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
299H	665	IA32_MC25_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
29AH	666	IA32_MC26_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
29BH	667	IA32_MC27_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
29CH	668	IA32_MC28_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
29DH	669	IA32_MC29_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
29EH	670	IA32_MC30_CTL2	(R/W) same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
29FH	671	IA32_MC31_CTL2	(R/W) same fields as IA32_MC0_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	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		9:3	Reserved.	
		10	Fixed Range MTRR Enable	
		11	MTRR Enable	
		63:12	Reserved.	
309H	777	IA32_FIXED_CTR0 (MSR_PERF_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 (MSR_PERF_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 (MSR_PERF_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	RO	If CPUID.01H: ECX[15] = 1
		5:0	LBR format	
		6	PEBS Trap	
		7	PEBSSaveArchRegs	
		11:8	PEBS Record Format	
		12	1: Freeze while SMM is supported.	
		13	1: Full width of counter writable via IA32_A_PMCx.	
		63:14	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	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0.	
		1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0.	
		2	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.	If CPUID.0AH: EAX[7:0] > 2
		3	EN0_PMI: Enable PMI when fixed counter 0 overflows.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		4	EN1_OS: Enable Fixed Counter 1to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1to count while CPL > 0.	
		6	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.	If CPUID.0AH: EAX[7:0] > 2
		7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
		8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	
		9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0.	
		10	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.	If CPUID.0AH: EAX[7:0] > 2
		11	EN2_PMI: Enable PMI when fixed counter 2 overflows.	
		63:12	Reserved.	
38EH	910	IA32_PERF_GLOBAL_STATUS	Global Performance Counter Status (RO)	If CPUID.0AH: EAX[7:0] > 0
		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
		31:4	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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH: EAX[7:0] > 1
		54:35	Reserved.	
		55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer was completely filled.	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && IA32_RTIT_CTL.ToPA = 1
		57:56	Reserved.	
		58	LBR_Frz: LBRs are frozen due to ▪ IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1, ▪ The LBR stack overflowed	If CPUID.0AH: EAX[7:0] > 3
		59	CTR_Frz: Performance counters in the core PMU are frozen due to ▪ IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1, ▪ one or more core PMU counters overflowed.	If CPUID.0AH: EAX[7:0] > 3
		60	ASCI: Data in the performance counters in the core PMU may include contributions from the direct or indirect operation intel SGX to protect an enclave.	If CPUID.(EAX=07H, ECX=0):EBX[2] = 1
		61	Ovf_Uncore: Uncore counter overflow status.	If CPUID.0AH: EAX[7:0] > 2
		62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.0AH: EAX[7:0] > 0
		63	CondChgd: status bits of this register has changed.	If CPUID.0AH: EAX[7:0] > 0
38FH	911	IA32_PERF_GLOBAL_CTRL	Global Performance Counter Control (R/W) Counter increments while the result of ANDing respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.	If CPUID.0AH: EAX[7:0] > 0
		0	EN_PMC0	If CPUID.0AH: EAX[15:8] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[15:8] > 1
		2	EN_PMC2	If CPUID.0AH: EAX[15:8] > 2
		n	EN_PMCn	If CPUID.0AH: EAX[15:8] > n
		31:n+1	Reserved.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and 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
		63:35	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 to 1to 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
		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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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[8] = 1
		57:56	Reserved.	
		58	Set 1 to Clear LBR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to Clear CTR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to Clear ASCI bit.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf: bit.	If CPUID.0AH: EAX[7:0] > 0
		63	Set to 1to 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
		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
		54:35	Reserved.	
		55	Set 1 to cause Trace_ToPA_PMI = 1.	If CPUID.0AH: EAX[7:0] > 3
		57:56	Reserved.	
		58	Set 1 to cause LBR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to cause CTR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to cause ASCI = 1.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.0AH: EAX[7:0] > 3
		62	Set 1 to cause OvfBuf = 1.	If CPUID.0AH: EAX[7:0] > 3

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63	Reserved	
392H	914	IA32_PERF_GLOBAL_INUSE	Indicator of core perfmon interface is in use (RO)	If CPUID.0AH: EAX[7:0] > 3
		0	IA32_PERFEVTSEL0 in use	
		1	IA32_PERFEVTSEL1 in use	If CPUID.0AH: EAX[15:8] > 1
		2	IA32_PERFEVTSEL2 in use	If CPUID.0AH: EAX[15:8] > 2
		n	IA32_PERFEVTSELn in use	If CPUID.0AH: EAX[15:8] > n
		31:n	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_MC0_CTL	MC0_CTL	If IA32_MCG_CAP.CNT >0
401H	1025	IA32_MC0_STATUS	MC0_STATUS	If IA32_MCG_CAP.CNT >0
402H	1026	IA32_MC0_ADDR[1]	MC0_ADDR	If IA32_MCG_CAP.CNT >0
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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		IA32_MC22_CTL	MC22_CTL	If IA32_MCG_CAP.CNT >22
459H		IA32_MC22_STATUS	MC22_STATUS	If IA32_MCG_CAP.CNT >22
45AH		IA32_MC22_ADDR[1]	MC22_ADDR	If IA32_MCG_CAP.CNT >22
45BH		IA32_MC22_MISC	MC22_MISC	If IA32_MCG_CAP.CNT >22
45CH		IA32_MC23_CTL	MC23_CTL	If IA32_MCG_CAP.CNT >23
45DH		IA32_MC23_STATUS	MC23_STATUS	If IA32_MCG_CAP.CNT >23
45EH		IA32_MC23_ADDR[1]	MC23_ADDR	If IA32_MCG_CAP.CNT >23
45FH		IA32_MC23_MISC	MC23_MISC	If IA32_MCG_CAP.CNT >23
460H		IA32_MC24_CTL	MC24_CTL	If IA32_MCG_CAP.CNT >24
461H		IA32_MC24_STATUS	MC24_STATUS	If IA32_MCG_CAP.CNT >24
462H		IA32_MC24_ADDR[1]	MC24_ADDR	If IA32_MCG_CAP.CNT >24
463H		IA32_MC24_MISC	MC24_MISC	If IA32_MCG_CAP.CNT >24
464H		IA32_MC25_CTL	MC25_CTL	If IA32_MCG_CAP.CNT >25
465H		IA32_MC25_STATUS	MC25_STATUS	If IA32_MCG_CAP.CNT >25
466H		IA32_MC25_ADDR[1]	MC25_ADDR	If IA32_MCG_CAP.CNT >25
467H		IA32_MC25_MISC	MC25_MISC	If IA32_MCG_CAP.CNT >25
468H		IA32_MC26_CTL	MC26_CTL	If IA32_MCG_CAP.CNT >26
469H		IA32_MC26_STATUS	MC26_STATUS	If IA32_MCG_CAP.CNT >26
46AH		IA32_MC26_ADDR[1]	MC26_ADDR	If IA32_MCG_CAP.CNT >26
46BH		IA32_MC26_MISC	MC26_MISC	If IA32_MCG_CAP.CNT >26
46CH		IA32_MC27_CTL	MC27_CTL	If IA32_MCG_CAP.CNT >27
46DH		IA32_MC27_STATUS	MC27_STATUS	If IA32_MCG_CAP.CNT >27
46EH		IA32_MC27_ADDR[1]	MC27_ADDR	If IA32_MCG_CAP.CNT >27
46FH		IA32_MC27_MISC	MC27_MISC	If IA32_MCG_CAP.CNT >27
470H		IA32_MC28_CTL	MC28_CTL	If IA32_MCG_CAP.CNT >28
471H		IA32_MC28_STATUS	MC28_STATUS	If IA32_MCG_CAP.CNT >28
472H		IA32_MC28_ADDR[1]	MC28_ADDR	If IA32_MCG_CAP.CNT >28

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
473H		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_CTLS	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
482H	1154	IA32_VMX_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If CPUID.01H:ECX.[5] = 1
483H	1155	IA32_VMX_EXIT_CTLS	**Capability Reporting Register of VM-exit Controls (R/O)** See Appendix A.4, "VM-Exit Controls."	If CPUID.01H:ECX.[5] = 1
484H	1156	IA32_VMX_ENTRY_CTLS	**Capability Reporting Register of VM-entry Controls (R/O)** See Appendix A.5, "VM-Entry Controls."	If CPUID.01H:ECX.[5] = 1
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_CR0_FIXED0	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0."	If CPUID.01H:ECX.[5] = 1
487H	1159	IA32_VMX_CR0_FIXED1	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0."	If CPUID.01H:ECX.[5] = 1
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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
48BH	1163	IA32_VMX_PROCBASED_CTLS2	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Appendix A.3.3, “Secondary Processor-Based VM-Execution Controls.”	If (CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTLS[63])
48CH	1164	IA32_VMX_EPT_VPID_CAP	**Capability Reporting Register of EPT and VPID (R/O)** See Appendix A.10, “VPID and EPT Capabilities.”	If (CPUID.01H:ECX.[5] && IA32_VMX_PROCBASED_CTLS[63] && (IA32_VMX_PROCBASED_CTLS2[33] || IA32_VMX_PROCBASED_CTLS2[37]))
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Appendix A.3.1, “Pin-Based VM-Execution Controls.”	If (CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Appendix A.3.2, “Primary Processor-Based VM-Execution Controls.”	If(CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Appendix A.4, “VM-Exit Controls.”	If(CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Appendix A.5, “VM-Entry Controls.”	If(CPUID.01H:ECX.[5] = 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])
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 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and 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	(R/W)	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 (RO).	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		0	Lock.	See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1	Reserved.	
		23:16	SGX_SVN_SINIT.	See Section 42.11.3, "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 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6:0	Reserved	
		MAXPHYADDR[3]-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**	
		5:4	Reserved,	
		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	Reserved, MBZ	
		13	**BranchEn**	
		17:14	**MTCFreq**	If (CPUID.(EAX=07H, ECX=0):EBX[3] = 1)
		18	Reserved, MBZ	
		22:19	**CYCThresh**	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		23	Reserved, MBZ	
		27:24	**PSBFreq**	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		31:28	Reserved, MBZ	
		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)
		63:48	Reserved, MBZ.	
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	
		31:6	Reserved, MBZ	
		48:32	PacketByteCnt	If (CPUID.(EAX=07H, ECX=0):EBX[1] > 3)
		63:49	**Reserved.**	
572H	1394	IA32_RTIT_CR3_MATCH	**Trace Filter CR3 Match Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		4:0	Reserved	
		63:5	CR3[63:5] value to match	
580H	1408	IA32_RTIT_ADDR0_A	**Region 0 Start Address (R/W)**	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
581H	1409	IA32_RTIT_ADDR0_B	**Region 0 End Address (R/W)**	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
582H	1410	IA32_RTIT_ADDR1_A	**Region 1 Start Address (R/W)**	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 18.12.4, "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.	
6E0H	1760	IA32_TSC_DEADLINE	**TSC Target of Local APIC's TSC Deadline Mode (R/W)**	If CPUID.01H:ECX.[24] = 1
770H	1904	IA32_PM_ENABLE	**Enable/disable HWP (R/W)**	If CPUID.06H:EAX.[7] = 1

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	HWP_ENABLE (R/W1-Once). See Section 14.4.2, “Enabling HWP”	If CPUID.06H:EAX.[7] = 1
		63:1	Reserved.	
771H	1905	IA32_HWP_CAPABILITIES	**HWP Performance Range Enumeration (RO)**	If CPUID.06H:EAX.[7] = 1
		7:0	Highest_Performance See Section 14.4.3, “HWP Performance Range and Dynamic Capabilities”	If CPUID.06H:EAX.[7] = 1
		15:8	Guaranteed_Performance See Section 14.4.3, “HWP Performance Range and Dynamic Capabilities”	If CPUID.06H:EAX.[7] = 1
		23:16	Most_Efficient_Performance See Section 14.4.3, “HWP Performance Range and Dynamic Capabilities”	If CPUID.06H:EAX.[7] = 1
		31:24	Lowest_Performance See Section 14.4.3, “HWP Performance Range and Dynamic Capabilities”	If CPUID.06H:EAX.[7] = 1
		63:32	Reserved.	
772H	1906	IA32_HWP_REQUEST_PKG	**Power Management Control Hints for All Logical Processors in a Package (R/W)**	If CPUID.06H:EAX.[11] = 1
		7:0	Minimum_Performance See Section 14.4.4, “Managing HWP”	If CPUID.06H:EAX.[11] = 1
		15:8	Maximum_Performance See Section 14.4.4, “Managing HWP”	If CPUID.06H:EAX.[11] = 1
		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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	
777H	1911	IA32_HWP_STATUS	**Log bits indicating changes to Guaranteed & excursions to Minimum (R/W)**	If CPUID.06H:EAX.[7] = 1
		0	Guaranteed_Performance_Change (R/WC0). See Section 14.4.5, “HWP Feedback”	If CPUID.06H:EAX.[7] = 1
		1	Reserved.	
		2	Excursion_To_Minimum (R/WC0). See Section 14.4.5, “HWP Feedback”	If CPUID.06H:EAX.[7] = 1
		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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
80BH	2059	IA32_X2APIC_EOI	**x2APIC EOI Register (W/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80DH	2061	IA32_X2APIC_LDR	**x2APIC Logical Destination Register (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80FH	2063	IA32_X2APIC_SIVR	**x2APIC Spurious Interrupt Vector Register (R/W)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
810H	2064	IA32_X2APIC_ISR0	**x2APIC In-Service Register Bits 31:0 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
811H	2065	IA32_X2APIC_ISR1	**x2APIC In-Service Register Bits 63:32 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
812H	2066	IA32_X2APIC_ISR2	**x2APIC In-Service Register Bits 95:64 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
813H	2067	IA32_X2APIC_ISR3	**x2APIC In-Service Register Bits 127:96 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
814H	2068	IA32_X2APIC_ISR4	**x2APIC In-Service Register Bits 159:128 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
815H	2069	IA32_X2APIC_ISR5	**x2APIC In-Service Register Bits 191:160 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
816H	2070	IA32_X2APIC_ISR6	**x2APIC In-Service Register Bits 223:192 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
817H	2071	IA32_X2APIC_ISR7	**x2APIC In-Service Register Bits 255:224 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
818H	2072	IA32_X2APIC_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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
81BH	2075	IA32_X2APIC_TMR3	**x2APIC Trigger Mode Register Bits 127:96 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81CH	2076	IA32_X2APIC_TMR4	**x2APIC Trigger Mode Register Bits 159:128 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81DH	2077	IA32_X2APIC_TMR5	**x2APIC Trigger Mode Register Bits 191:160 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
81EH	2078	IA32_X2APIC_TMR6	**x2APIC Trigger Mode Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1)
81FH	2079	IA32_X2APIC_TMR7	**x2APIC Trigger Mode Register Bits 255:224 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
820H	2080	IA32_X2APIC_IRR0	**x2APIC Interrupt Request Register Bits 31:0 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
821H	2081	IA32_X2APIC_IRR1	**x2APIC Interrupt Request Register Bits 63:32 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
822H	2082	IA32_X2APIC_IRR2	**x2APIC Interrupt Request Register Bits 95:64 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
823H	2083	IA32_X2APIC_IRR3	**x2APIC Interrupt Request Register Bits 127:96 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
824H	2084	IA32_X2APIC_IRR4	**x2APIC Interrupt Request Register Bits 159:128 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
825H	2085	IA32_X2APIC_IRR5	**x2APIC Interrupt Request Register Bits 191:160 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
826H	2086	IA32_X2APIC_IRR6	**x2APIC Interrupt Request Register Bits 223:192 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
827H	2087	IA32_X2APIC_IRR7	**x2APIC Interrupt Request Register Bits 255:224 (R/O)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
828H	2088	IA32_X2APIC_ESR	**x2APIC Error Status Register (R/W)**	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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)
		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.	
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 and 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**	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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)
C90H - D8FH		Reserved MSR Address Space for CAT Mask Registers	**See Section 17.17.3.1, "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		Reserved MSR Address Space for L2 CAT Mask Registers	**See Section 17.17.3.1, "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 0	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:12	**Base Address of Bound Directory.**	
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)**	
		63:9	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/0)**	If CPUID.06H:EAX.[13] = 1
		63:0	**Stall_Cycle_Cnt (R/W)** Stalled cycles due to HDC forced idle on this logical processor. See Section 14.5.4.1	If CPUID.06H:EAX.[13] = 1
4000_0000H - 4000_00FFH		Reserved MSR Address Space	**All existing and future processors will not implement MSR 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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		8	**IA-32e Mode Enable: IA32_EFER.LME (R/W)** Enables IA-32e mode operation.	
		9	Reserved.	
		10	**IA-32e Mode Active: IA32_EFER.LMA (R)** Indicates IA-32e mode is active when set.	
		11	**Execute Disable Bit Enable: IA32_EFER.NXE (R/W)**	
		63:12	Reserved.	
C000_0081H		IA32_STAR	**System Call Target Address (R/W)**	If CPUID.80000001:EDX.[29] = 1
C000_0082H		IA32_LSTAR	**IA-32e Mode System Call Target Address (R/W)**	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 (RW)	If CPUID.80000001H: EDX[27] = 1
		31:0	AUX: Auxiliary signature of TSC	
		63:32	Reserved.	

NOTES:

1. 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.
2. 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.
3. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

35.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 35-3 lists model-specific registers (MSRs) for Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table 35-3. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_0FH, see Table 35-1.

MSRs listed in Table 35-2 and Table 35-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. 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 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Unique	See Section 35.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Unique	See Section 35.22, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination." andTable 35-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.15, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Shared	**Platform ID (R)** See Table 35-2.
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 35-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location." and Table 35-2.
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 processor implements R/W.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		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.
		5		Reserved.
		6		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		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)**

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
3AH	58	MSR_FEATURE_CONTROL	Unique	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
		3	Unique	**SMRR Enable (R/WL)** When this bit is set and the lock bit is set 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: ▪ 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.
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 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	**BIOS Update Signature ID (RO)** See Table 35-2.
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.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		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 35-2.
C2H	194	IA32_PMC1	Unique	**Performance Counter Register** See Table 35-2.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333)
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
				266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.
		63:3		Reserved.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture:

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600)
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
				266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 35-2.
		11	Unique	**SMRR Capability Using MSR 0A0H and 0A1H (R)**
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		**L2 Hardware Enabled (RO)** 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 (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 35-2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
175H	373	IA32_SYSENTER_ESP	Unique	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 35-2.
179H	377	IA32_MCG_CAP	Unique	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		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	Unique	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 35-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 35-2.
198H	408	MSR_PERF_STATUS	Shared	
		15:0		Current Performance State Value.
		30:16		Reserved.
		31		XE Operation (R/O). If set, XE operation is enabled. Default is cleared.
		39:32		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		45		Reserved.
		46		Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.
		63:47		Reserved.
199H	409	IA32_PERF_CTL	Unique	See Table 35-2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_MODULATION	Unique	**Clock Modulation (R/W)** See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Unique	**Thermal Monitor Status (R/W)** See Table 35-2.
19DH	413	MSR_THERM2_CTL	Unique	
		15:0		Reserved.
		16		**TM_SELECT (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:16		Reserved.
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 35-2.
		2:1		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2.
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available (R)** See Table 35-2.
		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 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	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 (RO)** See Table 35-2.
		12	Shared	**Processor Event Based Sampling Unavailable (RO)** See Table 35-2.
		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 35-2.
		18	Shared	**ENABLE MONITOR FSM (R/W)** See Table 35-2.
		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 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	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: ▪ 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 35-2.
		23	Shared	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Unique	**XD Bit Disable (R/W)** See Table 35-2.
		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 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 IDA is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of IDA. If power-on default value is 1, IDA is available in the processor. If power-on default value is 0, IDA is not available.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	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 35-2
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	Unique	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Unique	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Unique	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Unique	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Unique	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Unique	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Unique	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Unique	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Unique	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Unique	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Unique	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Unique	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Unique	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Unique	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Unique	See Table 35-2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
20FH	527	IA32_MTRR_PHYSMASK7	Unique	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_ 00000	Unique	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Unique	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Unique	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Unique	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Unique	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Unique	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Unique	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Unique	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	See Table 35-2.
277H	631	IA32_PAT	Unique	See Table 35-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	**Default Memory Types (R/W)** See Table 35-2.
309H	777	IA32_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
309H	777	MSR_PERF_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0 (R/W)**
30AH	778	IA32_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30AH	778	MSR_PERF_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1 (R/W)**
30BH	779	IA32_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
30BH	779	MSR_PERF_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2 (R/W)**
345H	837	IA32_PERF_CAPABILITIES	Unique	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
345H	837	MSR_PERF_CAPABILITIES	Unique	RO. This applies to processors that do not support architectural perfmon version 2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		63:8		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38DH	909	MSR_PERF_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register (R/W)**
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 35-2. See Section 18.4.2, “Global Counter Control Facilities.”
38EH	910	MSR_PERF_GLOBAL_STATUS	Unique	See Section 18.4.2, “Global Counter Control Facilities.”
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 35-2. See Section 18.4.2, “Global Counter Control Facilities.”
38FH	911	MSR_PERF_GLOBAL_CTRL	Unique	See Section 18.4.2, “Global Counter Control Facilities.”
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 35-2. See Section 18.4.2, “Global Counter Control Facilities.”
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Unique	See Section 18.4.2, “Global Counter Control Facilities.”
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 35-2. See Section 18.4.4, “Precise Event Based Sampling (PEBS).”
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Unique	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
401H	1025	IA32_MC0_STATUS	Unique	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
402H	1026	IA32_MC0_ADDR	Unique	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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.”

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
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 ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC3_MISC	Unique	
414H	1044	IA32_MC5_CTL	Unique	
415H	1045	IA32_MC5_STATUS	Unique	
416H	1046	IA32_MC5_ADDR	Unique	
417H	1047	IA32_MC5_MISC	Unique	
419H	1045	IA32_MC6_STATUS	Unique	Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 15.3.2.2, "IA32_MCi_STATUS MSRS." and Chapter 23.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_ CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_ CTLS	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_CTLS	Unique	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. 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 35-2. 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 35-2. See Appendix A.9, "VMCS Enumeration."

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
48BH	1163	IA32_VMX_PROCBASED_ CTLS2	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 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."
107CC H		MSR_EMON_L3_CTR_CTL0	Unique	**GBUSQ Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CD H		MSR_EMON_L3_CTR_CTL1	Unique	**GBUSQ Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CE H		MSR_EMON_L3_CTR_CTL2	Unique	**GSNPQ Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CF H		MSR_EMON_L3_CTR_CTL3	Unique	**GSNPQ Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D0 H		MSR_EMON_L3_CTR_CTL4	Unique	**FSB Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D1 H		MSR_EMON_L3_CTR_CTL5	Unique	**FSB Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D2 H		MSR_EMON_L3_CTR_CTL6	Unique	**FSB Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D3 H		MSR_EMON_L3_CTR_CTL7	Unique	**FSB Event Control/Counter Register (R/W)** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D8 H		MSR_EMON_L3_GL_CTL	Unique	**L3/FSB Common Control Register (R/W)** Apply 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 35-2.
C000_ 0081H		IA32_STAR	Unique	**System Call Target Address (R/W)** See Table 35-2.

Table 35-3 MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
C000_ 0082H		IA32_LSTAR	Unique	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_ 0084H		IA32_FMASK	Unique	**System Call Flag Mask (R/W)** See Table 35-2.
C000_ 0100H		IA32_FS_BASE	Unique	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_ 0101H		IA32_GS_BASE	Unique	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0102H		IA32_KERNEL_GSBASE	Unique	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.

35.3 MSRS IN THE 45 NM AND 32 NM INTEL® ATOM™ PROCESSOR FAMILY

Table 35-4 lists model-specific registers (MSRs) for 45 nm and 32 nm Intel Atom processors, architectural MSR addresses are also included in Table 35-4. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1CH, 06_26H, 06_27H, 06_35H and 06_36H; see Table 35-1.

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 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 35.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Shared	See Section 35.22, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_ SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination." andTable 35-2
10H	16	IA32_TIME_STAMP_ COUNTER	Unique	See Section 17.15, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Shared	**Platform ID (R)** See Table 35-2.
17H	23	MSR_PLATFORM_ID	Shared	**Model Specific Platform ID (R)**
		7:0		Reserved.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		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 35-2.
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.
		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.**

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		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 35-2.
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: ▪ 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.
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.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
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 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Unique	**Performance counter register** See Table 35-2.
C2H	194	IA32_PMC1	Unique	**Performance Counter Register** See Table 35-2.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture:
		2:0		▪ 111B: 083 MHz (FSB 333) ▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Shared	**Memory Type Range Register (R)** See Table 35-2.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		**L2 Hardware Enabled (RO)** 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 (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 35-2.
179H	377	IA32_MCG_CAP	Unique	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		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	Unique	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 35-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 35-2.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
198H	408	MSR_PERF_STATUS	Shared	
		15:0		Current Performance State Value.
		39:16		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		63:45		Reserved.
199H	409	IA32_PERF_CTL	Unique	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	**Clock Modulation (R/W)** See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Unique	**Thermal Monitor Status (R/W)** See Table 35-2.
19DH	413	MSR_THERM2_CTL	Shared	
		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 35-2.
		2:1		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2. Default value is 0.
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available (R)** See Table 35-2.
		8		Reserved.
		9		Reserved.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	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 (RO)** See Table 35-2.
		12	Shared	**Processor Event Based Sampling Unavailable (RO)** See Table 35-2.
		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 35-2.
		18	Shared	**ENABLE MONITOR FSM (R/W)** See Table 35-2.
		19		Reserved.
		20	Shared	**Enhanced Intel SpeedStep Technology Select Lock (R/WO)** When set, this bit causes the following bits to become read-only: ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved.
		22	Unique	**Limit CPUID Maxval (R/W)** See Table 35-2.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		23	Shared	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Unique	**XD Bit Disable (R/W)** See Table 35-2.
		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 35-2.
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 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Shared	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Shared	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Shared	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Shared	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Shared	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Shared	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Shared	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Shared	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Shared	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Shared	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Shared	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Shared	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Shared	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Shared	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Shared	See Table 35-2.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
250H	592	IA32_MTRR_FIX64K_ 00000	Shared	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Shared	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Shared	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Shared	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Shared	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Shared	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Shared	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 35-2.
277H	631	IA32_PAT	Unique	See Table 35-2.
309H	777	IA32_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_ STATUS	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_ CTRL	Unique	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 35-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
402H	1026	IA32_MC0_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_ CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
482H	1154	IA32_VMX_PROCBASED_CTLS	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_CTLS	Unique	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, “VM-Exit Controls.”
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, “VM-Entry Controls.”
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. See Appendix A.6, “Miscellaneous Data.”
486H	1158	IA32_VMX_CR0_FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.7, “VMX-Fixed Bits in CR0.”
487H	1159	IA32_VMX_CR0_FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, “VMX-Fixed Bits in CR0.”
488H	1160	IA32_VMX_CR4_FIXED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. 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 35-2. 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 35-2. See Appendix A.9, “VMCS Enumeration.”
48BH	1163	IA32_VMX_PROCBASED_CTLS2	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 35-2. See Section 18.12.4, “Debug Store (DS) Mechanism.”
C000_0080H		IA32_EFER	Unique	**Extended Feature Enables** See Table 35-2.
C000_0081H		IA32_STAR	Unique	**System Call Target Address (R/W)** See Table 35-2.

Table 35-4 MSRs in 45 nm and 32 nm Intel® Atom™ Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
C000_ 0082H		IA32_LSTAR	Unique	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_ 0084H		IA32_FMASK	Unique	**System Call Flag Mask (R/W)** See Table 35-2.
C000_ 0100H		IA32_FS_BASE	Unique	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_ 0101H		IA32_GS_BASE	Unique	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0102H		IA32_KERNEL_GSBASE	Unique	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.

...

35.4 MSRS IN INTEL PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 35-6 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 35-1. The MSRs listed in Table 35-6 are also common to processors based on the Airmont microarchitecture and newer microarchitectures for next generation Intel Atom processors.

Table 35-7 lists MSRs common to processors based on the Silvermont and Airmont microarchitectures, but not newer microarchitectures.

Table 35-8, Table 35-9, and Table 35-10 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 pair of processor cores in the physical package. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 35.22, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Module	See Section 35.22, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_ SIZE	Core	See Section 8.10.5, “Monitor/Mwait Address Range Determination.” andTable 35-2
10H	16	IA32_TIME_STAMP_ COUNTER	Core	See Section 17.15, “Time-Stamp Counter,” and see Table 35-2.

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
1BH	27	IA32_APIC_BASE	Core	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
2AH	42	MSR_EBL_CR_POWERON	Module	**Processor Hard Power-On Configuration (R/W)** Writes ignored
		63:0		Reserved **(R/O)**
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 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Core	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Core	**Performance counter register** See Table 35-2.
C2H	194	IA32_PMC1	Core	**Performance Counter Register** See Table 35-2.
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 (RW)** See Table 35-2.
E8H	232	IA32_APERF	Core	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
FEH	254	IA32_MTRRCAP	Core	**Memory Type Range Register (R)** See Table 35-2.
13CH	52	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 instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
174H	372	IA32_SYSENTER_CS	Core	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Core	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Core	See Table 35-2.
179H	377	IA32_MCG_CAP	Core	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Core	
		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	Core	See Table 35-2.
		7:0		**Event Select**
		15:8		**UMask**

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-2.
198H	408	IA32_PERF_STATUS	Module	See Table 35-2.
199H	409	IA32_PERF_CTL	Core	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Core	**Clock Modulation (R/W)** See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Core	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status (R/W)** See Table 35-2.
1A2H	418	MSR_ TEMPERATURE_TARGET	Package	
		15:0		Reserved.
		23:16		**Temperature Target (R)** The default thermal throttling or PROCHOT# activation temperature in degree 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 35-2.

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Core	**Debug Control (R/W)** See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Core	**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	Core	**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 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 35-2.

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 35-2.
277H	631	IA32_PAT	Core	See Table 35-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	**Default Memory Types (R/W)** See Table 35-2.
309H	777	IA32_FIXED_CTR0	Core	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Core	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Core	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Core	See Table 35-2. See Section 17.4.1, “IA32_DEBUGCTL MSR.”
38DH	909	IA32_FIXED_CTR_CTRL	Core	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Core	See Table 35-2. See Section 18.4.2, “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_MC0_CTL	Module	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
401H	1025	IA32_MC0_STATUS	Module	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
402H	1026	IA32_MC0_ADDR	Module	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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.”

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Core	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	Core	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
482H	1154	IA32_VMX_PROCBASED_CTLS	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_CTLS	Core	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Core	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Core	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. 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 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Core	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. 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 35-2. 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 35-2. 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 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	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 35-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Core	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Table 35-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Core	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Table 35-2

Table 35-6 MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom Processors

Address		Register Name	Scope	Bit Description
Hex	Dec			
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Core	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Table 35-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Core	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Table 35-2
491H	1169	IA32_VMX_FMFUNC	Core	**Capability Reporting Register of VM-function Controls (R/O)** See Table 35-2
4C1H	1217	IA32_A_PMC0	Core	See Table 35-2.
4C2H	1218	IA32_A_PMC1	Core	See Table 35-2.
600H	1536	IA32_DS_AREA	Core	**DS Save Area (R/W)** See Table 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."
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 35-2
C000_0080H		IA32_EFER	Core	**Extended Feature Enables** See Table 35-2.
C000_0081H		IA32_STAR	Core	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Core	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_0084H		IA32_FMASK	Core	**System Call Flag Mask (R/W)** See Table 35-2.
C000_0100H		IA32_FS_BASE	Core	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_0101H		IA32_GS_BASE	Core	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Core	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.
C000_0103H		IA32_TSC_AUX	Core	**AUXILIARY TSC Signature. (R/W)** See Table 35-2

Table 35-7 lists model-specific registers (MSRs) that are common to Intel® Atom™ processors based on the Silvermont and Airmont microarchitectures but not newer microarchitectures.

Table 35-7 MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	**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 35-2
		63:33		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Core	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
		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: ▪ 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.

Table 35-7 MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
67H	103	MSR_ LASTBRANCH_7_TO_IP	Core	**Last Branch Record 7 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
CDH	205	MSR_FSB_FREQ	Module	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Silvermont microarchitecture:
		2:0		▪ 100B: 080.0 MHz ▪ 000B: 083.3 MHz ▪ 001B: 100.0 MHz ▪ 010B: 133.3 MHz ▪ 011B: 116.7 MHz
		63:3		Reserved.
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.

Table 35-7 MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
11EH	281	MSR_BBL_CR_CTL3	Module	
		0		**L2 Hardware Enabled (RO)** 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 (RO)** 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 35-2.
		2:1		Reserved.
		3	Module	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2. Default value is 0.
		6:4		Reserved.
		7	Core	**Performance Monitoring Available (R)** See Table 35-2.
		10:8		Reserved.
		11	Core	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Core	**Processor Event Based Sampling Unavailable (RO)** See Table 35-2.

Table 35-7 MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:13		Reserved.
		16	Module	**Enhanced Intel SpeedStep Technology Enable (R/W)** See Table 35-2.
		18	Core	**ENABLE MONITOR FSM (R/W)** See Table 35-2.
		21:19		Reserved.
		22	Core	**Limit CPUID Maxval (R/W)** See Table 35-2.
		23	Module	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Core	**XD Bit Disable (R/W)** See Table 35-2.
		37:35		Reserved.
		38	Module	**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 power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1C8H	456	MSR_LBR_SELECT	Core	**Last Branch Record Filtering Select Register (R/W)** See Section 17.7.2, "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.

Table 35-7 MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1C9H	457	MSR_LASTBRANCH_TOS	Core	**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.
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Core	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Core	See Table 35-2. See Section 18.4.4, "Precise 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.
		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/0)** 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.

35.4.1 MSRs with Model-Specific Behavior in the Silvermont Microarchitecture

Table 35-8 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 35-10 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor C2000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_4DH).

Table 35-10 Specific MSRs Supported by Intel® Atom™ Processor C2000 Series with CPUID Signature 06_4DH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A4H	420	MSR_MISC_FEATURE_CONTROL		**Miscellaneous Feature Control (R/W)**

Table 35-10 Specific MSRs Supported by Intel® Atom™ Processor C2000 Series (Contd.)with CPUID Signature

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 (RW)**
		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 14.9.1, "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 35-10 Specific MSRs Supported by Intel® Atom™ Processor C2000 Series (Contd.)with CPUID Signature

Register Address		Register Name	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)** See Section 14.9.3, "Package RAPL Domain."
66EH	1646	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameter (R/0)**
		14:0		Thermal Spec Power. (R/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
		63:15		Reserved

35.4.2 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 35-6, Table 35-7, Table 35-8, and Table 35-11. These processors have a CPUID signature with DisplayFamily_DisplayModel including 06_4CH; see Table 35-1.

Table 35-11 MSRs in Intel Atom Processors Based on the Airmont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Airmont microarchitecture:
		3:0		▪ 0000B: 083.3 MHz ▪ 0001B: 100.0 MHz ▪ 0010B: 133.3 MHz ▪ 0011B: 116.7 MHz ▪ 0100B: 080.0 MHz ▪ 0101B: 093.3 MHz ▪ 0110B: 090.0 MHz ▪ 0111B: 088.9 MHz ▪ 10sure00B: 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.

Table 35-11 MSRs in Intel Atom Processors Based on the Airmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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, lock 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.
		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_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)**
		14:0		PP0 Power Limit #1. (R/W) See Section 14.9.4, “PP0/PP1 RAPL Domains.” and MSR_RAPL_POWER_UNIT in Table 35-8.
		15		Enable Power Limit #1. (R/W) See Section 14.9.4, “PP0/PP1 RAPL Domains.”

Table 35-11 MSRs in Intel Atom Processors Based on the Airmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		16		Reserved
		23:17		Time Window for Power Limit #1. (R/W) Specifies the time duration over which the average power must remain below PP0_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.
		63:24		Reserved

35.5 MSRS IN NEXT GENERATION INTEL ATOM PROCESSORS

Next Generation Intel Atom processors are based on the Goldmont microarchitecture. These processors support MSRs listed in Table 35-6 and Table 35-12. These processors have a CPUID signature with DisplayFamily_DisplayModel including 06_5CH; see Table 35-1.

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 pair of processor cores in the physical package. "Package" means all processor cores in the physical package share the same MSR or bit interface.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	**Model Specific Platform ID (R)**
		49:0		Reserved.
		52:50		**See** Table 35-2
		63:33		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Core	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
		0		**Lock (R/WL)**

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-2.
C3H	195	IA32_PMC2	Core	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Core	**Performance Counter Register** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The 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 Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and 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 an temperature offset.
		39:31		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, 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.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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, lock bits 15:0 of this register until next reset.
		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 35-2.
189H	393	IA32_PERFEVTSEL3	Core	See Table 35-2.
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 35-2.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:1		Reserved.
		3	Package	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2. Default value is 1.
		6:4		Reserved.
		7	Core	**Performance Monitoring Available (R)** See Table 35-2.
		10:8		Reserved.
		11	Core	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Core	**Processor Event Based Sampling Unavailable (RO)** See Table 35-2.
		15:13		Reserved.
		16	Package	**Enhanced Intel SpeedStep Technology Enable (R/W)** See Table 35-2.
		18	Core	**ENABLE MONITOR FSM (R/W)** See Table 35-2.
		21:19		Reserved.
		22	Core	**Limit CPUID Maxval (R/W)** See Table 35-2.
		23	Package	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Core	**XD Bit Disable (R/W)** See Table 35-2.
		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 power-on default value is 1, turbo mode is available in the processor. If 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)**

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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	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.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode by Core Groups (RW)** **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 number of active cores is less or equal to Group 0 threshold.
		15:8	Package	**Maximum Ratio Limit for Active cores in Group 1** Maximum turbo ratio limit when number of active cores is less or equal to Group 1 threshold and greater than Group 0 threshold.
		23:16	Package	**Maximum Ratio Limit for Active cores in Group 2** Maximum turbo ratio limit when number of active cores is less or equal to Group 2 threshold and greater than Group 1 threshold.
		31:24	Package	**Maximum Ratio Limit for Active cores in Group 3** Maximum turbo ratio limit when number of active cores is less or equal to Group 3 threshold and greater than Group 2 threshold.
		39:32	Package	**Maximum Ratio Limit for Active cores in Group 4** Maximum turbo ratio limit when number of active cores is less or equal to Group 4 threshold and greater than Group 3 threshold.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		47:40	Package	**Maximum Ratio Limit for Active cores in Group 5** Maximum turbo ratio limit when number of active cores is less or equal to Group 5 threshold and greater than Group 4 threshold.
		55:48	Package	**Maximum Ratio Limit for Active cores in Group 6** Maximum turbo ratio limit when number of active cores is less or equal to Group 6 threshold and greater than Group 5 threshold.
		63:56	Package	**Maximum Ratio Limit for Active cores in Group 7** Maximum turbo ratio limit when number of active cores is less or equal to Group 7 threshold and greater than Group 6 threshold.
1AEH	430	MSR_TURBO_GROUP_CORECNT	Package	**Group Size of Active Cores for Turbo Mode Operation (RW)** **Writes of 0 threshold is ignored**
		7:0	Package	**Group 0 Core Count Threshold** Maximum number of active cores to operate under Group 0 Max Turbo Ratio limit.
		15:8	Package	**Group 1 Core Count Threshold** Maximum number of active cores to operate under Group 1 Max Turbo Ratio limit. Must be greater than Group 0 Core Count.
		23:16	Package	**Group 2 Core Count Threshold** Maximum number of active cores to operate under Group 2 Max Turbo Ratio limit. Must be greater than Group 1 Core Count.
		31:24	Package	**Group 3 Core Count Threshold** Maximum number of active cores to operate under Group 3 Max Turbo Ratio limit. Must be greater than Group 2 Core Count.
		39:32	Package	**Group 4 Core Count Threshold** Maximum number of active cores to operate under Group 4 Max Turbo Ratio limit. Must be greater than Group 3 Core Count.
		47:40	Package	**Group 5 Core Count Threshold** Maximum number of active cores to operate under Group 5 Max Turbo Ratio limit. Must be greater than Group 4 Core Count.
		55:48	Package	**Group 6 Core Count Threshold** Maximum number of active cores to operate under Group 6 Max Turbo Ratio limit. Must be greater than Group 5 Core Count.
		63:56	Package	**Group 7 Core Count Threshold** Maximum number of active cores to operate under Group 7 Max Turbo Ratio limit. Must be greater than 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 17.7.2, “Filtering of Last Branch Records.”
		0		**CPL_EQ_0**

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		1		**CPL_NEQ_0**
		2		**JCC**
		3		**NEAR_REL_CALL**
		4		**NEAR_IND_CALL**
		5		**NEAR_RET**
		6		**NEAR_IND_JMP**
		7		**NEAR_REL_JMP**
		8		**FAR_BRANCH**
		9		**EN_CALL_STACK**
		63:10		Reserved.
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 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Core	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Core	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Core	See Table 35-2.
280H	640	IA32_MC0_CTL2	Module	See Table 35-2.
281H	641	IA32_MC1_CTL2	Module	See Table 35-2.
282H	642	IA32_MC2_CTL2	Core	See Table 35-2.
283H	643	IA32_MC3_CTL2	Module	See Table 35-2.
284H	644	IA32_MC4_CTL2	Package	See Table 35-2.
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
300H	768	MSR_SGXOWNER0	Package	**Lower 64 Bit OwnerEpoch Component of SGX Key (RO).**
		63:0		**Low 64 bits of an 128-bit external entropy value for key derivation of an enclave.**
301H	769	MSR_SGXOWNER1	Package	**Upper 64 Bit OwnerEpoch Component of SGX Key (RO).**

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-2. See Section 18.2.2.3, "Full-Width Writes to Performance Counter Registers."
		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.**
		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 35-2. See Section 18.2.2.3, "Full-Width Writes to Performance Counter Registers."
		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.**

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-2. See Section 18.2.2.3, "Full-Width Writes to Performance Counter Registers."
		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		**Set 1 to cause CondChgd = 1**
392H	914	IA32_PERF_GLOBAL_INUSE		See Table 35-2.
3F1H	1009	MSR_PEBS_ENABLE	Core	See Table 35-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)
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 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	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.
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.
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 35-2.
4C4H	1220	IA32_A_PMC3	Core	See Table 35-2.
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-RWO)** When set to '1' locks this register from further changes
		1		Reserved
		2		**SMM_Code_Chk_En (SMM-RW)** This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	**SMM Delayed (SMM-RO)** Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	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 0FFFH. 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 (RO).**
		0		**Lock.** See Section 42.11.3, “Interactions with Authenticated Code Modules (ACMs)”
		15:1		Reserved.
		23:16		**SGX_SVN_SINIT.** See Section 42.11.3, “Interactions with Authenticated Code Modules (ACMs)”
		63:24		Reserved.
560H	1376	IA32_RTIT_OUTPUT_BASE	Core	**Trace Output Base Register (R/W).** See Table 35-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Core	**Trace Output Mask Pointers Register (R/W).** See Table 35-2.
570H	1392	IA32_RTIT_CTL	Core	**Trace Control Register (R/W)**
		0		**TraceEn**
		1		**CYCEn**
		2		**OS**
		3		**User**
		6:4		Reserved, MBZ
		7		**CR3 filter**
		8		**ToPA; writing 0 will #GP if also setting TraceEn**

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		9		**MTCEn**
		10		**TSCEn**
		11		**DisRETC**
		12		Reserved, MBZ
		13		**BranchEn**
		17:14		**MTCFreq**
		18		Reserved, MBZ
		22:19		**CYCThresh**
		23		Reserved, MBZ
		27:24		**PSBFreq**
		31:28		Reserved, MBZ
		35:32		**ADDR0_CFG**
		39:36		**ADDR1_CFG**
		63:40		Reserved, MBZ.
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. MBZ
		48:32		**PacketByteCnt**
		63:49		Reserved, MBZ.
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_ADDR0_A	Core	**Region 0 Start Address (R/W)**
		63:0		See Table 35-2.
581H	1409	IA32_RTIT_ADDR0_B	Core	**Region 0 End Address (R/W)**
		63:0		See Table 35-2.
582H	1410	IA32_RTIT_ADDR1_A	Core	**Region 1 Start Address (R/W)**
		63:0		See Table 35-2.
583H	1411	IA32_RTIT_ADDR1_B	Core	**Region 1 End Address (R/W)**
		63:0		See Table 35-2.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)** See Section 14.9.1, "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 35-18 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.
60BH	1547	MSR_PKGC_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 transition to 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.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-18 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.
60CH	1548	MSR_PKGC_IRTL2	Package	**Package C7 Interrupt Response Limit 2 (R/W)** This MSR defines the interrupt response time limit used by the processor to manage transition to 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 35-18 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.
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 14.9.3, “Package RAPL Domain.”
611H	1553	MSR_PKG_ENERGY_STATUS	Package	**PKG Energy Status (R/O)** See Section 14.9.3, “Package RAPL Domain.”
613H	1555	MSR_PKG_PERF_STATUS	Package	**PKG Perf Status (R/O)** See Section 14.9.3, “Package RAPL Domain.”

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
614H	1556	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameters (R/W)**
		14:0		**Thermal Spec Power (R/W)** See Section 14.9.3, "Package RAPL Domain."
		15		Reserved.
		30:16		**Minimum Power (R/W)** See Section 14.9.3, "Package RAPL Domain."
		31		Reserved.
		46:32		**Maximum Power (R/W)** See Section 14.9.3, "Package RAPL Domain."
		47		Reserved.
		54:48		**Maximum Time Window (R/W)** Specified by 2^Y * (1.0 + Z/4.0) * Time_Unit, where "Y" is the unsigned integer value represented. by bits 52:48, "Z" is an unsigned integer represented by bits 54:53. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT
		63:55		Reserved.
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names,
		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.
639H	1593	MSR_PP0_ENERGY_STATU S	Package	**PP0 Energy Status (R/O)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATU S	Package	**PP1 Energy Status (R/O)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
64CH	1612	MSR_TURBO_ACTIVATION_ RATIO	Package	**ConfigTDP Control (R/W)**
		7:0		**MAX_NON_TURBO_RATIO (RW/L)** System BIOS can program this field.
		30:8		Reserved.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		31		**TURBO_ACTIVATION_RATIO_Lock (RW/L)** When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	**Indicator of Frequency Clipping in Processor Cores (R/W)** **(frequency refers to processor core frequency)**
		0		**PROCHOT Status (R0)** When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		**Thermal Status (R0)** When set, frequency is reduced below the operating system request due to a thermal event.
		2		**Package-Level Power Limiting PL1 Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		3		**Package-Level PL2 Power Limiting Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		8:4		Reserved.
		9		**Core Power Limiting Status (R0)** When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		11		**Max Turbo Limit Status (R0)** When set, frequency is reduced below the operating system request due to multi-core turbo limits.
		12		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		13		**Turbo Transition Attenuation Status (R0)** When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		14		**Maximum Efficiency Frequency Status (R0)** When set, frequency is reduced below the maximum efficiency frequency.
		15		**Reserved**

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	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		**Package-Level PL1 Power Limiting Log** When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		**Package-Level PL2 Power Limiting Log** When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		24:20		Reserved.
		25		**Core Power Limiting Log** When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		**VR Therm Alert Log** When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		**Max Turbo Limit Log** When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		**Electrical Design Point Log** When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		**Turbo Transition Attenuation Log** When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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: ▪ 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.
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.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 19From 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.
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.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 also: ▪ 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.
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.

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
6CFH	1743	MSR_ LASTBRANCH_15_TO_IP	Core	**Last Branch Record 15 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D0H	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 19To 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.
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) See x2APIC Specification.
803H	2051	IA32_X2APIC_VERSION	Core	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Core	x2APIC Task Priority register (R/W)

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
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)
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)

Table 35-12 MSRs in Next Generation Intel Atom Processors Based on the Goldmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
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**
		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 35-2.
DA0H	3488	IA32_XSS	Core	See Table 35-2.
See Table 35-6, and Table 35-12 for MSR definitions applicable to processors with CPUID signature 06_5CH.				

35.6 MSRS IN THE INTEL® MICROARCHITECTURE CODE NAME NEHALEM

Table 35-13 lists model-specific registers (MSRs) that are common for Intel® microarchitecture code name Nehalem. These include 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 35-1. Additional MSRs specific to

06_1AH, 06_1EH, 06_1FH are listed in Table 35-14. 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 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 35.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 35.22, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 35-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.15, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Package	**Platform ID (R)** See Table 35-2.
17H	23	MSR_PLATFORM_ID	Package	**Model Specific Platform ID (R)**
		49:0		Reserved.
		52:50		See Table 35-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
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 35-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	**BIOS Update Trigger Register (W)** See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Thread	**Performance Counter Register** See Table 35-2.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C2H	194	IA32_PMC1	Thread	**Performance Counter Register** See Table 35-2.
C3H	195	IA32_PMC2	Thread	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Thread	**Performance Counter Register** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	see http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The 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 Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDC-TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDC/TDP Limits for Turbo mode are programmable, and 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)** The is the minimum ratio (maximum efficiency) that the processor can operates, 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.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-sate support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		23:16		Reserved.
		24		**Interrupt filtering enable (R/W)** When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.
		25		**C3 state auto demotion enable (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		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.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	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_PERFEVTSEL0	Thread	See Table 35-2.
		7:0		**Event Select**
		15:8		**UMask**
		16		**USR**
		17		**OS**
		18		**Edge**
		19		**PC**
		20		**INT**
		21		**AnyThread**
		22		**EN**
		23		**INV**
		31:24		**CMASK**
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 35-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 35-2.
198H	408	IA32_PERF_STATUS	Core	See Table 35-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	**Clock Modulation (R/W)** See Table 35-2. 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.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
19BH	411	IA32_THERM_INTERRUPT	Core	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status (R/W)** See Table 35-2.
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 35-2.
		2:1		Reserved.
		3	Thread	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2. Default value is 1.
		6:4		Reserved.
		7	Thread	**Performance Monitoring Available (R)** See Table 35-2.
		10:8		Reserved.
		11	Thread	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Thread	**Processor Event Based Sampling Unavailable (RO)** See Table 35-2.
		15:13		Reserved.
		16	Package	**Enhanced Intel SpeedStep Technology Enable (R/W)** See Table 35-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 35-2.
		21:19		Reserved.
		22	Thread	**Limit CPUID Maxval (R/W)** See Table 35-2.
		23	Thread	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Thread	**XD Bit Disable (R/W)** See Table 35-2.
		37:35		Reserved.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1A2H	418	MSR_ TEMPERATURE_TARGET	Thread	
		15:0		Reserved.
		23:16		**Temperature Target (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree 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		See http://biosbits.org.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	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, and a value = 1 indicates 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, and a value = 1 indicates active.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** **RO** if MSR_PLATFORM_INFO.[28] = 0, **RW** 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 17.7.2, "Filtering of Last Branch Records."

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-2.
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 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
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 35-2.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 35-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_ 00000	Thread	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Thread	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Thread	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 35-2.
277H	631	IA32_PAT	Thread	See Table 35-2.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
280H	640	IA32_MC0_CTL2	Package	See Table 35-2.
281H	641	IA32_MC1_CTL2	Package	See Table 35-2.
282H	642	IA32_MC2_CTL2	Core	See Table 35-2.
283H	643	IA32_MC3_CTL2	Core	See Table 35-2.
284H	644	IA32_MC4_CTL2	Core	See Table 35-2.
285H	645	IA32_MC5_CTL2	Core	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	**Default Memory Types (R/W)** See Table 35-2.
309H	777	IA32_FIXED_CTR0	Thread	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Thread	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Thread	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		11:8		PEBS_REC_FORMAT. See Table 35-2.
		12		SMM_FREEZE. See Table 35-2.
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Thread	**(RO)**
		61		**UNC_Ovf** Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2. See Section 18.4.2, “Global Counter Control Facilities.”
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Thread	**(R/W)**
		61		**CLR_UNC_Ovf** Set 1 to clear UNC_Ovf.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.7.1.1, “Precise 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)
		63:36		Reserved.
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 18.7.1.2, “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.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
400H	1024	IA32_MC0_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MC0_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 ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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."

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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."
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 ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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."

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	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_CTLS	Thread	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. 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 35-2. 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 35-2. 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 35-2. 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 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	**Capability Reporting Register of VMCS Field Enumeration (R/O).** See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	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 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."
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: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.7.1 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.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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) See x2APIC Specification.
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode register bits [223:192] (R/O)
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)

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 35-2.
C000_0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_0084H		IA32_FMASK	Thread	**System Call Flag Mask (R/W)** See Table 35-2.
C000_0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Thread	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.

Table 35-13 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C000_ 0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature. (R/W)** See Table 35-2 and Section 17.15.2, "IA32_TSC_AUX Register and RDTSCP Support."

...

35.6.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

Intel Xeon Processor 7500 series support MSRs listed in Table 35-13 (except MSR address 1ADH) and additional model-specific registers listed in Table 35-15. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2EH.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series

Register Address		Register Name	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 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
294H	660	IA32_MC20_CTL2	Package	See Table 35-2.
295H	661	IA32_MC21_CTL2	Package	See Table 35-2.
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."
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."

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.”
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.”

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_SEL0	Package	Uncore B-box 0 perfmon event select MSR.
C31H	3121	MSR_B0_PMON_CTR0	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_CTR1	Package	Uncore B-box 0 perfmon counter MSR.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C34H	3124	MSR_B0_PMON_EVNT_SEL2	Package	Uncore B-box 0 perfmon event select MSR.
C35H	3125	MSR_B0_PMON_CTR2	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_CTR3	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_SEL0	Package	Uncore S-box 0 perfmon event select MSR.
C51H	3153	MSR_S0_PMON_CTR0	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_CTR1	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_CTR2	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_CTR3	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_SEL0	Package	Uncore B-box 1 perfmon event select MSR.
C71H	3185	MSR_B1_PMON_CTR0	Package	Uncore B-box 1 perfmon counter MSR.
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR.
C73H	3187	MSR_B1_PMON_CTR1	Package	Uncore B-box 1 perfmon counter MSR.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	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 1vperfmon 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_SEL0	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.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_SEL0	Package	Uncore M-box 0 perfmon event select MSR.
CB1H	3249	MSR_M0_PMON_CTR0	Package	Uncore M-box 0 perfmon counter MSR.
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.
CD0H	3280	MSR_S1_PMON_EVNT_SEL0	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.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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_CTR0	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_CTR1	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_CTR2	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_CTR3	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_CTR4	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_CTR5	Package	Uncore M-box 1 perfmon counter MSR.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_CTR0	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_CTR1	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_CTR2	Package	Uncore C-box 0 perfmon counter MSR.
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_SEL0	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.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_SEL0	Package	Uncore C-box 2 perfmon event select MSR.
D51H	3409	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter MSR.
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.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_SEL0	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.
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_SEL0	Package	Uncore C-box 1 perfmon event select MSR.
D91H	3473	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter MSR.
D92H	3474	MSR_C1_PMON_EVNT_SEL1	Package	Uncore C-box 1 perfmon event select MSR.
D93H	3475	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter MSR.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D94H	3476	MSR_C1_PMON_EVNT_SEL2	Package	Uncore C-box 1 perfmon event select MSR.
D95H	3477	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter MSR.
D96H	3478	MSR_C1_PMON_EVNT_SEL3	Package	Uncore C-box 1 perfmon event select MSR.
D97H	3479	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter MSR.
D98H	3480	MSR_C1_PMON_EVNT_SEL4	Package	Uncore C-box 1 perfmon event select MSR.
D99H	3481	MSR_C1_PMON_CTR4	Package	Uncore C-box 1 perfmon counter MSR.
D9AH	3482	MSR_C1_PMON_EVNT_SEL5	Package	Uncore C-box 1 perfmon event select MSR.
D9BH	3483	MSR_C1_PMON_CTR5	Package	Uncore C-box 1 perfmon counter MSR.
DA0H	3488	MSR_C5_PMON_BOX_CTRL	Package	Uncore C-box 5 perfmon local box control MSR.
DA1H	3489	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon local box status MSR.
DA2H	3490	MSR_C5_PMON_BOX_OVF_CTRL	Package	Uncore C-box 5 perfmon local box overflow control MSR.
DB0H	3504	MSR_C5_PMON_EVNT_SEL0	Package	Uncore C-box 5 perfmon event select MSR.
DB1H	3505	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter MSR.
DB2H	3506	MSR_C5_PMON_EVNT_SEL1	Package	Uncore C-box 5 perfmon event select MSR.
DB3H	3507	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter MSR.
DB4H	3508	MSR_C5_PMON_EVNT_SEL2	Package	Uncore C-box 5 perfmon event select MSR.
DB5H	3509	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter MSR.
DB6H	3510	MSR_C5_PMON_EVNT_SEL3	Package	Uncore C-box 5 perfmon event select MSR.
DB7H	3511	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter MSR.
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.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_SEL0	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_SEL0	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.
DF3H	3571	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter MSR.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DF4H	3572	MSR_C7_PMON_EVNT_SEL2	Package	Uncore C-box 7 perfmon event select MSR.
DF5H	3573	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter MSR.
DF6H	3574	MSR_C7_PMON_EVNT_SEL3	Package	Uncore C-box 7 perfmon event select MSR.
DF7H	3575	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter MSR.
DF8H	3576	MSR_C7_PMON_EVNT_SEL4	Package	Uncore C-box 7 perfmon event select MSR.
DF9H	3577	MSR_C7_PMON_CTR4	Package	Uncore C-box 7 perfmon counter MSR.
DFAH	3578	MSR_C7_PMON_EVNT_SEL5	Package	Uncore C-box 7 perfmon event select MSR.
DFBH	3579	MSR_C7_PMON_CTR5	Package	Uncore C-box 7 perfmon counter MSR.
E00H	3584	MSR_R0_PMON_BOX_CTRL	Package	Uncore R-box 0 perfmon local box control MSR.
E01H	3585	MSR_R0_PMON_BOX_STATUS	Package	Uncore R-box 0 perfmon local box status MSR.
E02H	3586	MSR_R0_PMON_BOX_OVF_CTRL	Package	Uncore R-box 0 perfmon local box overflow control MSR.
E04H	3588	MSR_R0_PMON_IPERF0_P0	Package	Uncore R-box 0 perfmon IPERF0 unit Port 0 select MSR.
E05H	3589	MSR_R0_PMON_IPERF0_P1	Package	Uncore R-box 0 perfmon IPERF0 unit Port 1 select MSR.
E06H	3590	MSR_R0_PMON_IPERF0_P2	Package	Uncore R-box 0 perfmon IPERF0 unit Port 2 select MSR.
E07H	3591	MSR_R0_PMON_IPERF0_P3	Package	Uncore R-box 0 perfmon IPERF0 unit Port 3 select MSR.
E08H	3592	MSR_R0_PMON_IPERF0_P4	Package	Uncore R-box 0 perfmon IPERF0 unit Port 4 select MSR.
E09H	3593	MSR_R0_PMON_IPERF0_P5	Package	Uncore R-box 0 perfmon IPERF0 unit Port 5 select MSR.
E0AH	3594	MSR_R0_PMON_IPERF0_P6	Package	Uncore R-box 0 perfmon IPERF0 unit Port 6 select MSR.
E0BH	3595	MSR_R0_PMON_IPERF0_P7	Package	Uncore R-box 0 perfmon IPERF0 unit Port 7 select MSR.
E0CH	3596	MSR_R0_PMON_QLX_P0	Package	Uncore R-box 0 perfmon QLX unit Port 0 select MSR.
E0DH	3597	MSR_R0_PMON_QLX_P1	Package	Uncore R-box 0 perfmon QLX unit Port 1 select MSR.
E0EH	3598	MSR_R0_PMON_QLX_P2	Package	Uncore R-box 0 perfmon QLX unit Port 2 select MSR.
E0FH	3599	MSR_R0_PMON_QLX_P3	Package	Uncore R-box 0 perfmon QLX unit Port 3 select MSR.
E10H	3600	MSR_R0_PMON_EVNT_SEL0	Package	Uncore R-box 0 perfmon event select MSR.
E11H	3601	MSR_R0_PMON_CTR0	Package	Uncore R-box 0 perfmon counter MSR.
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.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
E2BH	3627	MSR_R1_PMON_IPERF1_P15	Package	Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 1perfmon 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.

Table 35-15 Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E54H	3668	MSR_M0_PMON_MM_CONFIG	Package	Uncore M-box 0 perfmon local box address match/mask config MSR.
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 18.7.2.2, "Uncore Performance Event Configuration Facility."

...

35.9 MSRS IN INTEL® PROCESSOR FAMILY BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Table 35-18 lists model-specific registers (MSRs) that are common to Intel® processor family based on Intel microarchitecture code name Sandy Bridge. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2AH, 06_2DH, see Table 35-1. Additional MSRs specific to 06_2AH are listed in Table 35-19.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 35.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 35.22, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 35-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.15, "Time-Stamp Counter," and see Table 35-2.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17H	23	IA32_PLATFORM_ID	Package	**Platform ID (R)** See Table 35-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
34H	52	MSR_SMI_COUNT	Thread	**SMI Counter (R/O)**
		31:0		**SMI Count (R/O)** Count SMIs.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64 Processor (R/W)** See Table 35-2.
		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 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Thread	**Performance Counter Register** See Table 35-2.
C2H	194	IA32_PMC1	Thread	**Performance Counter Register** See Table 35-2.
C3H	195	IA32_PMC2	Thread	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Thread	**Performance Counter Register** See Table 35-2.
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	See http://biosbits.org.
		7:0		Reserved.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The 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 Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, 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

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock 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.
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 (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
13CH	52	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 instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_ PERFEVTSEL0	Thread	See Table 35-2.
187H	391	IA32_ PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_ PERFEVTSEL2	Thread	See Table 35-2.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
189H	393	IA32_ PERFEVTSEL3	Thread	See Table 35-2.
18AH	394	IA32_ PERFEVTSEL4	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18BH	395	IA32_ PERFEVTSEL5	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18CH	396	IA32_ PERFEVTSEL6	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
18DH	397	IA32_ PERFEVTSEL7	Core	See Table 35-2; If CPUID.0AH:EAX[15:8] = 8
198H	408	IA32_PERF_STATUS	Package	See Table 35-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
198H	408	MSR_PERF_STATUS	Package	
		47:32		Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2^13).
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_ MODULATION	Thread	**Clock Modulation (R/W)** See Table 35-2 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 35-2.
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status (R/W)** See Table 35-2.
		0		**Thermal status (RO)** See Table 35-2.
		1		**Thermal status log (R/WC0)** See Table 35-2.
		2		**PROTCHOT # or FORCEPR# status (RO)** See Table 35-2.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		3		**PROTCHOT # or FORCEPR# log (R/WC0)** See Table 35-2.
		4		**Critical Temperature status (RO)** See Table 35-2.
		5		**Critical Temperature status log (R/WC0)** See Table 35-2.
		6		**Thermal threshold #1 status (RO)** See Table 35-2.
		7		**Thermal threshold #1 log (R/WC0)** See Table 35-2.
		8		**Thermal threshold #2 status (RO)** See Table 35-2.
		9		**Thermal threshold #2 log (R/WC0)** See Table 35-2.
		10		**Power Limitation status (RO)** See Table 35-2.
		11		**Power Limitation log (R/WC0)** See Table 35-2.
		15:12		Reserved.
		22:16		**Digital Readout (RO)** See Table 35-2.
		26:23		Reserved.
		30:27		**Resolution in degrees Celsius (RO)** See Table 35-2.
		31		**Reading Valid (RO)** See Table 35-2.
		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 35-2
		6:1		Reserved.
		7	Thread	**Performance Monitoring Available (R)** See Table 35-2.
		10:8		Reserved.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		11	Thread	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Thread	**Processor Event Based Sampling Unavailable (RO)** See Table 35-2.
		15:13		Reserved.
		16	Package	**Enhanced Intel SpeedStep Technology Enable (R/W)** See Table 35-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 35-2.
		21:19		Reserved.
		22	Thread	**Limit CPUID Maxval (R/W)** See Table 35-2.
		23	Thread	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Thread	**XD Bit Disable (R/W)** See Table 35-2.
		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 power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1A2H	418	MSR_TEMPERATURE_TARGET	Unique	
		15:0		Reserved.
		23:16		**Temperature Target (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A4H	420	MSR_MISC_FEATURE_CONTROL		**Miscellaneous Feature Control (R/W)**

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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		See http://biosbits.org.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 35-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 35-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 35-2.
1C8H	456	MSR_LBR_SELECT	Thread	**Last Branch Record Filtering Select Register (R/W)** See Section 17.7.2, "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.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 35-2.
		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)** 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 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 35-2.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 35-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_ 00000	Thread	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Thread	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Thread	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 35-2.
277H	631	IA32_PAT	Thread	See Table 35-2.
280H	640	IA32_MC0_CTL2	Core	See Table 35-2.
281H	641	IA32_MC1_CTL2	Core	See Table 35-2.
282H	642	IA32_MC2_CTL2	Core	See Table 35-2.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
283H	643	IA32_MC3_CTL2	Core	See Table 35-2.
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 35-2.
309H	777	IA32_FIXED_CTR0	Thread	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Thread	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Thread	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		11:8		PEBS_REC_FORMAT. See Table 35-2.
		12		SMM_FREEZE. See Table 35-2.
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 35-2. See Section 18.4.2, "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)**
		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**

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		60:35		Reserved.
		61	Thread	**Ovf_Uncore**
		62	Thread	**Ovf_BufDSSAVE**
		63	Thread	**CondChgd**
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "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 35-2. See Section 18.4.2, "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)**
		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.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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	MSR_PEBS_ENABLE	Thread	See Section 18.7.1.1, "Precise 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 See Section 18.7.1.2, "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.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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_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,” and Chapter 16.
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
403H	1027	IA32_MC0_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.”
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.”

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 35-2. See Appendix A.1, “Basic VMX Information.”
481H	1153	IA32_VMX_PINBASED_ CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, “VM-Execution Controls.”
482H	1154	IA32_VMX_PROCBASED_ CTLS	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_CTLS	Thread	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, “VM-Exit Controls.”
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, “VM-Entry Controls.”
485H	1157	IA32_VMX_MISC	Thread	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. 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 35-2. 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 35-2. See Appendix A.7, “VMX-Fixed Bits in CR0.”

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
488H	1160	IA32_VMX_CR4_FIXED0	Thread	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. 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 35-2. 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 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	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 35-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Table 35-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Table 35-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Table 35-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Table 35-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 35-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 35-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 35-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 35-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 35-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 35-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 35-2.
4C8H	1224	IA32_A_PMC7	Core	See Table 35-2.
600H	1536	IA32_DS_AREA	Thread	**DS Save Area (R/W)** See Table 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)** See Section 14.9.1, "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-sate management.
		63:16		Reserved.
60BH	1547	MSR_PKGC6_IRTL	Package	**Package C6 Interrupt Response Limit (R/W)** This MSR defines the budget allocated for the package to exit from C6 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy 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.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		12:10		**Time Unit (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved.
		15		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
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 14.9.3, “Package RAPL Domain.”
611H	1553	MSR_PKG_ENERGY_STATUS	Package	**PKG Energy Status (R/O)** See Section 14.9.3, “Package RAPL Domain.”
614H	1556	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameters (R/W)** See Section 14.9.3, “Package RAPL Domain.”
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)** See Section 14.9.4, “PP0/PP1 RAPL Domains.”
639H	1593	MSR_PP0_ENERGY_STATUS	Package	**PP0 Energy Status (R/O)** See Section 14.9.4, “PP0/PP1 RAPL Domains.”
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: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.7.1 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.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 35-2.
802H-83FH		X2APIC MSRs	Thread	See Table 35-2.
C000_ 0080H		IA32_EFER	Thread	**Extended Feature Enables** See Table 35-2.
C000_ 0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_ 0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.

Table 35-18 MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C000_ 0084H		IA32_FMASK	Thread	**System Call Flag Mask (R/W)** See Table 35-2.
C000_ 0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_ 0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0102H		IA32_KERNEL_GSBASE	Thread	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature (R/W)** See Table 35-2 and Section 17.15.2, “IA32_TSC_AUX Register and RDTSCP Support.”

35.9.1 MSRs In 2nd Generation Intel® Core™ Processor Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 35-19 and Table 35-20 list model-specific registers (MSRs) that are specific to the 2nd generation Intel® Core™ processor family (based on Intel microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2AH; see Table 35-1.

Table 35-19 MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® microarchitecture code name Sandy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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 35-19 MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® microarchitecture code name Sandy Bridge) (Contd.)

Register Address		Register Name	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 C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy 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-sate management.
		63:16		Reserved.
63AH	1594	MSR_PP0_POLICY	Package	**PP0 Balance Policy (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	**PP1 RAPL Power Limit Control (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	**PP1 Energy Status (R/O)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	**PP1 Balance Policy (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
See Table 35-18, Table 35-19, and Table 35-20 for MSR definitions applicable to processors with CPUID signature 06_2AH.				

Table 35-20 lists the MSRs of uncore PMU for Intel processors with CPUID signature 06_2AH.

Table 35-20 Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name	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.

Table 35-20 Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_PERFEVTSEL0	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_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

Table 35-20 Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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
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_PERFCTR0	Package	Uncore C-Box 4, performance counter 0.
747H	1863	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, performance counter 1.

Table 35-20 Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.

35.9.2 MSRs In Intel® Xeon® Processor E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 35-21 lists additional model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family (based on Intel® microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH, and also supports MSRs listed in Table 35-18 and Table 35-22.

Table 35-21 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture)

Register Address		Register Name	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** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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.

Table 35-21 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
39CH	924	MSR_PEBS_NUM_ALT	Package	
		0		**ENABLE_PEBS_NUM_ALT (RW)** Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see Table 19-9
		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.”
41DH	1053	IA32_MC7_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.

Table 35-21 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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."
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."

Table 35-21 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 14.9.5, “DRAM RAPL Domain.”
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, “DRAM RAPL Domain.”
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, “DRAM RAPL Domain.”
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, “DRAM RAPL Domain.”
See Table 35-18, Table 35-21, and Table 35-22 for MSR definitions applicable to processors with CPUID signature 06_2DH.				

...

35.10 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY (BASED ON INTEL® MICROARCHITECTURE CODE NAME IVY BRIDGE)

The 3rd generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v2 product family (based on Intel microarchitecture code name Ivy Bridge) support the MSR interfaces listed in Table 35-18, Table

35-19, Table 35-20, and Table 35-23. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3AH.

Table 35-23 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The 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 Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and 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, and 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. 11: Reserved
		39:35		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, 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.

Table 35-23 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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, lock 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.
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).

Table 35-23 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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)**

Table 35-23 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-18, Table 35-19 and Table 35-20 for other MSR definitions applicable to processors with CPUID signature 06_3AH				

35.10.1 MSRs In Intel® Xeon® Processor E5 v2 Product Family (Based on Ivy Bridge-E Microarchitecture)

Table 35-24 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 35-1. These processors supports the MSR interfaces listed in Table 35-18, and Table 35-24.

Table 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
4EH	78	MSR_PPIN_CTL	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) Set 1to prevent further writes to MSR_PPIN_CTL. Writing 1 to MSR_PPINCTL[bit 0] is permitted only if MSR_PPIN_CTL[bit 1] is clear, Default is 0. BIOS should provide an opt-in menu to enable the user to turn on MSR_PPIN_CTL[bit 1] for privileged inventory initialization agent to access MSR_PPIN. After reading MSR_PPIN, the privileged inventory initialization agent should write '01b' to MSR_PPIN_CTL to disable further access to MSR_PPIN and prevent unauthorized modification to MSR_PPIN_CTL.
		1		Enable_PPIN (R/W) If 1, enables MSR_PPIN to be accessible using RDMSR. Once set, attempt to write 1 to MSR_PPIN_CTL[bit 0] will cause #GP. If 0, an attempt to read MSR_PPIN will cause #GP. Default is 0.

Table 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:2		Reserved.
4FH	79	MSR_PPIN	Package	Protected Processor Inventory Number (R/O)
		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 MSR_PPIN is enabled. Access to MSR_PPIN is permitted only if MSR_PPIN_CTL[bits 1:0] = '10b'
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The 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 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 Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and 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 an temperature offset.
		39:31		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.

Table 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

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.
		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, lock bits 15:0 of this register until next reset.
		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.

Table 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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	
		15:0		Reserved.
		23:16		**Temperature Target (RO)** The minimum temperature at which PROCHOT# will be asserted. The value is degree 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 MSR_PLATFORM_INFO.[30] is set.
		63:28		Reserved.
1AEH	430	MSR_TURBO_RATIO_LIMIT 1	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 9C** Maximum turbo ratio limit of 9 core active.
		15:8	Package	**Maximum Ratio Limit for 10C** Maximum turbo ratio limit of 10core active.
		23:16	Package	**Maximum Ratio Limit for 11C** Maximum turbo ratio limit of 11 core active.
		31:24	Package	**Maximum Ratio Limit for 12C** Maximum turbo ratio limit of 12 core active.
		63:32		Reserved
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.

Table 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
294H	660	IA32_MC20_CTL2	Package	See Table 35-2.
295H	661	IA32_MC21_CTL2	Package	See Table 35-2.
296H	662	IA32_MC22_CTL2	Package	See Table 35-2.
297H	663	IA32_MC23_CTL2	Package	See Table 35-2.
298H	664	IA32_MC24_CTL2	Package	See Table 35-2.
299H	665	IA32_MC25_CTL2	Package	See Table 35-2.
29AH	666	IA32_MC26_CTL2	Package	See Table 35-2.
29BH	667	IA32_MC27_CTL2	Package	See Table 35-2.
29CH	668	IA32_MC28_CTL2	Package	See Table 35-2.
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 error 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 error 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.”. Banks MC7 and MC 8 report MC error from the two home agents.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	

Table 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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 error 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.”
42DH	1069	IA32_MC11_STATUS	Package	Bank MC11 reports MC error from a specific channel of the integrated memory controller.
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 error 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 error 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 error 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 error 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 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	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 error 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 error 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 error 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 error 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."
451H	1105	IA32_MC20_STATUS	Package	Bank MC20 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
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 error 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 error 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	
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 error 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	

Table 35-24 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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_MC2MISC	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 error 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 error 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 error 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 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
See Table 35-18, for other MSR definitions applicable to Intel Xeon processor E5 v2 with CPUID signature 06_3EH				

35.10.2 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 35-18, Table 35-24, and Table 35-25.

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64 Processor (R/W)** See Table 35-2.
		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	(R/W0)
		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** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 9C** Maximum turbo ratio limit of 9 core active.

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:8	Package	**Maximum Ratio Limit for 10C** Maximum turbo ratio limit of 10core active.
		23:16	Package	**Maximum Ratio Limit for 11C** Maximum turbo ratio limit of 11 core active.
		31:24	Package	**Maximum Ratio Limit for 12C** Maximum turbo ratio limit of 12 core active.
		39:32	Package	**Maximum Ratio Limit for 13C** Maximum turbo ratio limit of 13 core active.
		47:40	Package	**Maximum Ratio Limit for 14C** Maximum turbo ratio limit of 14 core active.
		55:48	Package	**Maximum Ratio Limit for 15C** Maximum turbo ratio limit of 15 core active.
		62:56		Reserved
		63	Package	**Semaphore for Turbo Ratio Limit Configuration** If 1, the processor uses override configuration[1] specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
29DH	669	IA32_MC29_CTL2	Package	See Table 35-2.
29EH	670	IA32_MC30_CTL2	Package	See Table 35-2.
29FH	671	IA32_MC31_CTL2	Package	See Table 35-2.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.7.1.1, "Precise 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)
		63:36		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

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 error from a specific CBo (core broadcast) and its corresponding slice of L3.
475H	1141	IA32_MC29_STATUS	Package	
476H	1142	IA32_MC29_ADDR	Package	
477H	1143	IA32_MC29_MISC	Package	
478H	1144	IA32_MC30_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC30 reports MC error 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 error 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 35-18, Table 35-24 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.

...

35.11 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 35-18, Table 35-19, Table 35-20, and Table 35-27. For an MSR listed in Table 35-18 that also appears in Table 35-27, Table 35-27 supercede Table 35-18.

The MSRs listed in Table 35-27 also apply to processors based on Haswell-E microarchitecture (see Section 35.12).

Table 35-27 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3BH	59	IA32_TSC_ADJUST	THREAD	**Per-Logical-Processor TSC ADJUST (R/W)** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The 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 Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and 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, and 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. 11: Reserved
		39:35		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, 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_PERFEVTSEL0	THREAD	**Performance Event Select for Counter 0 (R/W)** Supports all fields described inTable 35-2 and the fields below.

Table 35-27 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
187H	391	IA32_PERFEVTSEL1	THREAD	**Performance Event Select for Counter 1 (R/W)** Supports all fields described inTable 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 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 inTable 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
		33		IN_TXCP: see Section 18.11.5.1 When IN_TXCP=1 & IN_TX=1 and in sampling, 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 inTable 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 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.

Table 35-27 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 35-2.
		0		**LBR: Last Branch Record**
		1		**BTF**
		5:2		Reserved.
		6		**TR: Branch Trace**
		7		**BTS: Log Branch Trace Message to BTS buffer**
		8		**BTINT**
		9		**BTS_OFF_OS**
		10		**BTS_OFF_USER**
		11		**FREEZE_LBR_ON_PMI**
		12		**FREEZE_PERFMON_ON_PMI**
		13		**ENABLE_UNCORE_PMI**
		14		**FREEZE_WHILE_SMM**
		15		**RTM_DEBUG**
		63:15		Reserved.
491H	1169	IA32_VMX_VMFUNC	THREAD	**Capability Reporting Register of VM-function Controls (R/O)** See Table 35-2
60BH	1548	MSR_PKGC_IRTL1	Package	**Package C6/C7 Interrupt Response Limit 1 (R/W)** This MSR defines the interrupt response time limit used by the processor to manage transition to 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 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 35-18 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 35-27 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	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 transition to 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 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 35-18 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.
613H	1555	MSR_PKG_PERF_STATUS	Package	**PKG Perf Status (R/O)** See Section 14.9.3, "Package RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "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
		46:32		PKG_MAX_PWR_LVL1. Max Power setting allowed for ConfigTDP Level 1.

Table 35-27 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	**Dec**			
		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_FEATURE	Package	**Silicon Debug Feature Control (R/W)** See Table 35-2.

...

35.12 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 35-18, Table 35-27, and Table 35-30.

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

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.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		**I/O MWAIT Redirection Enable (R/W)**
		14:11		Reserved
		15		**CFG Lock (R/WO)**
		24:16		Reserved
		25		**C3 State Auto Demotion Enable (R/W)**
		26		**C1 State Auto Demotion Enable (R/W)**
		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**

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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	390	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
		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** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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.

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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.
		63:56	Package	**Maximum Ratio Limit for16C** Maximum turbo ratio limit of 16 core active.
1AFH	431	MSR_TURBO_RATIO_LIMIT2	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 error 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 error 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 error 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 error 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 error 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 error 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 error 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	

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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 error 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 error 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 error 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 error 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 error 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 error 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	

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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 14.9.1, "RAPL Interfaces."
		7:4	Package	**Reserved**
		12:8	Package	**Energy Status Units** Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	**Reserved**
		19:16	Package	**Time Units** See Section 14.9.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
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.

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit
		3		Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit
		4		Reserved.
		5		Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		Reserved.
		10		Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits
		12:11		Reserved.
		13		Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1
		14		Core Max n-core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency
		15		Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.
		16		**PROCHOT Log** When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		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.

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		18		**Power Budget Management Log** When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		**Platform Configuration Services Log** When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved.
		21		**Autonomous Utilization-Based Frequency Control Log** When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		**VR Therm Alert Log** When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		**Electrical Design Point Log** When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved.
		26		**Multi-Core Turbo Log** When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28:27		Reserved.
		29		**Core Frequency P1 Log** When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		**Core Max n-core Turbo Frequency Limiting Log** When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 35-30 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	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 (RW)** **Event encoding:** **0x0: no monitoring** **0x1: L3 occupancy monitoring** **all other encoding reserved.**
		31:8		Reserved.
		41:32		**RMID (RW)**
		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 35-18, Table 35-27 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.

...

35.13 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 35-18, Table 35-19, Table 35-20, Table 35-23, Table 35-27, Table 35-28, Table 35-32, and Table 35-33. For an MSR listed in Table 35-33 that also appears in the model-specific tables of prior generations, Table 35-33 supercede prior generation tables.

Table 35-32 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 35-32 Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 35-2. See Section 18.4.2, “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 36.2.4.2, “Table of Physical Addresses (ToPA).”
		60:56		Reserved.
		61		**Ovf_Uncore**
		62		**Ovf_BufDSSAVE**
		63		**CondChgd**
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2. See Section 18.4.2, “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.

Table 35-32 Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		55		**Set 1 to clear Trace_ToPA_PMI. See** Section 36.2.4.2, “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, MBZ.
		2		**OS**
		3		**User**
		6:4		Reserved, MBZ
		7		**CR3 filter**
		8		**ToPA; writing 0 will #GP if also setting TraceEn**
		9		Reserved, MBZ
		10		**TSCEn**
		11		**DisRETC**
		12		Reserved, MBZ
		13		**Reserved; writing 0 will #GP if also setting TraceEn**
		63:14		Reserved, MBZ.
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)**

Table 35-32 Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		5		**Stopped**
		63:6		Reserved, MBZ.
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

NOTES:

1. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

...

35.14 MSRS IN INTEL® XEON® PROCESSORS E5 V4 FAMILY

The MSRs listed in Table 35-34 are available and common to Intel® Xeon® Processor D product Family (CPUID DisplayFamily_DisplayModel = 06_56H) and to Intel Xeon processors E5 v4 family (CPUID DisplayFamily_DisplayModel = 06_4FH). They are based on the Broadwell microarchitecture.

See Section 35.14.1 for lists of tables of MSRs that are supported by Intel® Xeon® Processor D Family.

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
4EH	78	MSR_PPIN_CTL	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 35-24.
		1		Enable_PPIN (R/W) See Table 35-24.
		63:2		Reserved.
4FH	79	MSR_PPIN	Package	Protected Processor Inventory Number (R/O)
		63:0		**Protected Processor Inventory Number (R/O)** See Table 35-24.
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** See Table 35-24.
		22:16		Reserved.

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23	Package	**PPIN_CAP (R/O)** See Table 35-24.
		27:24		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** See Table 35-24.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** See Table 35-24.
		30	Package	**Programmable TJ OFFSET (R/O)** See Table 35-24.
		39:31		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** See Table 35-24.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.
		9:3		Reserved
		10		**I/O MWAIT Redirection Enable (R/W)**
		14:11		Reserved
		15		**CFG Lock (R/WO)**
		16		**Automatic C-State Conversion Enable (R/W)** If 1, the processor will convert HALT or MWAT(C1) to MWAIT(C6)
		24:17		Reserved
		25		**C3 State Auto Demotion Enable (R/W)**

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		26		**C1 State Auto Demotion Enable (R/W)**
		27		**Enable C3 Undemotion (R/W)**
		28		**Enable C1 Undemotion (R/W)**
		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	390	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 35-2.
		0		**Thermal status (RO)** See Table 35-2.
		1		**Thermal status log (R/WC0)** See Table 35-2.

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		**PROTCHOT # or FORCEPR# status (RO)** See Table 35-2.
		3		**PROTCHOT # or FORCEPR# log (R/WC0)** See Table 35-2.
		4		**Critical Temperature status (RO)** See Table 35-2.
		5		**Critical Temperature status log (R/WC0)** See Table 35-2.
		6		**Thermal threshold #1 status (RO)** See Table 35-2.
		7		**Thermal threshold #1 log (R/WC0)** See Table 35-2.
		8		**Thermal threshold #2 status (RO)** See Table 35-2.
		9		**Thermal threshold #2 log (R/WC0)** See Table 35-2.
		10		**Power Limitation status (RO)** See Table 35-2.
		11		**Power Limitation log (R/WC0)** See Table 35-2.
		12		**Current Limit status (RO)** See Table 35-2.
		13		**Current Limit log (R/WC0)** See Table 35-2.
		14		**Cross Domain Limit status (RO)** See Table 35-2.
		15		**Cross Domain Limit log (R/WC0)** See Table 35-2.
		22:16		**Digital Readout (RO)** See Table 35-2.
		26:23		Reserved.
		30:27		**Resolution in degrees Celsius (RO)** See Table 35-2.
		31		**Reading Valid (RO)** See Table 35-2.

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:32		Reserved.
1A2H	418	MSR_ TEMPERATURE_TARGET	Package	
		15:0		Reserved.
		23:16		**Temperature Target (RO)** See Table 35-24.
		27:24		**TCC Activation Offset (R/W)** See Table 35-24.
		63:28		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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 14.9.1, "RAPL Interfaces."

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		7:4	Package	**Reserved**
		12:8	Package	**Energy Status Units** Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	**Reserved**
		19:16	Package	**Time Units** See Section 14.9.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
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.
		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

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		Reserved.
		10		Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits
		12:11		Reserved.
		13		Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1
		14		Core Max n-core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency
		15		Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.
		16		**PROCHOT Log** When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		**Thermal Log** When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		18		**Power Budget Management Log** When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19		**Platform Configuration Services Log** When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		20		Reserved.

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		21		**Autonomous Utilization-Based Frequency Control Log** When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		**VR Therm Alert Log** When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		**Electrical Design Point Log** When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		Reserved.
		26		**Multi-Core Turbo Log** When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28:27		Reserved.
		29		Core Frequency P1 **Log** When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		30		Core Max n-core Turbo Frequency Limiting **Log** When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		31		Core Frequency Limiting **Log** When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:32		Reserved.
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)**

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 (RO)**
		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 (RW)** **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 (RW)**
		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

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0:19		**CBM: Bit vector of available L3 ways for COS 2 enforcement**
		63:20		**Reserved**
C93H	3219	IA32_L3_QOS_MASK_3	Package	**L3 Class Of Service Mask - COS 3 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3
		0:19		**CBM: Bit vector of available L3 ways for COS 3 enforcement**
		63:20		**Reserved**
C94H	3220	IA32_L3_QOS_MASK_4	Package	**L3 Class Of Service Mask - COS 4 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4
		0:19		**CBM: Bit vector of available L3 ways for COS 4 enforcement**
		63:20		**Reserved**
C95H	3221	IA32_L3_QOS_MASK_5	Package	**L3 Class Of Service Mask - COS 5 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5
		0:19		**CBM: Bit vector of available L3 ways for COS 5 enforcement**
		63:20		**Reserved**
C96H	3222	IA32_L3_QOS_MASK_6	Package	**L3 Class Of Service Mask - COS 6 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6
		0:19		**CBM: Bit vector of available L3 ways for COS 6 enforcement**
		63:20		**Reserved**
C97H	3223	IA32_L3_QOS_MASK_7	Package	**L3 Class Of Service Mask - COS 7 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7
		0:19		**CBM: Bit vector of available L3 ways for COS 7 enforcement**
		63:20		**Reserved**
C98H	3224	IA32_L3_QOS_MASK_8	Package	**L3 Class Of Service Mask - COS 8 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8
		0:19		**CBM: Bit vector of available L3 ways for COS 8 enforcement**
		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**

Table 35-34 Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:20		**Reserved**
C9BH	3227	IA32_L3_QOS_MASK_11	Package	**L3 Class Of Service Mask - COS 11 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11
		0:19		**CBM: Bit vector of available L3 ways for COS 11 enforcement**
		63:20		**Reserved**
C9CH	3228	IA32_L3_QOS_MASK_12	Package	**L3 Class Of Service Mask - COS 12 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12
		0:19		**CBM: Bit vector of available L3 ways for COS 12 enforcement**
		63:20		**Reserved**
C9DH	3229	IA32_L3_QOS_MASK_13	Package	**L3 Class Of Service Mask - COS 13 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13
		0:19		**CBM: Bit vector of available L3 ways for COS 13 enforcement**
		63:20		**Reserved**
C9EH	3230	IA32_L3_QOS_MASK_14	Package	**L3 Class Of Service Mask - COS 14 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14
		0:19		**CBM: Bit vector of available L3 ways for COS 14 enforcement**
		63:20		**Reserved**
C9FH	3231	IA32_L3_QOS_MASK_15	Package	**L3 Class Of Service Mask - COS 15 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15
		0:19		**CBM: Bit vector of available L3 ways for COS 15 enforcement**
		63:20		**Reserved**

35.14.1 Additional MSRs Supported in the Intel® Xeon® Processor D Product Family

The MSRs listed in Table 35-35 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 35-18, Table 35-27, Table 35-32, Table 35-34, and Table 35-35.

Table 35-35 Additional MSRs Supported by Intel® Xeon® Processor D with DisplayFamily_DisplayModel 06_56H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	**Config Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
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 error 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 error 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 error 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 error 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	

Table 35-35 Additional MSRs Supported by Intel® Xeon® Processor D with DisplayFamily_DisplayModel 06_56H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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 error 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 35-18, Table 35-27, Table 35-32, and Table 35-34 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.

35.14.2 Additional MSRs Supported in Intel® Xeon® Processors E5 v4 Family

The MSRs listed in Table 35-35 are available to Intel® Xeon® Processor E5 v4 Family (CPUID DisplayFamily_DisplayModel = 06_4FH). The Intel® Xeon® processor E5 v4 family is based on the Broadwell microarchitecture and supports the MSR interfaces listed in Table 35-18, Table 35-19, Table 35-27, Table 35-32, Table 35-34, and Table 35-36.

Table 35-36 Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	**Config Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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).

Table 35-36 Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
294H	660	IA32_MC20_CTL2	Package	See Table 35-2.
295H	661	IA32_MC21_CTL2	Package	See Table 35-2.
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 error 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 error 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 error 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 error from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	

Table 35-36 Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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 error 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 error 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 error 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 error 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 error 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 error 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	

Table 35-36 Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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 error 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 error 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 error from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
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.

Table 35-36 Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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
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 35-18, Table 35-19, Table 35-27, and Table 35-28 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.

35.15 MSRS IN THE 6TH GENERATION INTEL® CORE™ PROCESSORS

The 6th generation Intel® Core™ processor family is based on the Skylake microarchitecture. They have CPUID DisplayFamily_DisplayModel signatures of 06_4EH and 06_5EH, supports the MSR interfaces listed in Table 35-18, Table 35-19, Table 35-23, Table 35-27, Table 35-33, Table 35-37, and Table 35-38. For an MSR listed in Table 35-37 that also appears in the model-specific tables of prior generations, Table 35-37 supercede prior generation tables.

The notation of “Platform” in the Scope column (with respect to MSR_PLATFORM_ENERGY_COUNTER and MSR_PLATFORM_POWER_LIMIT) is limited to the power-delivery domain and the specifics of the power delivery integration may vary by platform vendor’s implementation.

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64 Processor (R/W)** See Table 35-2.
		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_ON (R/WL)**

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:21		Reserved.
FEH	254	IA32_MTRRCAP	Thread	**MTRR Capality (RO, Architectural).** See Table 35-2
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status (R/W)** See Table 35-2.
		0		**Thermal status (RO)** See Table 35-2.
		1		**Thermal status log (R/WC0)** See Table 35-2.
		2		**PROTCHOT # or FORCEPR# status (RO)** See Table 35-2.
		3		**PROTCHOT # or FORCEPR# log (R/WC0)** See Table 35-2.
		4		**Critical Temperature status (RO)** See Table 35-2.
		5		**Critical Temperature status log (R/WC0)** See Table 35-2.
		6		**Thermal threshold #1 status (RO)** See Table 35-2.
		7		**Thermal threshold #1 log (R/WC0)** See Table 35-2.
		8		**Thermal threshold #2 status (RO)** See Table 35-2.
		9		**Thermal threshold #2 log (R/WC0)** See Table 35-2.
		10		**Power Limitation status (RO)** See Table 35-2.
		11		**Power Limitation log (R/WC0)** See Table 35-2.
		12		**Current Limit status (RO)** See Table 35-2.
		13		**Current Limit log (R/WC0)** See Table 35-2.
		14		**Cross Domain Limit status (RO)** See Table 35-2.

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15		**Cross Domain Limit log (R/WC0)** See Table 35-2.
		22:16		**Digital Readout (RO)** See Table 35-2.
		26:23		Reserved.
		30:27		**Resolution in degrees Celsius (RO)** See Table 35-2.
		31		**Reading Valid (RO)** See Table 35-2.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW 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.
300H	768	MSR_SGXOWNER0	Package	**Lower 64 Bit OwnerEpoch Component of SGX Key (RO).**
		63:0		**Low 64 bits of an 128-bit external entropy value for key derivation of an enclave.**
301H	768	MSR_SGXOWNER1	Package	**Upper 64 Bit OwnerEpoch Component of SGX Key (RO).**
		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 35-2. See Section 18.2.2.3, "Full-Width Writes to Performance Counter Registers."
		0	Thread	**Ovf_PMC0**
		1	Thread	**Ovf_PMC1**
		2	Thread	**Ovf_PMC2**

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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**
		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 35-2. See Section 18.2.2.3, "Full-Width Writes to Performance Counter Registers."
		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.

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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**
391H	913	IA32_PERF_GLOBAL_STATUS_SET		See Table 35-2. See Section 18.2.2.3, "Full-Width Writes to Performance Counter Registers."
		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	Thread	**Set 1 to cause CondChgd = 1**
392H	913	IA32_PERF_GLOBAL_INUSE		See Table 35-2.
3F7H	1015	MSR_PEBS_FRONTEND	Thread	**FrontEnd Precise Event Condition Select (R/W)**

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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**
500H	1280	IA32_SGX_SVN_STATUS	Thread	**Status and SVN Threshold of SGX Support for ACM (RO).**
		0		**Lock.** See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)"
		15:1		Reserved.
		23:16		**SGX_SVN_SINIT.** See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)"
		63:24		Reserved.
560H	1376	IA32_RTIT_OUTPUT_BASE	Thread	**Trace Output Base Register (R/W).** See Table 35-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Thread	**Trace Output Mask Pointers Register (R/W).** See Table 35-2.
570H	1392	IA32_RTIT_CTL	Thread	**Trace Control Register (R/W)**
		0		**TraceEn**
		1		**CYCEn**
		2		**OS**
		3		**User**
		6:4		Reserved, MBZ
		7		**CR3 filter**
		8		**ToPA; writing 0 will #GP if also setting TraceEn**
		9		**MTCEn**
		10		**TSCEn**
		11		**DisRETC**
		12		Reserved, MBZ
		13		**BranchEn**
		17:14		**MTCFreq**
		18		Reserved, MBZ
		22:19		**CYCThresh**
		23		Reserved, MBZ

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		27:24		**PSBFreq**
		31:28		Reserved, MBZ
		35:32		**ADDR0_CFG**
		39:36		**ADDR1_CFG**
		63:40		Reserved, MBZ.
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. MBZ
		48:32		**PacketByteCnt**
		63:49		Reserved, MBZ.
572H	1394	IA32_RTIT_CR3_MATCH	Thread	**Trace Filter CR3 Match Register (R/W)**
		4:0		Reserved
		63:5		CR3[63:5] value to match
580H	1408	IA32_RTIT_ADDR0_A	Thread	**Region 0 Start Address (R/W)**
		63:0		See Table 35-2.
581H	1409	IA32_RTIT_ADDR0_B	Thread	**Region 0 End Address (R/W)**
		63:0		See Table 35-2.
582H	1410	IA32_RTIT_ADDR1_A	Thread	**Region 1 Start Address (R/W)**
		63:0		See Table 35-2.
583H	1411	IA32_RTIT_ADDR1_B	Thread	**Region 1 End Address (R/W)**
		63:0		See Table 35-2.
64DH	1613	MSR_PLATFORM_ENERGY_COUNTER	Platform*	Platform Energy Counter. (R/O). This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it. This MSR will read 0 if not valid.

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:0		Total energy consumed by all devices in the platform that receive power from integrated power delivery mechanism, Included platform devices are processor cores, SOC, memory, add-on or peripheral devices that get powered directly from the platform power delivery means. The energy units are specified in the MSR_RAPL_POWER_UNIT.Enery_Status_Unit.
		63:32		Reserved.
64EH	1614	MSR_PPERF	Thread	Productive Performance Count. (R/O).
		63:0		Hardware's view of workload scalability. See Section 14.4.5.1
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.
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_C0	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_C0	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_C0	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_C0	Package	Core and Graphics Engine Overlapped C0 Residency. (R/O)

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
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: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.10
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 19From 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.

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.10
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 19To 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.

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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 35-2.
DA0H	3488	IA32_XSS	Thread	See Table 35-2.
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"

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.7.1, "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.
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.

Table 35-37 Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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 35-38 lists the MSRs of uncore PMU for Intel processors with CPUID DisplayFamily_DisplayModel signatures of 06_4EH and 06_5EH.

Table 35-38 Uncore PMU MSRs Supported by 6th Generation Intel® Core™ Processors

Register Address		Register Name	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
		63:44		Reserved.
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box configuration information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics),
		63:4		Reserved.
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb unit, performance counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSEL0	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_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
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_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
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

Table 35-38 Uncore PMU MSRs Supported by 6th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, performance counter 1
730H	1840	MSR_UNC_CBO_3_ 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.
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, performance counter 0.
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, performance counter 1.
E01H	3585	MSR_UNC_PERF_GLOBAL_ CTRL	Package	Uncore PMU global control
		0		Slice 0 select
		1		Slice 1 select
		2		Slice 2 select
		3		Slice 3 select
		4		Slice 4select
		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.

35.16 MSRS IN FUTURE INTEL® XEON® PROCESSORS

Future Intel® Xeon® Processors (CPUID DisplayFamily_DisplayModel = 06_55H) support the machine check bank registers listed in Table 35-40.

Table 35-39 Machine Check MSRs Supported by Future Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
280H	640	IA32_MC0_CTL2	Package	See Table 35-2.
281H	641	IA32_MC1_CTL2	Package	See Table 35-2.
282H	642	IA32_MC2_CTL2	Package	See Table 35-2.
283H	643	IA32_MC3_CTL2	Package	See Table 35-2.
284H	644	IA32_MC4_CTL2	Package	See Table 35-2.
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
400H	1024	IA32_MC0_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.” through Section 15.3.2.4, “IA32_MCi_MISC MSRs.”. Bank MC0 reports MC error from the IFU module.
401H	1025	IA32_MC0_STATUS	Package	
402H	1026	IA32_MC0_ADDR	Package	
403H	1027	IA32_MC0_MISC	Package	
404H	1028	IA32_MC1_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.” through Section 15.3.2.4, “IA32_MCi_MISC MSRs.”. Bank MC1 reports MC error from the DCU module.
405H	1029	IA32_MC1_STATUS	Package	
406H	1030	IA32_MC1_ADDR	Package	
407H	1031	IA32_MC1_MISC	Package	
408H	1032	IA32_MC2_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.” through Section 15.3.2.4, “IA32_MCi_MISC MSRs.”. Bank MC2 reports MC error from the DTLB module.
409H	1033	IA32_MC2_STATUS	Package	
40AH	1034	IA32_MC2_ADDR	Package	
40BH	1035	IA32_MC2_MISC	Package	

Table 35-39 Machine Check MSRs Supported by Future Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
40CH	1036	IA32_MC3_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC3 reports MC error from the MLC module.
40DH	1037	IA32_MC3_STATUS	Package	
40EH	1038	IA32_MC3_ADDR	Package	
40FH	1039	IA32_MC3_MISC	Package	
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 error 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 error 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 error 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 error 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 error 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 error from the CHA
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 - MC11 report MC error from the CHA.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	

Table 35-39 Machine Check MSRs Supported by Future Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 error 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 error 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 error 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 error 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 error 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 error 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 error 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 error from the integrated memory controllers.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	

Table 35-39 Machine Check MSRs Supported by Future Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_55H

Register Address		Register 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 error from a link interconnect module.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	

35.17 MSRS IN THE NEXT GENERATION INTEL® XEON PHI™ PROCESSORS

The next generation Intel® Xeon Phi™ processor family, with CPUID DisplayFamily_DisplayModel signature 06_57H, supports the MSR interfaces listed in Table 35-40. These processors are based on the Knights Landing microarchitecture. Some MSRs are shared between a pair of processor cores, the scope is marked as module.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 35.22, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 35.22, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_ SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination." and Table 35-2
10H	16	IA32_TIME_STAMP_ COUNTER	Thread	See Section 17.15, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Package	**Platform ID (R)** See Table 35-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
34H	52	MSR_SMI_COUNT	Thread	**SMI Counter (R/O)**
		31:0		**SMI Count (R/O)**
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
		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 35-2.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
79H	121	IA32_BIOS_UPDT_TRIG	Core	**BIOS Update Trigger Register (W)** See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	THREAD	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	THREAD	**Performance counter register** See Table 35-2.
C2H	194	IA32_PMC1	THREAD	**Performance Counter Register** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The 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 Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Module	**C-State Configuration Control (R/W)**
		2:0		**Package C-State Limit (R/W)** The following C-state code name encodings are supported: 000b: C0/C1 001b: C2 010b: C6 No Retention 011b: C6 Retention 111b: No limit
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)**

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		14:11		Reserved.
		15		**CFG Lock (R/WO)**
		63:16		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_ BASE	Module	**Power Management IO Redirection in C-state (R/W)**
		15:0		**LVL_2 Base Address (R/W)**
		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
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Core	**Memory Type Range Register (R)** See Table 35-2.
13CH	52	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 instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	See Table 35-2.
186H	390	IA32_PERFEVTSEL0	Thread	Performance Monitoring Event Select Register (R/W) See Table 35-2.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		7:0		**Event Select**
		15:8		**UMask**
		16		**USR**
		17		**OS**
		18		**Edge**
		19		**PC**
		20		**INT**
		21		**AnyThread**
		22		**EN**
		23		**INV**
		31:24		**CMASK**
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 35-2.
198H	408	IA32_PERF_STATUS	Package	See Table 35-2.
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	**Clock Modulation (R/W)** See Table 35-2.
19BH	411	IA32_THERM_INTERRUPT	Module	**Thermal Interrupt Control (R/W)** See Table 35-2.
19CH	412	IA32_THERM_STATUS	Module	**Thermal Monitor Status (R/W)** See Table 35-2.
		0		**Thermal status (RO)**
		1		**Thermal status log (R/WC0)**
		2		**PROTCHOT # or FORCEPR# status (RO)**
		3		**PROTCHOT # or FORCEPR# log (R/WC0)**
		4		**Critical Temperature status (RO)**
		5		**Critical Temperature status log (R/WC0)**
		6		**Thermal threshold #1 status (RO)**
		7		**Thermal threshold #1 log (R/WC0)**
		8		**Thermal threshold #2 status (RO)**
		9		**Thermal threshold #2 log (R/WC0)**
		10		**Power Limitation status (RO)**
		11		**Power Limitation log (R/WC0)**
		15:12		Reserved.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		22:16		**Digital Readout (RO)**
		26:23		Reserved.
		30:27		**Resolution in degrees Celsius (RO)**
		31		**Reading Valid (RO)**
		63:32		Reserved.
1A0H	416	IA32_MISC_ENABLE	Thread	**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0		**Fast-Strings Enable**
		2:1		Reserved.
		3		**Automatic Thermal Control Circuit Enable (R/W)** Default value is 1.
		6:4		Reserved.
		7		**Performance Monitoring Available (R)**
		10:8		Reserved.
		11		**Branch Trace Storage Unavailable (RO)**
		12		**Processor Event Based Sampling Unavailable (RO)**
		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	
		15:0		Reserved.
		23:16		**Temperature Target (R)**
		29:24		**Target Offset (R/W)**
		63:30		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)**

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode for Groups of Cores (RW)**
		0		**Reserved**
		7:1	Package	**Maximum Number of Cores in Group 0** Number active processor cores which operates under the maximum ratio limit for group 0.
		15:8	Package	**Maximum Ratio Limit for Group 0** Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.
		20:16	Package	**Number of Incremental Cores Added to Group 1** Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".
		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.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		52:48	Package	**Number of Incremental Cores Added to Group 5** Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".
		55:53	Package	**Group Ratio Delta for Group 5** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.
		60:56	Package	**Number of Incremental Cores Added to Group 6** Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".
		63:61	Package	**Group Ratio Delta for Group 6** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.
1B0H	432	IA32_ENERGY_PERF_BIAS	Thread	See Table 35-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 35-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 35-2.
1C8H	456	MSR_LBR_SELECT	Thread	**Last Branch Record Filtering Select Register (R/W)**
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)**
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 35-2.
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 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 35-2.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 35-2.
277H	631	IA32_PAT	Core	See Table 35-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	**Default Memory Types (R/W)** See Table 35-2.
309H	777	IA32_FIXED_CTR0	Thread	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Thread	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Thread	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Core	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 35-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Table 35-2.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O)
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	
		63:0		Package C6 Residency Counter. (R/O)
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	
		63:0		Package C7 Residency Counter. (R/O)
3FCH	1020	MSR_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.
		63:0		Module C0 Residency Counter. (R/O)
3FDH	1021	MSR_MC6_RESIDENCY	Module	
		63:0		Module C6 Residency Counter. (R/O)
3FFH	1023	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O)
400H	1024	IA32_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."

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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."
480H	1152	IA32_VMX_BASIC	Core	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2.
481H	1153	IA32_VMX_PINBASED_CTLS	Core	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2.
482H	1154	IA32_VMX_PROCBASED_CTLS	Core	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)**
483H	1155	IA32_VMX_EXIT_CTLS	Core	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2.
484H	1156	IA32_VMX_ENTRY_CTLS	Core	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2.
485H	1157	IA32_VMX_MISC	Core	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2.
486H	1158	IA32_VMX_CR0_FIXED0	Core	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2.
487H	1159	IA32_VMX_CR0_FIXED1	Core	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2.
488H	1160	IA32_VMX_CR4_FIXED0	Core	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2.
489H	1161	IA32_VMX_CR4_FIXED1	Core	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Table 35-2.
48AH	1162	IA32_VMX_VMCS_ENUM	Core	**Capability Reporting Register of VMCS Field Enumeration (R/O)** See Table 35-2.
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Core	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Table 35-2
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	**Capability Reporting Register of EPT and VPID (R/O)** See Table 35-2

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Core	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Table 35-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Core	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Table 35-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Core	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Table 35-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Core	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Table 35-2
491H	1169	IA32_VMX_FMFUNC	Core	**Capability Reporting Register of VM-function Controls (R/O)** See Table 35-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 35-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 35-2.
600H	1536	IA32_DS_AREA	Thread	**DS Save Area (R/W)** See Table 35-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)**
		3:0	Package	**Power Units** See Section 14.9.1, “RAPL Interfaces.”
		7:4	Package	**Reserved**
		12:8	Package	**Energy Status Units** Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	**Reserved**
		19:16	Package	**Time Units** See Section 14.9.1, “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 14.9.3, “Package RAPL Domain.”
611H	1553	MSR_PKG_ENERGY_STATUS	Package	**PKG Energy Status (R/O)** See Section 14.9.3, “Package RAPL Domain.”
613H	1555	MSR_PKG_PERF_STATUS	Package	**PKG Perf Status (R/O)** See Section 14.9.3, “Package RAPL Domain.”

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
614H	1556	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameters (R/W)** See Section 14.9.3, "Package RAPL Domain."
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	**PP0 Energy Status (R/O)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
648H	1608	MSR_CONFIG_TDP_ NOMINAL	Package	**Base TDP Ratio (R/O)** See Table 35-23
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O). See Table 35-23
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O). See Table 35-23
64BH	1611	MSR_CONFIG_TDP_ CONTROL	Package	**ConfigTDP Control (R/W)** See Table 35-23
64CH	1612	MSR_TURBO_ACTIVATION_ RATIO	Package	**ConfigTDP Control (R/W)** See Table 35-23
690H	1680	MSR_CORE_PERF_LIMIT_RE ASONS	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 35-2
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID register (R/O) See x2APIC Specification.

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode register bits [223:192] (R/O)
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)

Table 35-40 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 35-2.
C000_0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_0084H		IA32_FMASK	Thread	**System Call Flag Mask (R/W)** See Table 35-2.
C000_0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Thread	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.
C000_0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature. (R/W)** See Table 35-2

35.18 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 35-41 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 35-1.

- 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 35-41 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 35.22, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Section 35.22, “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 35-2.
					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 35-2. 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 35-2. 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.

Table 35-41 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				
		2			**In Order Queue Depth (R)** Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		3			**MCERR# Observation Disabled (R)** Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		4			**BINIT# Observation Enabled (R)** Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		6:5			**APIC Cluster ID (R)** Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		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.

Table 35-41 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			**RCNT/SCNT On Request Encoding Enable (R/W)** Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).
		1			**Data Error Checking Disable (R/W)** Set to disable system data bus parity checking; clear to enable parity checking.
		2			**Response Error Checking Disable (R/W)** Set to disable (default); clear to enable.
		3			**Address/Request Error Checking Disable (R/W)** Set to disable (default); clear to enable.
		4			**Initiator MCERR# Disable (R/W)** Set to disable MCERR# driving for initiator bus requests (default); clear to enable.
		5			**Internal MCERR# Disable (R/W)** Set to disable MCERR# driving for initiator internal errors (default); clear to enable.
		6			**BINIT# Driver Disable (R/W)** Set to disable BINIT# driver (default); clear to enable driver.
		63:7			Reserved.
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: EncodingScalable Bus Speed 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6)

Table 35-41 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				
					133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
					266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.
		23:19			Reserved.
		31:24			**Core Clock Frequency to System Bus Frequency Ratio (R)** The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.
		63:25			Reserved.
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: Encoding Scalable Bus Speed 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 35-2 (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 35-2.

Table 35-41 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				
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	**BIOS Update Signature ID (R/W)** See Table 35-2.
9BH	155	IA32_SMM_MONITOR_CTL	3, 4, 6	Unique	**SMM Monitor Configuration (R/W)** See Table 35-2.
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 35-2. 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 35-2. See Section 5.8.7, “Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions.”
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	**CPL 0 code entry point (R/W)** See Table 35-2. 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 35-2. 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 35-2. See Section 15.3.1.2, “IA32_MCG_STATUS MSR.”
17BH	379	IA32_MCG_CTL			**Machine Check Feature Enable (R/W)** See Table 35-2. 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.”

Table 35-41 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			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.
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.

Table 35-41 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				
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	MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5			Reserved.
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.

Table 35-41 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				
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."
		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."

Table 35-41 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.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	See Table 35-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	See Table 35-2. 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 35-2. See Section 14.7.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3, 4, 6	Unique	**Thermal Interrupt Control (R/W)** See Section 14.7.2, "Thermal Monitor," and see Table 35-2.
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	**Thermal Monitor Status (R/W)** See Section 14.7.2, "Thermal Monitor," and see Table 35-2.
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)**
		0			Fast-Strings Enable. See Table 35-2.
		1			Reserved.
		2			**x87 FPU Fopcode Compatibility Mode Enable**
		3			**Thermal Monitor 1 Enable** See Section 14.7.2, "Thermal Monitor," and see Table 35-2.

Table 35-41 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			**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 35-2.
		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.
					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.

Table 35-41 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				
					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 35-2. 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 35-2. 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 35-2.
		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.

Table 35-41 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				
					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 35-2. 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 35-2.
		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 35-2.
		63:35			Reserved.
1A1H	417	MSR_PLATFORM_BRV	3, 4, 6	Shared	**Platform Feature Requirements (R)**
		17:0			Reserved.
		18			**PLATFORM Requirements** When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.
		63:19			Reserved.

Table 35-41 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				
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 17.11.3, "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 17.11.3, "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 17.11.1, "MSR_DEBUGCTLA MSR."
1DAH	474	MSR_LASTBRANCH _TOS	0, 1, 2, 3, 4, 6	Unique	**Last Branch Record Stack TOS (R/W)** 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 17.11.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture"; and addresses 1DBH-1DEH and 680H-68FH.

Table 35-41 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/W)** 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 17.10, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
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."

Table 35-41 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				
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."
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."

Table 35-41 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				
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 35-2. See Section 11.11.2.1, “IA32_MTRR_DEF_TYPE MSR.”
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
308H	776	MSR_FLAME_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
309H	777	MSR_FLAME_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
30AH	778	MSR_FLAME_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
30BH	779	MSR_FLAME_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, “Performance Counters.”

Table 35-41 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				
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.2, "Performance Counters."
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
368H	872	MSR_FLAME_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.3, "CCCR MSRs."
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."

Table 35-41 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				
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3AEH	942	MSR_SAAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3AFH	943	MSR_SAAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."

Table 35-41 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				
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 18.12.1, "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 18.12.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, "ESCR MSRs."

Table 35-41 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				
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, “ESCR MSRs.”
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, “ESCR MSRs.”
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, “ESCR MSRs.”
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, “ESCR MSRs.”
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, “ESCR MSRs.”
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, “ESCR MSRs.”
3F0H	1008	MSR_TC_PRECISE_EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 18.12.1, “ESCR MSRs.”
3F1H	1009	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 Table 19-26.
		23:13			Reserved.
		24			**UOP Tag** Enables replay tagging when set.
		25			**ENABLE_PEBS_MY_THR (R/W)** Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 18.13.3, “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 18.13.3, “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 Table 19-26.

Table 35-41 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				
400H	1024	IA32_MC0_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
401H	1025	IA32_MC0_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
402H	1026	IA32_MC0_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MC0_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.” The IA32_MC0_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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.”
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 ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 MISCV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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.”

Table 35-41 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				
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 MISCV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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.”

Table 35-41 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				
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 ADDRV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	3, 4, 6	Unique	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	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 35-2.
483H	1155	IA32_VMX_EXIT_CTLS	3, 4, 6	Unique	**Capability Reporting Register of VM-exit Controls (R/O)** See Appendix A.4, "VM-Exit Controls," and see Table 35-2.
484H	1156	IA32_VMX_ENTRY_CTLS	3, 4, 6	Unique	**Capability Reporting Register of VM-entry Controls (R/O)** See Appendix A.5, "VM-Entry Controls," and see Table 35-2.
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 35-2.

Table 35-41 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				
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 35-2.
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 35-2.
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 35-2.
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 35-2.
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 35-2.
48BH	1163	IA32_VMX_PROCBASED_CTLS2	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 35-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	**DS Save Area (R/W)** See Table 35-2. See Section 18.12.4, “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.
					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 17.10, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture.”

Table 35-41 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				
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.
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.

Table 35-41 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				
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 17.10, "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.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	3, 4, 6	Unique	**Last Branch Record 9** See description of MSR_LASTBRANCH_0 at 6C0H.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	3, 4, 6	Unique	**Last Branch Record 10** See description of MSR_LASTBRANCH_0 at 6C0H.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	3, 4, 6	Unique	**Last Branch Record 11** See description of MSR_LASTBRANCH_0 at 6C0H.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	3, 4, 6	Unique	**Last Branch Record 12** See description of MSR_LASTBRANCH_0 at 6C0H.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	3, 4, 6	Unique	**Last Branch Record 13** See description of MSR_LASTBRANCH_0 at 6C0H.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	3, 4, 6	Unique	**Last Branch Record 14** See description of MSR_LASTBRANCH_0 at 6C0H.

Table 35-41 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				
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	3, 4, 6	Unique	**Last Branch Record 15** See description of MSR_LASTBRANCH_0 at 6C0H.
C000_ 0080H		IA32_EFER	3, 4, 6	Unique	**Extended Feature Enables** See Table 35-2.
C000_ 0081H		IA32_STAR	3, 4, 6	Unique	**System Call Target Address (R/W)** See Table 35-2.
C000_ 0082H		IA32_LSTAR	3, 4, 6	Unique	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_ 0084H		IA32_FMASK	3, 4, 6	Unique	**System Call Flag Mask (R/W)** See Table 35-2.
C000_ 0100H		IA32_FS_BASE	3, 4, 6	Unique	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_ 0101H		IA32_GS_BASE	3, 4, 6	Unique	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0102H		IA32_KERNEL_GSBASE	3, 4, 6	Unique	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.

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.

...

35.19 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 35-44. 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 35-44 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 35.22, "MSRs in Pentium Processors," and see Table 35-2.
1H	1	P5_MC_TYPE	Unique	See Section 35.22, "MSRs in Pentium Processors," and see Table 35-2.
6H	6	IA32_MONITOR_FILTER_ SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and see Table 35-2.

Table 35-44 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			
10H	16	IA32_TIME_STAMP_ COUNTER	Unique	See Section 17.15, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Shared	**Platform ID (R)** See Table 35-2. 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 35-2.
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 processor implements R/W.
		2		**Response Error Checking Enable (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		**MCERR# Drive Enable (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		**Address Parity Enable (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		6: 5		Reserved
		7		**BINIT# Driver Enable (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		**Output Tri-state Enabled (R/O)** 1 = Enabled; 0 = Disabled
		9		**Execute BIST (R/O)** 1 = Enabled; 0 = Disabled
		10		**MCERR# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled
		11		Reserved
		12		**BINIT# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled

Table 35-44 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			
		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)**
3AH	58	IA32_FEATURE_CONTROL	Unique	**Control Features in IA-32 Processor (R/W)** See Table 35-2.
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: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.13, "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 35-2.

Table 35-44 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			
8BH	139	IA32_BIOS_SIGN_ID	Unique	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Unique	**Performance counter register** See Table 35-2.
C2H	194	IA32_PMC1	Unique	**Performance counter register** See Table 35-2.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed (RO)** This field indicates the scaleable bus clock speed:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count. (RW)** See Table 35-2.
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count. (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 35-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		**L2 Hardware Enabled (RO)** 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 (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.

Table 35-44 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			
174H	372	IA32_SYSENTER_CS	Unique	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 35-2.
179H	377	IA32_MCG_CAP	Unique	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP** When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSEL0	Unique	See Table 35-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 35-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 35-2.
199H	409	IA32_PERF_CTL	Unique	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	**Clock Modulation (R/W)** See Table 35-2.
19BH	411	IA32_THERM_INTERRUPT	Unique	**Thermal Interrupt Control (R/W)** See Table 35-2. See Section 14.7.2, “Thermal Monitor.”
19CH	412	IA32_THERM_STATUS	Unique	**Thermal Monitor Status (R/W)** See Table 35-2. See Section 14.7.2, “Thermal Monitor”.
19DH	413	MSR_THERM2_CTL	Unique	
		15:0		Reserved.

Table 35-44 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			
		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 35-2.
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available (R)** See Table 35-2.
		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 (RO)** See Table 35-2.
		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.

Table 35-44 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			
		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 35-2.
		19		**Reserved.**
		22	Shared	**Limit CPUID Maxval (R/W)** See Table 35-2. 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 35-2.
		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).
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control (R/W)** Controls how several debug features are used. Bit definitions are discussed in the referenced section.
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.
1E0H	480	ROB_CR_ BKUPTMPDR6	Unique	
		1:0		Reserved.
		2		Fast String Enable bit. (Default, enabled)
200H	512	MTRRphysBase0	Unique	
201H	513	MTRRphysMask0	Unique	
202H	514	MTRRphysBase1	Unique	
203H	515	MTRRphysMask1	Unique	

Table 35-44 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			
204H	516	MTRRphysBase2	Unique	
205H	517	MTRRphysMask2	Unique	
206H	518	MTRRphysBase3	Unique	
207H	519	MTRRphysMask3	Unique	
208H	520	MTRRphysBase4	Unique	
209H	521	MTRRphysMask4	Unique	
20AH	522	MTRRphysBase5	Unique	
20BH	523	MTRRphysMask5	Unique	
20CH	524	MTRRphysBase6	Unique	
20DH	525	MTRRphysMask6	Unique	
20EH	526	MTRRphysBase7	Unique	
20FH	527	MTRRphysMask7	Unique	
250H	592	MTRRfix64K_00000	Unique	
258H	600	MTRRfix16K_80000	Unique	
259H	601	MTRRfix16K_A0000	Unique	
268H	616	MTRRfix4K_C0000	Unique	
269H	617	MTRRfix4K_C8000	Unique	
26AH	618	MTRRfix4K_D0000	Unique	
26BH	619	MTRRfix4K_D8000	Unique	
26CH	620	MTRRfix4K_E0000	Unique	
26DH	621	MTRRfix4K_E8000	Unique	
26EH	622	MTRRfix4K_F0000	Unique	
26FH	623	MTRRfix4K_F8000	Unique	
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	**Default Memory Types (R/W)** See Table 35-2. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MC0_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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."

Table 35-44 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			
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 ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	MSR_MC3_MISC	Unique	
414H	1044	MSR_MC5_CTL	Unique	
415H	1045	MSR_MC5_STATUS	Unique	
416H	1046	MSR_MC5_ADDR	Unique	
417H	1047	MSR_MC5_MISC	Unique	
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, "Basic VMX Information" (If CPUID.01H:ECX.[bit 9])
481H	1153	IA32_VMX_PINBASED_ CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls" (If CPUID.01H:ECX.[bit 9])

Table 35-44 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			
482H	1154	IA32_VMX_PROCBASED_ CTLS	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 9])
483H	1155	IA32_VMX_EXIT_CTLS	Unique	**Capability Reporting Register of VM-exit Controls (R/O)** See Appendix A.4, "VM-Exit Controls" (If CPUID.01H:ECX.[bit 9])
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls (R/O)** See Appendix A.5, "VM-Entry Controls" (If CPUID.01H:ECX.[bit 9])
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 9])
486H	1158	IA32_VMX_CR0_FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0" (If CPUID.01H:ECX.[bit 9])
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 9])
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 9])
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 9])
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 9])
48BH	1163	IA32_VMX_PROCBASED_ CTLS2	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 9] and IA32_VMX_PROCBASED_CTLS[bit 63])
600H	1536	IA32_DS_AREA	Unique	**DS Save Area (R/W)** See Table 35-2. See Section 18.12.4, "Debug Store (DS) Mechanism."
		31:0		**DS Buffer Management Area** Linear address of the first byte of the DS buffer management area.

Table 35-44 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			
		63:32		Reserved.
C000_ 0080H		IA32_EFER	Unique	See Table 35-2.
		10:0		Reserved.
		11		**Execute Disable Bit Enable**
		63:12		Reserved.

35.20 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 35.21 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table 35-45 MSRs in Pentium M Processors

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 35.22, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Section 35.22, "MSRs in Pentium Processors."
10H	16	IA32_TIME_STAMP_COUNTER	See Section 17.15, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	**Platform ID (R)** See Table 35-2. 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.

Table 35-45 MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		4	**Address Parity Enable (R)** 0 = Disabled Always 0 on the Pentium M processor.
		6:5	Reserved.
		7	**BINIT# Driver Enable (R)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		8	**Output Tri-state Enabled (R/O)** 1 = Enabled; 0 = Disabled
		9	**Execute BIST (R/O)** 1 = Enabled; 0 = Disabled
		10	**MCERR# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		11	Reserved.
		12	**BINIT# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		13	Reserved.
		14	**1 MByte Power on Reset Vector (R/O)** 1 = 1 MByte; 0 = 4 GBytes Always 0 on the Pentium M processor.
		15	Reserved.
		17:16	**APIC Cluster ID (R/O)** Always 00B on the Pentium M processor.
		18	**System Bus Frequency (R/O)** 0 = 100 MHz 1 = Reserved Always 0 on the Pentium M processor.
		19	Reserved.
		21: 20	**Symmetric Arbitration ID (R/O)** Always 00B on the Pentium M processor.
		26:22	Clock Frequency Ratio (R/O)

Table 35-45 MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
40H	64	MSR_LASTBRANCH_0	**Last Branch Record 0 (R/W)** One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.13, "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	
		63:0	Reserved.
11EH	281	MSR_BBL_CR_CTL3	
		0	**L2 Hardware Enabled (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		4:1	Reserved.
		5	**ECC Check Enable (RO)** 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.

Table 35-45 MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	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 (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24	Reserved.
179H	377	IA32_MCG_CAP	
		7:0	**Count (RO)** Indicates the number of hardware unit error reporting banks available in the processor.
		8	**IA32_MCG_CTL Present (RO)** 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	
		0	**RIPV** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1	**EIPV** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2	**MCIP** When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3	Reserved.
198H	408	IA32_PERF_STATUS	See Table 35-2.
199H	409	IA32_PERF_CTL	See Table 35-2.

Table 35-45 MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
19AH	410	IA32_CLOCK_MODULATION	**Clock Modulation (R/W).** See Table 35-2. See Section 14.7.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	**Thermal Interrupt Control (R/W)** See Table 35-2. See Section 14.7.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	**Thermal Monitor Status (R/W)** See Table 35-2. See Section 14.7.2, "Thermal Monitor."
19DH	413	MSR_THERM2_CTL	
		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	**Automatic Thermal Control Circuit Enable (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
		6:4	Reserved.
		7	**Performance Monitoring Available (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled

Table 35-45 MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		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 (RO) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
		12	**Processor Event Based Sampling Unavailable (RO)** 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.
1C9H	457	MSR_LASTBRANCH_TOS	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: ▪ MSR_LASTBRANCH_0_FROM_IP (at 40H) ▪ Section 17.13, "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 17.13, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."

Table 35-45 MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
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 17.13, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.14.2, "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 17.13, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.14.2, "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_MC0_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	See Section 14.3.2.3., "IA32_MCi_ADDR MSRs". The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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."
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 ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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."

Table 35-45 MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
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 ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
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 ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
600H	1536	IA32_DS_AREA	**DS Save Area (R/W)** See Table 35-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.12.4, “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.

...

35.23 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 Name and CPUID DisplayFamily_DisplayModel	**Location**
06_2EH	See Table 35-15
MSR_B1_PMON_EVNT_SEL1	
06_2EH	See Table 35-15
MSR_B1_PMON_EVNT_SEL2	
06_2EH	See Table 35-15
MSR_B1_PMON_EVNT_SEL3	
06_2EH	See Table 35-15
MSR_B1_PMON_MASK	
06_2EH	See Table 35-15
MSR_B1_PMON_MATCH	
06_2EH	See Table 35-15
MSR_BBL_CR_CTL	
06_09H	See Table 35-45
MSR_BBL_CR_CTL3	
06_0FH, 06_17H	See Table 35-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_0EH	See Table 35-44
06_09H	See Table 35-45
MSR_BPU_CCCR0	
0FH	See Table 35-41
MSR_BPU_CCCR1	
0FH	See Table 35-41
MSR_BPU_CCCR2	
0FH	See Table 35-41
MSR_BPU_CCCR3	
0FH	See Table 35-41
MSR_BPU_COUNTER0	
0FH	See Table 35-41
MSR_BPU_COUNTER1	
0FH	See Table 35-41
MSR_BPU_COUNTER2	
0FH	See Table 35-41
MSR_BPU_COUNTER3	
0FH	See Table 35-41
MSR_BPU_ESCR0	
0FH	See Table 35-41
MSR_BPU_ESCR1	
0FH	See Table 35-41

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	**Location**
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_CTR1	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_CTR2	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_CTR3	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_CTR4	
06_2FH	See Table 35-17
MSR_C8_PMON_CTR5	
06_2FH	See Table 35-17
MSR_C8_PMON_EVNT_SEL0	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_EVNT_SEL1	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_EVNT_SEL2	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_EVNT_SEL3	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C8_PMON_EVNT_SEL4	
06_2FH	See Table 35-17
MSR_C8_PMON_EVNT_SEL5	
06_2FH	See Table 35-17
MSR_C9_PMON_BOX_CTRL	

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C9_PMON_EVNT_SEL1	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C9_PMON_EVNT_SEL2	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C9_PMON_EVNT_SEL3	
06_2FH	See Table 35-17
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_C9_PMON_EVNT_SEL4	
06_2FH	See Table 35-17
MSR_C9_PMON_EVNT_SEL5	
06_2FH	See Table 35-17
MSR_CC6_DEMOTION_POLICY_CONFIG	
06_37H	See Table 35-9
MSR_CONFIG_TDP_CONTROL	
06_3AH	See Table 35-23
06_3CH, 06_45H, 06_46H	See Table 35-27
06_57H	See Table 35-40
MSR_CONFIG_TDP_LEVEL1	
06_3AH	See Table 35-23
06_3CH, 06_45H, 06_46H	See Table 35-27
06_57H	See Table 35-40
MSR_CONFIG_TDP_LEVEL2	
06_3AH.	See Table 35-23
06_3CH, 06_45H, 06_46H	See Table 35-27
06_57H	See Table 35-40
MSR_CONFIG_TDP_NOMINAL	
06_3AH	See Table 35-23
06_3CH, 06_45H, 06_46H	See Table 35-27
06_57H	See Table 35-40
MSR_CORE_C1_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
MSR_CORE_C3_RESIDENCY	
06_5CH.	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 35-13

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
MSR_CORE_C6_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 35-13
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
06_57H	See Table 35-40
MSR_CORE_C7_RESIDENCY	
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
MSR_CORE_GFXE_OVERLAP_C0	
06_4EH, 06_5EH	See Table 35-37
MSR_CORE_HDC_RESIDENCY	
06_4EH, 06_5EH	See Table 35-37
MSR_CORE_PERF_LIMIT_REASONS	
06_5CH.	See Table 35-12
06_3CH, 06_45H, 06_46H	See Table 35-28
06_3F	See Table 35-30
06_56H, 06_4FH	See Table 35-34
06_57H	See Table 35-40
MSR_CRU_ESCR0	
0FH	See Table 35-41
MSR_CRU_ESCR1	
0FH	See Table 35-41
MSR_CRU_ESCR2	
0FH	See Table 35-41
MSR_CRU_ESCR3	
0FH	See Table 35-41
MSR_CRU_ESCR4	
0FH	See Table 35-41
MSR_CRU_ESCR5	
0FH	See Table 35-41
MSR_DAC_ESCR0	
0FH	See Table 35-41
MSR_DAC_ESCR1	
0FH	See Table 35-41
MSR_DRAM_ENERGY_ STATUS	
06_5CH.	See Table 35-12
06_2DH	See Table 35-21
06_3EH, 06_3FH	See Table 35-24
06_3CH, 06_45H, 06_46H	See Table 35-27

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_27_TO_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_28_FROM_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_28_TO_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_29_FROM_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_29_TO_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_3	
0FH	See Table 35-41
06_0EH	See Table 35-44
06_09H	See Table 35-45
MSR_LASTBRANCH_3_FROM_IP	
06_0FH, 06_17H	See Table 35-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_3_TO_IP	
06_0FH, 06_17H	See Table 35-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_30_FROM_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_30_TO_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_31_FROM_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_31_TO_IP	
06_5CH	See Table 35-12
06_4EH, 06_5EH	See Table 35-37
MSR_LASTBRANCH_4	
06_0EH	See Table 35-44
06_09H	See Table 35-45
MSR_LASTBRANCH_4_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_4_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_5	
06_0EH	See Table 35-44
06_09H	See Table 35-45
MSR_LASTBRANCH_5_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_5_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_8_TO_IP	
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_9_FROM_IP	
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_9_TO_IP	
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
0FH	See Table 35-41
MSR_LASTBRANCH_TOS	
06_0FH, 06_17H	See Table 35-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
06_4EH, 06_5EH	See Table 35-37
06_57H	See Table 35-40
06_0EH	See Table 35-44
06_09H	See Table 35-45
MSR_LBR_INFO_1	
06_4EH, 06_5EH	See Table 35-37
MSR_LBR_INFO_10	
06_4EH, 06_5EH	See Table 35-37
MSR_LBR_INFO_11	
06_4EH, 06_5EH	See Table 35-37
MSR_LBR_INFO_12	
06_4EH, 06_5EH	See Table 35-37
MSR_LBR_INFO_13	

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC10_CTL / MSR_MC10_CTL	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC10_MISC / MSR_MC10_MISC	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC10_STATUS / MSR_MC10_STATUS	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC11_ADDR / MSR_MC11_ADDR	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC11_CTL / MSR_MC11_CTL	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC11_MISC / MSR_MC11_MISC	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC11_STATUS / MSR_MC11_STATUS	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC12_ADDR / MSR_MC12_ADDR	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC12_CTL / MSR_MC12_CTL	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC12_MISC / MSR_MC12_MISC	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC12_STATUS / MSR_MC12_STATUS	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC13_ADDR / MSR_MC13_ADDR	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC13_CTL / MSR_MC13_CTL	

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC18_ADDR / MSR_MC18_ADDR	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC18_CTL / MSR_MC18_CTL	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC18_MISC / MSR_MC18_MISC	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC18_STATUS / MSR_MC18_STATUS	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC19_ADDR / MSR_MC19_ADDR	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC19_CTL / MSR_MC19_CTL	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24

MSR Name and CPUID DisplayFamily_DisplayModel	**Location**
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC19_MISC / MSR_MC19_MISC	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC19_STATUS / MSR_MC19_STATUS	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC2_MISC / MSR_MC2_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
IA32_MC20_ADDR / MSR_MC20_ADDR	
06_2EH	See Table 35-15
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC20_CTL / MSR_MC20_CTL	
06_2EH	See Table 35-15
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC20_MISC / MSR_MC20_MISC	
06_2EH	See Table 35-15
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC20_STATUS / MSR_MC20_STATUS	
06_2EH	See Table 35-15
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 35-25
IA32_MC30_STATUS / MSR_MC30_STATUS	
06_3EH	See Table 35-25
IA32_MC31_ADDR / MSR_MC31_ADDR	
06_3EH	See Table 35-25
IA32_MC31_CTL / MSR_MC31_CTL	
06_3EH	See Table 35-25
IA32_MC31_MISC / MSR_MC31_MISC	
06_3EH	See Table 35-25
IA32_MC31_STATUS / MSR_MC31_STATUS	
06_3EH	See Table 35-25
IA32_MC4_ADDR / MSR_MC4_ADDR	
06_0FH, 06_17H	See Table 35-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_57H	See Table 35-40
06_0EH	See Table 35-44
06_09H	See Table 35-45
IA32_MC4_CTL / MSR_MC4_CTL	
06_0FH, 06_17H	See Table 35-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_57H	See Table 35-40
06_0EH	See Table 35-44
06_09H	See Table 35-45
IA32_MC4_CTL2 / MSR_MC4_CTL2	
06_2AH, 06_2DH	See Table 35-18
IA32_MC4_STATUS / MSR_MC4_STATUS	
06_0FH, 06_17H	See Table 35-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 35-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_57H	See Table 35-40
06_0EH	See Table 35-44
06_09H	See Table 35-45
MSR_MC5_ADDR / MSR_MC5_ADDR	
06_0FH, 06_17H	See Table 35-3

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC6_CTL / MSR_MC6_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
MSR_MC6_DEMOTION_POLICY_CONFIG	
06_37H	See Table 35-9
IA32_MC6_MISC / MSR_MC6_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
MSR_MC6_RESIDENCY_COUNTER	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_37H	See Table 35-9
06_57H	See Table 35-40
IA32_MC6_STATUS / MSR_MC6_STATUS	
06_0FH, 06_17H	See Table 35-3
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3FH	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC7_ADDR / MSR_MC7_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36

MSR Name and CPUID DisplayFamily_DisplayModel	**Location**
IA32_MC8_STATUS / MSR_MC8_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_4FH	See Table 35-36
IA32_MC9_ADDR / MSR_MC9_ADDR	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC9_CTL / MSR_MC9_CTL	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC9_MISC / MSR_MC9_MISC	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
IA32_MC9_STATUS / MSR_MC9_STATUS	
06_2EH	See Table 35-15
06_2DH	See Table 35-21
06_3EH	See Table 35-24
06_3F	See Table 35-30
06_56H	See Table 35-35
06_4FH	See Table 35-36
MSR_MCG_MISC	
0FH	See Table 35-41
MSR_MCG_R10	
0FH	See Table 35-41
MSR_MCG_R11	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_MISC_PWR_MGMT	
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
MSR_MOB_ESCR0	
0FH	See Table 35-41
MSR_MOB_ESCR1	
0FH	See Table 35-41
MSR_MS_CCCR0	
0FH	See Table 35-41
MSR_MS_CCCR1	
0FH	See Table 35-41
MSR_MS_CCCR2	
0FH	See Table 35-41
MSR_MS_CCCR3	
0FH	See Table 35-41
MSR_MS_COUNTER0	
0FH	See Table 35-41
MSR_MS_COUNTER1	
0FH	See Table 35-41
MSR_MS_COUNTER2	
0FH	See Table 35-41
MSR_MS_COUNTER3	
0FH	See Table 35-41
MSR_MS_ESCR0	
0FH	See Table 35-41
MSR_MS_ESCR1	
0FH	See Table 35-41
MSR_MTRRCAP	
06_4EH, 06_5EH	See Table 35-37
MSR_OFFCORE_RSP_0	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
06_57H	See Table 35-40
MSR_OFFCORE_RSP_1	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
06_25H, 06_2CH	See Table 35-16
06_2FH	See Table 35-17

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_45H	See Table 35-28 and Table 35-29
06_4FH	See Table 35-36
MSR_PKG_C2_RESIDENCY	
06_27H	See Table 35-5
06_5CH	See Table 35-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
06_57H	See Table Table 35-40
MSR_PKG_C3_RESIDENCY	
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 35-13
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-12
06_57H	See Table 35-40
MSR_PKG_C4_RESIDENCY	
06_27H	See Table 35-5
MSR_PKG_C6_RESIDENCY	
06_27H	See Table 35-5
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 35-13
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
06_57H	See Table 35-40
MSR_PKG_C7_RESIDENCY	
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 35-13
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
06_57H	See Table 35-40
MSR_PKG_C8_RESIDENCY	
06_45H	See Table 35-29
06_4FH	See Table 35-36
MSR_PKG_C9_RESIDENCY	
06_45H	See Table 35-29
06_4FH	See Table 35-36
MSR_PKG_CST_CONFIG_CONTROL	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-7
06_4CH	See Table 35-11
06_5CH	See Table 35-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3AH	See Table 35-23
06_3EH	See Table 35-24
06_3CH, 06_45H, 06_46H	See Table 35-28
06_45H	See Table 35-29
06_3F	See Table 35-30
06_3DH	See Table 35-33
06_56H, 06_4FH	See Table 35-34
06_57H	See Table 35-40
MSR_PKG_ENERGY_STATUS	
06_37H, 06_4AH, 06_5AH, 06_5DH	See Table 35-8
06_5CH	See Table 35-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
MSR_PKG_HDC_CONFIG	
06_4EH, 06_5EH	See Table 35-37
MSR_PKG_HDC_DEEP_RESIDENCY	
06_4EH, 06_5EH	See Table 35-37
MSR_PKG_HDC_SHALLOW_RESIDENCY	
06_4EH, 06_5EH	See Table 35-37
MSR_PKG_PERF_STATUS	
06_5CH	See Table 35-12
06_2DH	See Table 35-21
06_3EH, 06_3FH	See Table 35-24
06_3CH, 06_45H, 06_46H	See Table 35-28
06_57H	See Table 35-40
MSR_PKG_POWER_INFO	
06_4DH	See Table 35-10
06_5CH	See Table 35-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
06_57H	See Table 35-40
MSR_PKG_POWER_LIMIT	
06_37H, 06_4AH, 06_5AH, 06_5DH	See Table 35-8
06_4DH	See Table 35-10
06_5CH	See Table 35-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 35-18
06_57H	See Table 35-40
MSR_PKGC_IRTL1	
06_5CH	See Table 35-12
06_3CH, 06_45H, 06_46H.	See Table 35-27
MSR_PKGC_IRTL2	

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_57H	See Table 35-40
MSR_SMM_BLOCKED	
06_5CH	See Table 35-12
06_3CH, 06_45H, 06_46H	See Table 35-28
MSR_SMM_DELAYED	
06_5CH	See Table 35-12
06_3CH, 06_45H, 06_46H	See Table 35-28
MSR_SMM_FEATURE_CONTROL	
06_5CH	See Table 35-12
06_3CH, 06_45H, 06_46H	See Table 35-28
MSR_SMM_MCA_CAP	
06_5CH	See Table 35-12
06_3CH, 06_45H, 06_46H	See Table 35-28
06_3F	See Table 35-30
06_56H, 06_4FH	See Table 35-34
MSR_SMRR_PHYSBASE	
06_0FH, 06_17H	See Table 35-3
MSR_SMRR_PHYSMASK	
06_0FH, 06_17H	See Table 35-3
MSR_SSU_ESCR0	
0FH	See Table 35-41
MSR_TBPU_ESCR0	
0FH	See Table 35-41
MSR_TBPU_ESCR1	
0FH	See Table 35-41
MSR_TC_ESCR0	
0FH	See Table 35-41
MSR_TC_ESCR1	
0FH	See Table 35-41
MSR_TC_PRECISE_EVENT	
0FH	See Table 35-41
MSR_TEMPERATURE_TARGET	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 35-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 35-13
06_2AH, 06_2DH	See Table 35-18
06_3EH	See Table 35-24
06_56H, 06_4FH	See Table 35-34
06_57H	See Table 35-40
MSR_THERM2_CTL	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_TURBO_RATIO_LIMIT3	
06_56H	See Table 35-35
06_4FH	See Table 35-36
MSR_U_PMON_BOX_STATUS	
06_3EH	See Table 35-26
06_3FH	See Table 35-31
MSR_U_PMON_CTR	
06_2EH	See Table 35-15
MSR_U_PMON_CTR0	
06_2DH	See Table 35-22
06_3FH	See Table 35-31
MSR_U_PMON_CTR1	
06_2DH	See Table 35-22
06_3FH	See Table 35-31
MSR_U_PMON_EVNT_SEL	
06_2EH	See Table 35-15
MSR_U_PMON_EVNTSEL0	
06_2DH	See Table 35-22
06_3FH	See Table 35-31
MSR_U_PMON_EVNTSEL1	
06_2DH	See Table 35-22
06_3FH	See Table 35-31
MSR_U_PMON_GLOBAL_CTRL	
06_2EH	See Table 35-15
MSR_U_PMON_GLOBAL_OVF_CTRL	
06_2EH	See Table 35-15
MSR_U_PMON_GLOBAL_STATUS	
06_2EH	See Table 35-15
MSR_U_PMON_UCLK_FIXED_CTL	
06_2DH	See Table 35-22
06_3FH	See Table 35-31
MSR_U_PMON_UCLK_FIXED_CTR	
06_2DH	See Table 35-22
06_3FH	See Table 35-31
MSR_U2L_ESCR0	
0FH	See Table 35-41
MSR_U2L_ESCR1	
0FH	See Table 35-41
MSR_UNC_ARB_PERFCTR0	

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel **Location**

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MTRRphysMask7	
06_0EH	See Table 35-44
P6 Family	See Table 35-46
ROB_CR_BKUPTMPDR6	
06_0EH	See Table 35-44
P6 Family	See Table 35-46

...

28. Updates to Chapter 36, Volume 3C

Change bars show changes to Chapter 36 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

36.2.6.2 Virtual-Machine Extensions (VMX)

Initial implementations of Intel Processor Trace do not support tracing in VMX operation. Such processors indicate this by returning 0 for IA32_VMX_MISC[bit 14]. On these processors, execution of the VMXON instruction clears IA32_RTIT_CTL.TraceEn and any attempt to set that bit in VMX operation using WRMSR causes a general-protection exception (#GP).

Processors that support Intel Processor Trace in VMX operation return 1 for IA32_VMX_MISC[bit 14]. Details of tracing in VMX operation are described in Section 36.5.

...

36.3.7 Decoder Synchronization (PSB+)

The PSB packet (Section 36.4.2.17) serves as a synchronization point for a trace-packet decoder. It is a pattern in the trace log for which the decoder can quickly scan to align packet boundaries. No legal packet combination can result in such a byte sequence. As such, it serves as the starting point for packet decode. To decode a trace log properly, the decoder needs more than simply to be aligned: it needs to know some state and potentially some timing information as well. The decoder should never need to retain any information (e.g., LastIP, call stack, compound packet event) across a PSB; all compound packet events will be completed before a PSB, and any compression state will be reset.

When a PSB packet is generated, it is followed by a PSBEND packet (Section 36.4.2.18). One or more packets may be generated in between those two packets, and these inform the decoder of the current state of the processor. These packets, known collectively as PSB+, should be interpreted as "status only", since they do not imply any change of state at the time of the PSB, nor are they associated directly with any instruction or event. Thus, the normal binding and ordering rules that apply to these packets outside of PSB+ can be ignored when these packets are between a PSB and PSBEND. They inform the decoder of the state of the processor at the time of the PSB.

PSB+ can include:

- Timestamp (TSC), if IA32_RTIT_CTL.TSCEn=1.
- Timestamp-MTC Align (TMA), if IA32_RTIT_CTL.TSCEn=1 && IA32_RTIT_CTL.MTCEn=1.
- Paging Info Packet (PIP), if ContextEn=1 and IA32_RTIT_CTL.OS=1. The non-root bit (NR) is set if the logical processor is in VMX non-root operation and the “conceal VMX non-root operation from Intel PT”. VM-execution control is 0.
- VMCS packet, if either the logical is in VMX root operation or the logical processor is in VMX non-root operation and the “conceal VMX non-root operation from Intel PT” VM-execution control is 0.
- Core Bus Ratio (CBR).
- MODE.TSX, if ContextEn=1 and BranchEn = 1.
- MODE.Exec, if PacketEn=1.
- Flow Update Packet (FUP), if PacketEn=1.

PSB is generated only when TriggerEn=1; hence PSB+ has the same dependencies. The ordering of packets within PSB+ is not fixed. Timing packets such as CYC and MTC may be generated between PSB and PSBEND, and their meanings are the same as outside PSB+.

Note that an overflow can occur during PSB+, and this could cause the PSBEND packet to be lost. For this reason, the OVF packet should also be viewed as terminating PSB+.

...

36.4.2.7 Paging Information (PIP) Packet

Table 36-26 PIP Packet Definition

<table>
<tr><td>Name</td><td colspan="3">Paging Information (PIP) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td colspan="7">CR3[11:5] or 0</td><td>RSVD/NR</td></tr>
<tr><td>3</td><td colspan="8">CR3[19:12]</td></tr>
<tr><td>4</td><td colspan="8">CR3[27:20]</td></tr>
<tr><td>5</td><td colspan="8">CR3[35:28]</td></tr>
<tr><td>6</td><td colspan="8">CR3[43:36]</td></tr>
<tr><td>7</td><td colspan="8">CR3[51:44]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn && IA32_RTIT_CTL.OS</td><td>Generation Scenario</td><td>MOV CR3, Task switch, INIT, SIPI, PSB+;
If IA32_VMX_MISC[bit 14] reports 1: VM exit, VM entry</td></tr>
<tr><td>Description</td><td colspan="3">The CR3 payload shown includes only the address portion of the CR3 value. For PAE paging, CR3[11:5] are thus included. For other page modes (32-bit and IA-32e paging), these bits are 0.
This packet holds the CR3 address value. It will be generated on operations that modify CR3:
▪ MOV CR3 operation
▪ Task Switch
▪ INIT and SIPI
▪ VM exit and VM entry, if appropriate controls in the VMCS are clear (see Section 36.5.1)
PIPs are not generated, despite changes to CR3, on SMI and RSM. This is due to the special behavior on these operations, see Section for details. Note that, for some cases of task switch where CR3 is not modified, no PIP will be produced.
The purpose of the PIP is to indicate to the decoder which application is running, so that it can apply the proper binaries to the linear addresses that are being traced.
The PIP packet contains the new CR3 value when CR3 is written.
PIPs generated by VM entries set the NR bit. PIPs generated in VMX non-root operation set the NR bit if the “conceal VMX non-root operation from Intel PT” VM-execution control is 0. All other PIPs clear the NR bit.</td></tr>
<tr><td>Application</td><td colspan="3">The purpose of the PIP packet is to help the decoder uniquely identify what software is running at any given time. When a PIP is encountered, a decoder should do the following:
1) If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this PIP is part of a compound packet event (Section 36.4.1). Find the ending TIP and apply the new CR3/NR values to the TIP payload IP.
2) Otherwise, look for the next MOV CR3, far branch, or VMRESUME/VMLAUNCH in the disassembly, and apply the new CR3 to the next (or target) IP.
For examples of the packets generated by these flows, see Section 36.7.</td></tr>
</table>

...

36.4.2.15 VMCS Packet

Table 36-34 VMCS Packet Definition

<table>
<tr><td>Name</td><td colspan="3">VMCS Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3"><table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>2</td><td colspan="8">VMCS Base Address [19:12]</td></tr>
<tr><td>3</td><td colspan="8">VMCS Base Address [27:20]</td></tr>
<tr><td>4</td><td colspan="8">VMCS Base Address [35:28]</td></tr>
<tr><td>5</td><td colspan="8">VMCS Base Address [43:36]</td></tr>
<tr><td>6</td><td colspan="8">VMCS Base Address [51:44]</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn; Also in VMX operation.</td><td>Generation Scenario</td><td>Generated on successful VMPTRLD, and optionally on SMM VM exits and VM entries that return from SMM (see Section 36.5).</td></tr>
<tr><td>Description</td><td colspan="3">The VMCS packet provides an address related to a VMCS pointer for a decoder to determine the transition of code contexts:
▪ On a successful VMPTRLD (i.e, a VMPTRLD that doesn’t fault, fail, or VM exit), the VMCS packet contains the address of the current working VMCS pointer of the logical processor that will execute a VM guest context.
▪ On SMM VM exits, the VMCS packet provides the STM VMCS base address (SMM Transfer VMCS pointer), if VMCS-based controls are clear (see Section 36.5.1). See Section 36.6 on tracing inside and outside STM.
▪ On VM entries that return from SMM, the VMCS packet provides the current working VMCS pointer of the guest VM (see Section 36.6), if VMCS-based controls are clear (see Section 36.5.1). Root versus Non-Root operation can be distinguished from the PIP.NR bit.
If a VMCS packet is generated before a VMCS has been loaded, or after it has been cleared, the base address value will be all 1s.
VMCS packets will not be seen on processors with IA32_VMX_MISC[bit 14]=0, as these processors do not allow TraceEn to be set in VMX operation.</td></tr>
<tr><td>Application</td><td colspan="3">The purpose of the VMCS packet is to help the decoder uniquely identify changes in the executing software context in situations that CR3 may not be unique.
When a VMCS is encountered, a decoder should do the following:
▪ If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this VMCS is part of a compound packet event (Section 36.4.1). Find the ending TIP and apply the new VMCS base pointer value to the TIP payload IP.
▪ Otherwise, look for the next VMPTRLD, VMRESUME, or VMLAUNCH in the disassembly, and apply the new VMCS base pointer on the next VM entry.
For examples of the packets generated by these flows, see Section 36.7.</td></tr>
</table>

...

36.5 TRACING IN VMX OPERATION

On processors that IA32_VMX_MISC[bit 14] reports 1, TraceEn can be set in VMX operation. A series of mechanisms exist to allow the VMM to configure tracing based on the desired trace domain, and on the consumer of the trace output. The VMM can configure specific VM-execution controls to control what virtualization-specific data are included within the trace packets (see Section 36.5.1 for details). MSR save and load lists can be employed by the VMM to restrict tracing to the desired context (see Section 36.5.2 for details). These configuration options are summarized in Table 36-40. Table 36-40 covers common Intel PT usages while SMIs are handled by the default SMM treatment. Tracing with SMM Transfer Monitor is described in Section 36.6.

Table 36-40 Common Usages of Intel PT and VMX

Target Domain	Output Consumer	Virtualize Output	Configure VMCS Controls	TraceEN Configuration	Save/Restore MSR states of Trace Configuration
System-Wide (VMM + VMs)	Host	NA	Default Setting (no suppression)	WRMSR or XRSTORS by Host	NA
VMM Only	Intel PT Aware VMM	NA	Enable suppression	MSR load list to disable tracing in VM, enable tracing on VM exits	NA
VM Only	Intel PT Aware VMM	NA	Enable suppression	MSR load list to enable tracing in VM, disable tracing on VM exits	NA
Intel PT Aware Guest(s)	Per Guest	VMM adds trace output virtualization	Enable suppression	MSR load list to enable tracing in VM, disable tracing on VM exits	VMM Update guest state on XRSTORS-exiting VM exits

36.5.1 VMX-Specific Packets and VMCS Controls

In all of the usages of VMX and Intel PT, the decoder in the host or VMM context can identify the occurrences of VMX transitions with the aid of VMX-specific packets. Packets relevant to VMX fall into the follow two kinds:

- VMCS Packet: The VMX transitions of individual VM can be distinguished by a decoder using the base address field in a VMCS packet. The base address field stores the VMCS pointer address of a successful VMPTRLD. A VMCS packet is sent on a successful execution of VMPTRLD. See Section 36.4.2.15 for details.
- NonRoot (NR) bit field in PIP packet: PIP packets are generated with each VM entry/exit. The NR bit in a PIP packet is set when in VMX non-Root operation. Thus a transition of the NR bit from 0 to 1 indicates the occurrence of a VM entry, and a transition of 1 to 0 indicates the occurrence of a VM exit.

Processors with IA32_VMX_MISC[bit 14]= 1 also provides VMCS controls that a VMM can configure to prevent VMX-specific information from leaking across virtualization boundaries.

Table 36-41 VMCS Controls For Intel Processor Trace

Name	Type	Bit Position	Value	Behavior
Conceal VMX non-root operation from Intel PT	VM-execution control	19	0	PIPs generated in VM non-root operation will set the PIP.NR bit. PSB+ in VMX non-root operation will include the VMCS packet, to ensure that the decoder knows which guest is currently in use.
			1	PIPs generated in VMX non-root operation will not set the PIP.NR bit. PSB+ in VMX non-root operation will not include the VMCS packet.
Conceal VM exits from Intel PT	VM-exit control	24	0	PIPs are generated on VM exit, with NonRoot=0. On VM exit to SMM, VMCS packets are additionally generated.
			1	No PIP is generated on VM exit, and no VMCS packet is generated on VM exit to SMM.

Table 36-41 VMCS Controls For Intel Processor Trace

Name	Type	Bit Position	Value	Behavior
Conceal VM entries from Intel PT	VM-entry control	17	0	PIPs are generated on VM entry, with NonRoot=1 if the destination of the VM entry is VMX non-root operation. On VM entry to SMM, VMCS packets are additionally generated.
			1	No PIP is generated on VM entry, and no VMCS packet is generated on VM entry to SMM.

The default setting for the VMCS controls that interacts with Intel PT is to enable all VMX-specific packet information. The scenarios that would use the default setting also do not require the VMM to use MSR load list to manage the configuration of turning-on/off of trace packet generation across VM exits.

If IA32_VMX_MISC[bit 14] reports 0, any attempt to set the VMCS control bits in Table 36-41 will result in a failure on guest entry.

36.5.2 Managing Trace Packet Generation Across VMX Transitions

In tracing scenarios that collect packets for both VMX root operation and VMX non-root operation, a host executive can manage the MSRs associated with trace packet generation directly. The states of these MSRs need not be modified using MSR load list or MSR save list across VMX transitions.

For tracing scenarios that collect only packets within either VMX root operation or VMX non-root operation, the VMM can use the MSR load list and/or MSR save list to toggle IA32_RTIT_CTL.TraceEn.

36.5.2.1 System-Wide Tracing

When a host or VMM configures Intel PT to collect trace packets of the entire system, it can leave the VMCS controls clear to allow VMX-specific packets to provide information across VMX transitions. MSR load list is not used across VM exits or VM entries, nor is VM-exit MSR save list.

The decoder will desire to identify the occurrence of VMX transitions. The packets of interests to a decoder are shown in Table 36-42.

Table 36-42 Packets on VMX Transitions (System-Wide Tracing)

Event	Packets	Description
VM exit	FUP(GuestIP)	The FUP indicates at which point in the guest flow the VM exit occurred. This is important, since VM exit can be an asynchronous event. The IP will match that written into the VMCS.
	PIP(HostCR3, NR=0)	The PIP packet provides the new host CR3 value, as well as indication that the logical processor is entering VMX root operation. This allows the decoder to identify the change of executing context from guest to host and load the appropriate set of binaries to continue decode.
	TIP(HostIP)	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX root operation. Note, this packet could be preceded by a MODE.Exec packet (Section 36.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.

Table 36-42 Packets on VMX Transitions (System-Wide Tracing)

Event	Packets	Description
VM entry	PIP(GuestCR3, NR=1)	The PIP packet provides the new guest CR3 value, as well as indication that the logical processor is entering VMX non-root operation. This allows the decoder to identify the change of executing context from host to guest and load the appropriate set of binaries to continue decode.
	TIP(GuestIP)	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX non-root operation. This should match the IP value read out from the VMCS. Note, this packet could be preceded by a MODE.Exec packet (Section 36.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.

Since the packet suppression controls are cleared, the VMCS packet will be included in all PSB+ for this usage scenario. Thus the decoder can distinguish the execution context of different VMs. Additionally, it will be generated on VMPTRLD. Thus the decoder can distinguish the execution context of different VMs.

When the host VMM configures a system to collect trace packets in this scenario, it should emulate CPUID to report CPUID.(EAX=07H, ECX=0):EBX[bit 26] with 0 to guests, indicating to guests that Intel PT is not available.

VMX TSC Manipulation

The TSC packets generated while in VMX non-root operation will include any changes resulting from the use of a VMM's use of the TSC offsetting or TSC scaling VMCS control (see Chapter 25, "VMX Non-Root Operation"). In this system-wide usage model, the decoder may need to account for the effect of per-VM adjustments in the TSC packets generated in VMX non-root operation and the absence of TSC adjustments in TSC packets generated in VMX root operation. The VMM can supply this information to the decoder.

36.5.2.2 Host-Only Tracing

When trace packets in VMX non-root operation are not desired, the VMM can use VM-entry MSR load list with IA32_RTIT_CTL.TraceEn=0 to disable trace packet generation in guests, set IA32_RTIT_CTL.TraceEn=1 via VM-exit MSR load list.

When tracing only the host, the decoder does not need information about the guests, the VMCS controls for suppressing VMX-specific packets can be set to reduce the packets generated. VMCS packets will still be generated on successful VMPTRLD and in PSB+ generated in the Host, but these will be unused by the decoder.

The packets of interests to a decoder when trace packets are collected for host-only tracing are shown in Table 36-43.

Table 36-43 Packets on VMX Transitions (Host-Only Tracing)

Event	Packets	Description
VM exit	TIP.PGE(HostIP)	The TIP.PGE indicates that trace packet generation is enabled and gives the IP of the first instruction to be executed in VMX root operation. Note, this packet could be preceded by a MODE.Exec packet (Section 36.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.
VM entry	TIP.PGD()	The TIP indicates that trace packet generation was disabled. This ensure that all buffered packets are flushed out.

36.5.2.3 Guest-Only Tracing

A VMM can configure trace packet generation while in non-root operation for guests executing normally. This is accomplished by utilizing the MSR load lists across VM exit and VM entry to confine trace packet generation to stay within the guest environment.

For this usage, the VM-entry MSR load list is programmed to turn on trace packet generation. The VM-exit MSR load list is used to clear TraceEn=0 to disable trace packet generation in the host. Further, if it is preferred that the guest packet stream contain no indication that execution was in VMX non-root operation, the VMM should set the VMCS controls described in Table 36-41.

...

36.5.2.7 Failed VM Entry

The packets generated by a failed VM entry depend both on the VMCS configuration, as well as on the type of failure. The results to expect are summarized in the table below. Note that packets in *italics* may or may not be generated, depending on implementation choice, and the point of failure.

Table 36-44 Packets on a Failed VM Entry

Usage Model	Entry Configuration	Early Failure (fall through to Next IP)	Late Failure (VM exit)
System-Wide	No MSR load list	TIP (NextIP)	*PIP(Guest CR3, NR=1), TraceEn 0->1 Packets (See Section 36.2.5.3),* PIP(HostCR3, NR=0), TIP(HostIP)
VMM Only	MSR load list disables TraceEn	TIP (NextIP)	*TraceEn 0->1 Packets (See Section 36.2.5.3),* TIP(HostIP)
VM Only	MSR load list Enables TraceEn	None	None

36.5.2.8 VMX Abort

VMX abort conditions take the processor into a shutdown state. On a VM exit that leads to VMX abort, some packets (FUP, PIP) may be generated, but any expected TIP, TIP.PGE, or TIP.PGD may be dropped.

36.6 TRACING AND SMM TRANSFER MONITOR (STM)

SMM Transfer Monitor is a VMM that operates inside SMM while in VMX root operation. An STM operates in conjunction with an executive monitor. The latter operates outside SMM and in VMX root operation. Transitions from the executive monitor or its VMs to the STM are called SMM VM exits. The STM returns from SMM via a VM entry to the VM in VMX non-root operation or the executive monitor in VMX root operation.

Intel PT supports tracing in an STM similar to tracing support for VMX operation as described above in Section 36.7. As a result, on a SMM VM exit resulting from #SMI, TraceEn is not saved and then cleared. Software can save the state of the trace configuration MSRs and clear TraceEN using the MSR load/save lists.

...

36.7 PACKET GENERATION SCENARIOS

Table 36-45 illustrates the packets generated in various scenarios. In the heading row, PacketEn is abbreviated as PktEn, ContextEn as CntxEn. Note that this assumes that TraceEn=1 in IA32_RTIT_CTL, while TriggerEn=1 and

Error=0 in IA32_RTIT_STATUS, unless otherwise specified. Entries that do not matter in packet generation are marked "D.C."

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
1a	Normal non-jump operation	0	0	D.C.		None
1b	Normal non-jump operation	1	1	1		None
2a	WRMSR/XRSTORS/RSM that changes TraceEn 0 -> 1, with PacketByteCnt >0	0	0	D.C.	*TSC if TSCEn=1; *TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR
2b	WRMSR/XRSTORS/RSM that changes TraceEn 0 -> 1, with PacketByteCnt =0	0	0	D.C.	*TSC if TSCEn=1; *TMA if TSCEn=MTCEn=1	PSB, PSBEND (see Section 36.4.2.17)
2d	WRMSR/XRSTORS/RSM that changes TraceEn 0 -> 1, with PacketByteCnt >0	0	1	1	TSC if TSCEn=1; TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, MODE.Exec, TIP.PGE(NLIP)
2e	WRMSR/XRSTORS/RSM that changes TraceEn 0 -> 1, with PacketByteCnt =0	0	1	1		MODE.Exec, TIP.PGE(NLIP), PSB, PSBEND (see Section 36.4.2.8, 36.4.2.7, 36.4.2.13,36.4.2.15, 36.4.2.17)
3a	WRMSR that changes TraceEn 1 -> 0	0	0	D.C.		None
3b	WRMSR that changes TraceEn 1 -> 0	1	0	D.C.		FUP(CLIP), TIP.PGD()
5a	MOV to CR3	0	0	0		None
5f	MOV to CR3	0	0	1	TraceStop if executed in a TraceStop region	PIP(NewCR3,NR?), TraceStop?
5b	MOV to CR3	0	1	1	*PIP.NR=1 if not in root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0 *MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	PIP(NewCR3, NR?), MODE.Exec?, TIP.PGE(NLIP)
5c	MOV to CR3	1	0	0		TIP.PGD()
5e	MOV to CR3	1	0	1	*PIP.NR=1 if not in root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0 *TraceStop if executed in a TraceStop region	PIP(NewCR3, NR?), TIP.PGE(NLIP), TraceStop?
5d	MOV to CR3	1	1	1	*PIP.NR=1 if not in root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0	PIP(NewCR3, NR?)

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
6a	Unconditional direct near jump	0	0	D.C.		None
6b	Unconditional direct near jump	1	0	1	TraceStop if BLIP is in a TraceStop region	TIP.PGD(BLIP), TraceStop?
6c	Unconditional direct near jump	0	1	1	MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	MODE.Exec?, TIP.PGE(BLIP)
6d	Unconditional direct near jump	1	1	1		None
7a	Conditional taken jump or compressed RET that does not fill up the internal TNT buffer	0	0	D.C.		None
7b	Conditional taken jump or compressed RET	0	1	1	MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	MODE.Exec?, TIP.PGE(BLIP)
7e	Conditional taken jump or compressed RET, with empty TNT buffer	1	0	1	TraceStop if BLIP is in a TraceStop region	TIP.PGD(), TraceStop?
7f	Conditional taken jump or compressed RET, with non-empty TNT buffer	1	0	1	TraceStop if BLIP is in a TraceStop region	TNT, TIP.PGD(), TraceStop?
7d	Conditional taken jump or compressed RET that fills up the internal TNT buffer	1	1	1		TNT
8a	Conditional non-taken jump	0	0	D.C.		None
8d	Conditional not-taken jump that fills up the internal TNT buffer	1	1	1		TNT
9a	Near indirect jump (JMP, CALL, or uncompressed RET)	0	0	D.C.		None
9b	Near indirect jump (JMP, CALL, or uncompressed RET)	0	1	1	MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	MODE.Exec?, TIP.PGE(BLIP)
9c	Near indirect jump (JMP, CALL, or uncompressed RET)	1	0	1	TraceStop if BLIP is in a TraceStop region	TIP.PGD(BLIP), TraceStop?
9d	Near indirect jump (JMP, CALL, or uncompressed RET)	1	1	1		TIP(BLIP)
10a	Far Branch (CALL/JMP/RET)	0	0	0		None

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
10f	Far Branch (CALL/JMP/RET)	0	0	1	*PIP if CR3 is updated (i.e., task switch), and OS=1; *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *TraceStop if BLIP is in a TraceStop region	PIP(new CR3, NR?), TraceStop?
10b	Far Branch (CALL/JMP/RET)	0	1	1	*PIP if CR3 is updated (i.e., task switch), and OS=1; *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	PIP(new CR3, NR?), MODE.Exec?, TIP.PGE(BLIP)
10c	Far Branch (CALL/JMP/RET)	1	0	0		TIP.PGD()
10d	Far Branch (CALL/JMP/RET)	1	0	1	*PIP if CR3 is updated (i.e., task switch), and OS=1; *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *TraceStop if BLIP is in a TraceStop region	PIP(new CR3, NR?), TIP.PGD(BLIP), TraceStop?
10e	Far Branch (CALL/JMP/RET)	1	1	1	*PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; * MODE.Exec if the operation changes CS.L/D or IA32_EFER.LMA	PIP(NewCR3, NR?)?, MODE.Exec?, TIP(BLIP)
11a	HW Interrupt	0	0	0		None

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
11f	HW Interrupt	0	0	1	*PIP if CR3 is updated (i.e., task switch), and OS=1; *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *TraceStop if BLIP is in a TraceStop region	PIP(new CR3, NR?), TraceStop?
11b	HW Interrupt	0	1	1	*PIP if CR3 is updated (i.e., task switch), and OS=1; *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; * MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	PIP(new CR3, NR?), MODE.Exec?, TIP.PGE(BLIP)
11c	HW Interrupt	1	0	0		FUP(NLIP), TIP.PGD()
11d	HW Interrupt	1	0	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *TraceStop if BLIP is in a TraceStop region	FUP(NLIP), PIP(NewCR3, NR?)?, TIP.PGD(BLIP), TraceStop
11e	HW Interrupt	1	1	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; * MODE.Exec if the operation changes CS.L/D or IA32_EFER.LMA	FUP(NLIP), PIP(NewCR3, NR?)?, MODE.Exec?, TIP(BLIP)
12a	SW Interrupt	0	0	0		None

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
12f	SW Interrupt	0	0	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *TraceStop if BLIP is in a TraceStop region	PIP(NewCR3, NR?)?, TraceStop?
12b	SW Interrupt	0	1	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	PIP(NewCR3, NR?)?, MODE.Exec?, TIP.PGE(BLIP)
12c	SW Interrupt	1	0	0		FUP(CLIP), TIP.PGD()
12d	SW Interrupt	1	0	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; *TraceStop if BLIP is in a TraceStop region	FUP(CLIP), PIP(NewCR3, NR?)?, TIP.PGD(BLIP), TraceStop?
12e	SW Interrupt	1	1	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and "Conceal VMX non-root operation from Intel PT" execution control = 0; * MODE.Exec if the operation changes CS.L/D or IA32_EFER.LMA	FUP(CLIP), PIP(NewCR3, NR?)?, FUP(NLIP), MODE.Exec?, TIP(BLIP)
13a	Exception/Fault	0	0	0		None

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
13f	Exception/Fault	0	0	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and “Conceal VMX non-root operation from Intel PT” execution control = 0; *TraceStop if BLIP is in a TraceStop region	PIP(NewCR3, NR?)?, TraceStop?
13b	Exception/Fault	0	1	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and “Conceal VMX non-root operation from Intel PT” execution control = 0; *MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	PIP(NewCR3, NR?)?, MODE.Exec?, TIP.PGE(BLIP)
13c	Exception/Fault	1	0	0		FUP(CLIP), TIP.PGD()
13d	Exception/Fault	1	0	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and “Conceal VMX non-root operation from Intel PT” execution control = 0; *TraceStop if BLIP is in a TraceStop region	FUP(CLIP), PIP(NewCR3, NR?)?, TIP.PGD(BLIP), TraceStop?
13e	Exception/Fault	1	1	1	* PIP if CR3 is updated (i.e., task switch), and OS=1 *PIP.NR=1 if destination is not root operation, and “Conceal VMX non-root operation from Intel PT” execution control = 0; * MODE.Exec if the operation changes CS.L/D or IA32_EFER.LMA	FUP(CLIP), PIP(NewCR3, NR?)?, MODE.Exec?, TIP(BLIP)
14a	SMI (TraceEn cleared)	0	0	D.C.		None
14b	SMI (TraceEn cleared)	1	0	0		FUP(SMRAM,LIP), TIP.PGD()
14f	SMI (TraceEn cleared)	1	0	1		NA
14c	SMI (TraceEn cleared)	1	1	1		NA
15a	RSM, TraceEn restored to 0	0	0	0		None

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
15b	RSM, TraceEn restored to 1	0	0	D.C.		See WRMSR cases for packets on enable
15c	RSM, TraceEn restored to 1	0	1	1		See WRMSR cases for packets on enable. FUP/ TIP.PGE IP is SMRAM.LIP
15e	RSM (TraceEn=1, goes to shutdown)	1	0	0		None
15f	RSM (TraceEn=1, goes to shutdown)	1	0	1		None
15d	RSM (TraceEn=1, goes to shutdown)	1	1	1		None
16i	Vmext	0	0	0		None
16a	Vmext	0	0	1	*PIP if OF=1, and "Conceal VM exits from Intel PT" execution control = 0; *TraceStop if VMCSh.LIP is in a TraceStop region	PIP(HostCR3, NR=0)?, TraceStop?
16b	VM exit, MSR list sets TraceEn=1	0	0	0		See WRMSR cases for packets on enable. FUP IP is VMCSh.LIP
16c	VM exit, MSR list sets TraceEn=1	0	1	1		See WRMSR cases for packets on enable. FUP/ TIP.PGE IP is VMCSh.LIP
16e	VM exit	0	1	1	*PIP if OF=1, and "Conceal VM exits from Intel PT" execution control = 0; *MODE.Exec if the value is different, since last TIP.PGD	PIP(HostCR3, NR=0)?, MODE.Exec?, TIP.PGE(VMCSh.LIP)
16f	VM exit, MSR list clears TraceEn=0	1	0	0	*PIP if OF=1, and "Conceal VM exits from Intel PT" execution control = 0;	FUP(VMCSg.LIP), PIP(HostCR3, NR=0)?, TIP.PGD
16j	VM exit, ContextEN 1->0	1	0	0		FUP(VMCSg.LIP), TIP.PGD
16g	VM exit	1	0	1	*PIP if OF=1, and "Conceal VM exits from Intel PT" execution control = 0; *TraceStop if VMCSh.LIP is in a TraceStop region	FUP(VMCSg.LIP), PIP(HostCR3, NR=0)?, TIP.PGD(VMCSh.LIP), TraceStop?
16h	VM exit	1	1	1	*PIP if OF=1, and "Conceal VM exits from Intel PT" execution control = 0; *MODE.Exec if the value is different, since last TIP.PGD	FUP(VMCSg.LIP), PIP(HostCR3, NR=0)?, MODE.Exec, TIP(VMCSh.LIP)
17a	Vmentry	0	0	0		None

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
17b	VM entry	0	0	1	*PIP if OF=1, and "Conceal VM entries from Intel PT" execution control = 0; *TraceStop if VMCSg.LIP is in a TraceStop region	PIP(GuestCR3, NR=1)?, TraceStop?
17c	VM entry, MSR load list sets TraceEn=1	0	0	1		See WRMSR cases for packets on enable. FUP IP is VMCSg.LIP
17d	VM entry, MSR load list sets TraceEn=1	0	1	1		See WRMSR cases for packets on enable. FUP/ TIP.PGE IP is VMCSg.LIP
17f	VM entry, FilterEN 0->1	0	1	1	*PIP if OF=1, and "Conceal VM entries from Intel PT" execution control = 0; *MODE.Exec if the value is different, since last TIP.PGD	PIP(GuestCR3, NR=1)?, MODE.Exec?, TIP.PGE(VMCSg.LIP)
17j	VM entry, ContextEN 0->1	0	1	1	*MODE.Exec if the value is different, since last TIP.PGD	MODE.Exec, TIP.PGE(VMCSg.LIP)
17g	VM entry, MSR list clears TraceEn=0	1	0	0	*PIP if OF=1, and "Conceal VM entries from Intel PT" execution control = 0;	PIP(GuestCR3, NR=1)?, TIP.PGD
17h	VM entry	1	0	1	*PIP if OF=1, and "Conceal VM entries from Intel PT" execution control = 0; *TraceStop if VMCSg.LIP is in a TraceStop region	PIP(GuestCR3, NR=1)?, TIP.PGD(VMCSg.LIP), TraceStop?
17i	VM entry	1	1	1	*PIP if OF=1, and "Conceal VM entries from Intel PT" execution control = 0; *MODE.Exec if the value is different, since last TIP.PGD	PIP(GuestCR3, NR=1)?, MODE.Exec, TIP(VMCSg.LIP)
20a	EENTER/ERESUME to non-debug enclave	0	0	0		None
20c	EENTER/ERESUME to non-debug enclave	1	0	0		FUP(CLIP), TIP.PGD()
21a	EEXIT from non-debug enclave	0	0	D.C.		None
21b	EEXIT from non-debug enclave	0	1	1	*MODE.Exec if the value is different, since last TIP.PGD	MODE.Exec?, TIP.PGE(BLIP)
22a	AEX/EEE from non-debug enclave	0	0	D.C.		None
22b	AEX/EEE from non-debug enclave	0	1	1	*MODE.Exec if the value is different, since last TIP.PGD	MODE.Exec?, TIP.PGE(AEP.LIP)
23a	EENTER/ERESUME to debug enclave	0	0	D.C.		None
23b	EENTER/ERESUME to debug enclave	0	1	1	*MODE.Exec if the value is different, since last TIP.PGD	MODE.Exec?, TIP.PGE(BLIP)

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
23c	EENTER/ERESUME to debug enclave	1	0	0		FUP(CLIP), TIP.PGD()
23d	EENTER/ERESUME to debug enclave	0	0	1	*TraceStop if BLIP is in a TraceStop region	FUP(CLIP), TIP.PGD(BLIP), TraceStop?
23e	EENTER/ERESUME to debug enclave	1	1	1		FUP(CLIP), TIP(BLIP)
24f	EEXIT from debug enclave	0	0	D.C.		None
24b	EEXIT from debug enclave	0	1	1	*MODE.Exec if the value is different, since last TIP.PGD	MODE.Exec?, TIP.PGE(BLIP)
24d	EEXIT from debug enclave	1	0	1	*TraceStop if BLIP is in a TraceStop region	FUP(CLIP), TIP.PGD(BLIP), TraceStop?
24e	EEXIT from debug enclave	1	1	1		FUP(CLIP), TIP(BLIP)
25a	AEX/EEE from debug enclave	0	0	D.C.		None
25b	AEX/EEE from debug enclave	0	1	1	*MODE.Exec if the value is different, since last TIP.PGD	MODE.Exec?, TIP.PGE(AEP.LIP)
25d	AEX/EEE from debug enclave	1	0	1	*For AEX, FUP IP could be NLIP, for trap-like events	FUP(CLIP), TIP.PGD(AEP.LIP)
25e	AEX/EEE from debug enclave	1	1	1	*MODE.Exec if the value is different, since last TIP.PGD *For AEX, FUP IP could be NLIP, for trap-like events	FUP(CLIP), MODE.Exec?, TIP(AEP.LIP)
26a	XBEGIN/XACQUIRE	0	0	D.C.		None
26d	XBEGIN/XACQUIRE that does not set InTX	1	1	1		None
26e	XBEGIN/XACQUIRE that sets InTX	1	1	1		MODE(InTX=1, TXAbort=0), FUP(CLIP)
27a	XEND/XRELEASE	0	0	D.C.		None
27d	XEND/XRELEASE that does not clear InTX	1	1	1		None
27e	XEND/XRELEASE that clears InTX	1	1	1		MODE(InTX=0, TXAbort=0), FUP(CLIP)
28a	XABORT(Async XAbort, or other)	0	0	0		None
28e	XABORT(Async XAbort, or other)	0	0	1	*TraceStop if BLIP is in a TraceStop region	MODE(InTX=0, TXAbort=1), TraceStop?
28b	XABORT(Async XAbort, or other)	0	1	1		MODE(InTX=0, TXAbort=1), TIP.PGE(BLIP)
28c	XABORT(Async XAbort, or other)	1	0	1	*TraceStop if BLIP is in a TraceStop region	MODE(InTX=0, TXAbort=1), TIP.PGD (BLIP), TraceStop?
28d	XABORT(Async XAbort, or other)	1	1	1		MODE(InTX=0, TXAbort=1), FUP(CLIP), TIP(BLIP)

Table 36-45 Packet Generation under Different Enable Conditions

Case	Operation	PktEn Before	PktEn After	CntxEn After	Other Dependencies	Packets Output
30a	INIT (BSP)	0	0	0		None
30b	INIT (BSP)	0	0	1	*TraceStop if RESET.LIP is in a TraceStop region	BIP(0), TraceStop?
30c	INIT (BSP)	0	1	1	* MODE.Exec if the value is different, since last TIP.PGD	MODE.Exec?, PIP(0), TIP.PGE(ResetLIP)
30d	INIT (BSP)	1	0	0		FUP(NLIP), TIP.PGD()
30e	INIT (BSP)	1	0	1	* PIP if OS=1 *TraceStop if RESET.LIP is in a TraceStop region	FUP(NLIP), PIP(0), TIP.PGD, TraceStop?
30f	INIT (BSP)	1	1	1	* MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB * PIP if OS=1	FUP(NLIP), PIP(0)?, MODE.Exec?, TIP(ResetLIP)
31a	INIT (AP, goes to wait-for-SIPI)	0	D.C.	D.C.		None
31b	INIT (AP, goes to wait-for-SIPI)	1	D.C.	D.C.	* PIP if OS=1	FUP(NLIP), PIP(0)
32a	SIPI	0	0	0		None
32c	SIPI	0	1	1	* MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	MODE.Exec?, TIP.PGE(SIPI-LIP)
32d	SIPI	1	0	0		TIP.PGD
32e	SIPI	1	0	1	*TraceStop if SIPI LIP is in a TraceStop region	TIP.PGD(SIPILIP); TraceS-top?
32f	SIPI	1	1	1	* MODE.Exec if the mode has changed since the last MODE.Exec, or if no MODE.Exec since last PSB	MODE.Exec?, TIP(SIPILIP)
33a	MWAIT (to C0)	D.C.	D.C.	D.C.		None
33b	MWAIT (to higher-numbered C-State, packet sent on wake)	D.C.	D.C.	D.C.	*TSC if TSCEn=1 *TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR

...

29. Updates to Chapter 37, Volume 3D

Change bars show changes to Chapter 37 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

37.3 ENCLAVE LIFE CYCLE

Enclave memory management is divided into two parts: address space allocation and memory commitment. Address space allocation is the specification of the range of logical addresses that the enclave may use. This range is called the ELRANGE. No actual resources are committed to this region. Memory commitment is the assignment of actual memory resources (as pages) within the allocated address space. This two-phase technique allows flexibility for enclaves to control their memory usage and to adjust dynamically without overusing memory resources when enclave needs are low. Commitment adds physical pages to the enclave. An operating system may support separate allocate and commit operations.

During enclave creation, code and data for an enclave are loaded from a clear-text source, i.e. from non-enclave memory.

Untrusted application code starts using an initialized enclave typically by using the EENTER leaf function provided by Intel SGX to transfer control to the enclave code residing in the protected Enclave Page Cache (EPC). The enclave code returns to the caller via the EEXIT leaf function. Upon enclave entry, control is transferred by hardware to software inside the enclave. The software inside the enclave switches the stack pointer to one inside the enclave. When returning back from the enclave, the software swaps back the stack pointer then executes the EEXIT leaf function.

On processors that supports the SGX2 extensions, an enclave writer may add memory to an enclave using the SGX2 instruction set, after the enclave is built and running. These instructions allow adding additional memory resources to the enclave for use in such areas as the heap. In addition, SGX2 instructions allow the enclave to add new threads to the enclave. The SGX2 features provide additional capabilities to the software model without changing the security properties of the Intel SGX architecture.

Calling an external procedure from an enclave could be done using the EEXIT leaf function. Software would use EEXIT and a software convention between the trusted section and the untrusted section.

An active enclave consumes resource from the Enclave Page Cache (EPC, see Section 37.5). Intel SGX provides the EREMOVE instruction that an EPC manager can use to reclaim EPC pages committed to an enclave. The EPC manager uses EREMOVE on every enclave page when the enclave is torn down. After successful execution of EREMOVE the EPC page is available for allocation to another enclave.

37.4 DATA STRUCTURES AND ENCLAVE OPERATION

There are 2 main data structures associated with operating an enclave, the SGX Enclave Control Structure (SECS, see Section 38.7) and the Thread Control Structure (TCS, see Section 38.8).

There is one SECS for each enclave. The SECS contains meta-data about the enclave which is used by the hardware and cannot be directly accessed by software. Included in the SECS is a field that stores the enclave build measurement value. This field, MRENCLAVE, is initialized by the ECREATE instruction and updated by every EADD and EEXTEND. It is locked by EINIT.

Every enclave contains one or more TCS structures. The TCS contains meta-data used by the hardware to save and restore thread specific information when entering/exiting the enclave. There is one field, FLAGS, that may be accessed by software. This field can only be accessed by debug enclaves. The flag bit, DBGOPTIN, allows to single step into the thread associated with the TCS. (see Section 38.8.1)

The SECS is created when ECREATE (see Table 37-1) is executed. The TCS can be created using the EADD instruction or the SGX2 instructions (see Table 37-2).

37.5 ENCLAVE PAGE CACHE

The Enclave Page Cache (EPC) is the secure storage used to store enclave pages when they are a part of an executing enclave. For an EPC page, hardware performs additional access control checks to restrict access to the page. After the current page access checks and translations are performed, the hardware checks that the EPC page is accessible to the program currently executing. Generally an EPC page is only accessed by the owner of the executing enclave or an instruction which is setting up an EPC page

The EPC is divided into EPC pages. An EPC page is 4KB in size and always aligned on a 4KB boundary.

Pages in the EPC can either be valid or invalid. Every valid page in the EPC belongs to one enclave instance. Each enclave instance has an EPC page that holds its SECS. The security metadata for each EPC page is held in an internal micro-architectural structure called Enclave Page Cache Map (EPCM, see Section 37.5.1).

The EPC is managed by privileged software. Intel SGX provides a set of instructions for adding and removing content to and from the EPC. The EPC may be configured by BIOS at boot time. On implementations in which EPC memory is part of system DRAM, the contents of the EPC are protected by an encryption engine.

37.6 ENCLAVE INSTRUCTIONS AND INTEL® SGX

The enclave instructions available with Intel SGX are organized as leaf functions under two instruction mnemonics: ENCLS (ring 0) and ENCLU (ring 3). Each leaf function uses EAX to specify the leaf function index, and may require additional implicit input registers as parameters. The use of EAX is implied implicitly by the ENCLS and ENCLU instructions, ModR/M byte encoding is not used with ENCLS and ENCLU. The use of additional registers does not use ModR/M encoding and is implied implicitly by the respective leaf function index.

Each leaf function index is also associated with a unique, leaf-specific mnemonic. A long-form expression of Intel SGX instruction takes the form of ENCLx[LEAF_MNEMONIC], where ‘x’ is either ‘S’ or ‘U’. The long-form expression provides clear association of the privilege-level requirement of a given “leaf mnemonic”. For simplicity, the unique “Leaf_Mnemonic” name is used (omitting the ENCLx for convenience) throughout in this document.

Details of Individual SGX leaf functions are described in Chapter 41. Table 37-1 provides a summary of the instruction leaves that are available in the initial implementation of Intel SGX, which is introduced in the 6th generation Intel Core processors. Table 37-1 summarizes enhancement of Intel SGX for future Intel processors.

Table 37-1 Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EADD]	Add an EPC page to an enclave.	ENCLU[EENTER]	Enter an enclave.
ENCLS[EBLOCK]	Block an EPC page.	ENCLU[EEXIT]	Exit an enclave.
ENCLS[ECREATE]	Create an enclave.	ENCLU[EGETKEY]	Create a cryptographic key.
ENCLS[EDBGRD]	Read data from a debug enclave by debugger.	ENCLU[EREPORT]	Create a cryptographic report.
ENCLS[EDBGWR]	Write data into a debug enclave by debugger.	ENCLU[ERESUME]	Re-enter an enclave.
ENCLS[EEXTEND]	Extend EPC page measurement.		
ENCLS[EINIT]	Initialize an enclave.		
ENCLS[ELDB]	Load an EPC page in blocked state.		

Table 37-1 Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[ELDU]	Load an EPC page in unblocked state.		
ENCLS[EPA]	Add an EPC page to create a version array.		
ENCLS[EREMOVE]	Remove an EPC page from an enclave.		
ENCLS[ETRACK]	Activate EBLOCK checks.		
ENCLS[EWB]	Write back/invalidate an EPC page.		

...

37.7.2 Intel® SGX Resource Enumeration Leaves

If CPUID.(EAX=07H, ECX=0H):EBX.SGX = 1, the processor also supports querying CPUID with EAX=12H on Intel SGX resource capability and configuration. The number of available sub-leaves in leaf 12H depends on the Opt-in and system software configuration. Information returned by CPUID.12H is thread specific; software should not assume that if Intel SGX instructions are supported on one hardware thread, they are also supported elsewhere.

A properly configured processor exposes Intel SGX functionality with CPUID.EAX=12H reporting valid information (non-zero content) in three or more sub-leaves, see Table 37-4.

- CPUID.(EAX=12H, ECX=0H) enumerates Intel SGX capability, including enclave instruction opcode support.
- CPUID.(EAX=12H, ECX=1H) enumerates Intel SGX capability of processor state configuration and enclave configuration in the SECS structure (see Table 38-3).
- CPUID.(EAX=12H, ECX >1) enumerates available EPC resources.

Table 37-4 CPUID Leaf 12H, Sub-Leaf 0 Enumeration of Intel® SGX Capabilities

CPUID.(EAX=12H,ECX=0)		Description Behavior
Register	Bits	
EAX	0	SGX1: If 1, indicates leaf functions of SGX1 instruction listed in Table 37-1 are supported.
	1	SGX2: If 1, indicates leaf functions of SGX2 instruction listed in Table 37-2 are supported.
	31:2	Reserved (0)
EBX	31:0	MISCSELECT: Reports the bit vector of supported extended features that can be written to the MISC region of the SSA.
ECX	31:0	Reserved (0).
EDX	7:0	MaxEnclaveSize_Not64: the maximum supported enclave size is 2^(EDX[7:0]) bytes when not in 64-bit mode.
	15:8	MaxEnclaveSize_64: the maximum supported enclave size is 2^(EDX[15:8]) bytes when operating in 64-bit mode.
	31:16	Reserved (0).

Table 37-5 CPUID Leaf 12H, Sub-Leaf 1 Enumeration of Intel® SGX Capabilities

CPUID.(EAX=12H,ECX=1)		Description Behavior
Register	Bits	
EAX	31:0	Report the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE. SECS.ATTRIBUTES[n] can be set to 1 using ECREATE only if EAX[n] is 1, where n < 32.
EBX	31:0	Report the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE. SECS.ATTRIBUTES[n+32] can be set to 1 using ECREATE only if EBX[n] is 1, where n < 32.
ECX	31:0	Report the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE. SECS.ATTRIBUTES[n+64] can be set to 1 using ECREATE only if ECX[n] is 1, where n < 32.
EDX	31:0	Report the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE. SECS.ATTRIBUTES[n+96] can be set to 1 using ECREATE only if EDX[n] is 1, where n < 32.

On processors that support Intel SGX1 and SGX2, CPUID leaf 12H sub-leaf 2 report physical memory resources available for use with Intel SGX. These physical memory sections are typically allocated by BIOS as **Processor Reserved Memory**, and available to the OS to manage as EPC.

To enumerate how many EPC sections are available to the EPC manager, software can enumerate CPUID leaf 12H with sub-leaf index starting from 2, and decode the sub-leaf-type encoding (returned in EAX[3:0]) until the sub-leaf type is invalid. All invalid sub-leaves of CPUID leaf 12H return EAX/EBX/ECX/EDX with 0.

Table 37-6 CPUID Leaf 12H, Sub-Leaf Index 2 or Higher Enumeration of Intel® SGX Resources

CPUID.(EAX=12H,ECX > 1)		Description Behavior
Register	Bits	
EAX	3:0	0000b: This sub-leaf is invalid; EDX:ECX:EBX:EAX return 0. 0001b: This sub-leaf enumerates an EPC section. EBX:EAX and EDX:ECX provide information on the Enclave Page Cache (EPC) section. All other encoding are reserved.
	11:4	Reserved (enumerate 0).
	31:12	If EAX[3:0] = 0001b, these are bits 31:12 of the physical address of the base of the EPC section.
EBX	19:0	If EAX[3:0] = 0001b, these are bits 51:32 of the physical address of the base of the EPC section.
	31:20	Reserved.
ECX	3: 0	If EAX[3:0] 0000b, then all bits of the EDX:ECX pair are enumerated as 0. If EAX[3:0] 0001b, then this section has confidentiality and integrity protection. All other encoding are reserved.
	11:4	Reserved (enumerate 0).
	31:12	If EAX[3:0] = 0001b, these are bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory.
EDX	19: 0	If EAX[3:0] = 0001b, these are bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory.
	31:20	Reserved.

...

30. Updates to Chapter 38, Volume 3D

Change bars show changes to Chapter 38 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

38.1 OVERVIEW OF ENCLAVE EXECUTION ENVIRONMENT

When an enclave is created, it has a range of linear addresses that the processor applies enhanced access control. This ranged is called the ELRANGE (see Section 37.3). 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).

Attempt to execute, read or write to linear addresses mapped to EPC pages when not inside an enclave will result in undefined behavior. The processor will provide the protections as described in Section 38.4 and Section 38.5 on such accesses.

...

38.3 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.
- Non-enclave accesses to EPC memory result in undefined behavior. EPC memory is protected as described in Section 38.4 and Section 38.5 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.
  - — Target page must not have a restricted page type[1] (PT_SECS, PT_TCS, PT_VA, or PT_TRIM).

— The EPC page must not be BLOCKED.

— The EPC page must not be PENDING.

— The EPC page must not be MODIFIED.

...

38.5 PAGE-BASED ACCESS CONTROL

38.5.1 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 only accessible 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 38.3. or through certain Intel SGX instructions. Attempts to execute, read or write to linear addresses mapped to EPC pages using non-enclave access results in undefined behavior.

...

38.5.3.2 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 38-1 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 38-1 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 40.

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 38-1 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

[1] EPCM may allow write, read or execute access only for pages with page type PT_REG.

Table 38-1 List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions

Instr. Leaf	Enum.	Explicit 1	Explicit 2	Explicit 3	Implicit
EBLOCK	SGX1	EPCPAGE			SECS
ECREATE	SGX1	PAGEINFO and linked structures	EPCPAGE		
EDBGRD	SGX1	EPCADDR	Destination		SECS
EDBGWR	SGX1	EPCADDR	Source		SECS
EENTER	SGX1	TCS and linked SSA			SECS
EEXIT	SGX1				SECS, TCS
EEXTEND	SGX1	SECS	EPCPAGE		
EGETKEY	SGX1	KEYREQUEST	KEY		SECS
EINIT	SGX1	SIGSTRUCT	SECS	EINITTOKEN	
ELDB/ELDU	SGX1	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	
EMODPE	SGX2	SECINFO	EPCPAGE		
EMODPR	SGX2	SECINFO	EPCPAGE		SECS
EMODT	SGX2	SECINFO	EPCPAGE		SECS
EPA	SGX1	EPCADDR			
EREMOVE	SGX1	EPCPAGE			SECS
EREPORT	SGX1	TARGETINFO	REPORTDATA	OUTPUTDATA	SECS
ERESUME	SGX1	TCS and linked SSA			SECS
ETRACK	SGX1	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 40.4					

...

38.7 SGX ENCLAVE CONTROL STRUCTURE (SECS)

The SECS data structure requires 4K-Bytes alignment.

Table 38-2 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 38.7.2) of the SSA frame when an AEX occurs.
RESERVED	24	24	
ATTRIBUTES	48	16	Attributes of the Enclave, see Table 38-3.

Table 38-2 Layout of SGX Enclave Control Structure (SECS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
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	96	
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
RESERVED	260	3836	The RESERVED field consists of the following: ▪ EID: An 8 byte Enclave Identifier. Its location is implementation specific. ▪ PAD: A 352 bytes padding pattern from the Signature (used for key derivation strings). It's location is implementation specific. ▪ The remaining 3476 bytes are reserved area. The entire 3836 byte field must be cleared prior to executing ECREATE or EREPORT.

38.7.1 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 38-3 Layout of ATTRIBUTES Structure

Field	Bit Position	Description
INIT	0	This bit indicates if the enclave has been initialized by EINIT. It must be cleared when loaded as part of ECREATE. For EREPORT instruction, TARGET_INFO.ATTRIBUTES[ENIT] must always be 1 to match the state after EINIT has initialized the enclave.
DEBUG	1	If 1, the enclave permit debugger to read and write enclave data using EDBGRD and EDBGWR.
MODE64BIT	2	Enclave runs in 64-bit mode.
RESERVED	3	Must be Zero.
PROVISIONKEY	4	Provisioning Key is available from EGETKEY.
EINITTOKENKEY	5	EINIT token key is available from EGETKEY.
RESERVED	63:6	
XFRM	127:64	XSAVE Feature Request Mask. See Section 42.7.

38.7.2 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 38-4.

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 38.9.2.

Table 38-4 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.
Reserved	31:1	Reserved (0).

38.8 THREAD CONTROL STRUCTURE (TCS)

Each executing thread in the enclave is associated with a Thread Control Structure. It requires 4K-Bytes alignment.

Table 38-5 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 processer is currently executing an enclave in the context of this TCS.
FLAGS	8	8	The thread's execution flags (see Section 38.8.1).
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.
OFSBASGX	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.
OGSBASGX	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.
RESERVED	72	4024	Must be zero.

...

38.8.2 State Save Area Offset (OSSA)

The OSSA points to a stack of State Save Area (SSA) frames (see Section 38.9) used to save the processor state when an interrupt or exception occurs while executing in the enclave.

...

38.9 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 38-7). 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 38.9.2.

Table 38-7 Top-to-Bottom Layout of an SSA Frame

Region	Offset (Byte)	Size (Bytes)	Description
XSAVE	0	Calculate using CPUID leaf 0DH information	The size of XSAVE region in SSA is derived from the enclave's support of the collection of processor extended states that would be managed by XSAVE. The enablement of those processor extended state components in conjunction with CPUID leaf 0DH information determines the XSAVE region size in SSA.
Pad	End of XSAVE region	Chosen by enclave writer	Ensure the end of GPRSGX region is aligned to the end of a 4KB page.
MISC	base of GPRSGX -sizeof(MISC)	Calculate from highest set bit of SECS.MISCSELECT	See Section 38.9.2.
GPRSGX	SSAFRAMESIZE -176	176	See Table 38-8 for layout of the GPRSGX region.

...

38.9.2 MISC Region

The layout of the MISC region is shown in Table 38-11. The number of components that the processor supports in the MISC region corresponds to the set 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 38-11. 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 38-11. 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 38-11 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.
Future Extension	Below EXINFO	TBD	Reserved. (Zero size if CPUID.(EAX=12H, ECX=0):EBX[31:1] =0).

...

38.9.2.2 Page Fault Error Codes

Table 38-13 contains page fault error code that may be reported in EXINFO.ERRCD.

Table 38-13 Page Fault Error Codes

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

...

38.11.2 PAGE_TYPE Field Definition

The SECINFO flags and EPC flags contain bits indicating the type of page.

Table 38.17 Supported PAGE_TYPE

TYPE	Value	Description
PT_SECS	0	Page is an SECS.
PT_TCS	1	Page is a TCS.

Table 38.17   Supported PAGE_TYPE

TYPE	Value	Description
PT_REG	2	Page is a regular page.
PT_VA	3	Page is a Version Array.
PT_TRIM	4	Page is in trimmed state.
	All other	Reserved.

38.12 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 38-18   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.

38.13 ENCLAVE SIGNATURE STRUCTURE (SIGSTRUCT)

SIGSTRUCT is a structure created and signed by the enclave developer that contains information about the enclave. SIGSTRUCT is processed by the EINIT leaf function to verify that the enclave was properly built.

SIGSTRUCT includes ENCLAVEHASH as SHA256 digest, as defined in FIPS PUB 180-4. The digests are byte strings of length 32. Each of the 8 HASH dwords is stored in little-endian order.

SIGSTRUCT includes four 3072-bit integers (MODULUS, SIGNATURE, Q1, Q2). Each such integer is represented as a byte strings of length 384, with the most significant byte at the position “offset + 383”, and the least significant byte at position “offset”.

The (3072-bit integer) SIGNATURE should be an RSA signature, where: a) the RSA modulus (MODULUS) is a 3072-bit integer; b) the public exponent is set to 3; c) the signing procedure uses the EMSA-PKCS1-v1.5 format with DER encoding of the “DigestInfo” value as specified in of PKCS#1 v2.1/RFC 3447.

The 3072-bit integers Q1 and Q2 are defined by:

q1 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

SIGSTRUCT must be page aligned

In column 5 of Table 38-19, ‘Y’ indicates that this field should be included in the signature generated by the developer.

...

38.14 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 38-20 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.
RESERVED	212	24	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 EINITOKENKEY.

38.15 REPORT (REPORT)

The REPORT structure is the output of the EREPORT instruction, and must be 512-Byte aligned.

Table 38-21 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.
RESERVED	20	28	Must be zero.
ATTRIBUTES	48	16	ATTRIBUTES of the Enclave. See Section 38.7.1.
MRENCLAVE	64	32	The value of SECS.MRENCLAVE.

Table 38-21 Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
RESERVED	96	32	Reserved.
MRSIGNER	128	32	The value of SECS.MRSIGNER.
RESERVED	160	96	Zero.
ISVPRODID	256	02	Product ID of enclave.
ISVSVN	258	02	Security version number (SVN) of the enclave.
RESERVED	260	60	Zero.
REPORTDATA	320	64	Data provided by the user and protected by the REPORT's MAC, see Section 38.15.1.
KEYID	384	32	Value for key wear-out protection.
MAC	416	16	Message Authentication Code on the report using report key.

38.15.1 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.

...

38.17 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.

...

38.17.1 KEY REQUEST KeyNames

Table 38-24 Supported KEYName Values

Key Name	Value	Description
EINITOKEN_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 other	Reserved

...

38.19 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 38-27 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).
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.

1. The application hands over the enclave content along with additional information required by the enclave creation API to the enclave creation service running at privilege level 0.
2. The enclave creation service running at privilege level 0 uses the ECREATE leaf function to set up the initial environment, specifying base address and size of the enclave. This address range, the ELRANGE, is part of the application's address space. This reserves the memory range. The enclave will now reside in this address region. ECREATE also allocates an Enclave Page Cache (EPC) page for the SGX Enclave Control Structure (SECS). Note that this page is not required to be a part of the enclave linear address space and is not required to be mapped into the process.
3. The enclave creation service uses the EADD leaf function to commit EPC pages to the enclave, and use EEXTEND to measure the committed memory content of the enclave. For each page to be added to the enclave:
   - Use EADD to add the new page to the enclave.
   - If the enclave developer requires measurement of the page as a proof for the content, use EEXTEND to add a measurement for 256 bytes of the page. Repeat this operation until the entire page is measured.
4. The enclave creation service uses the EINIT leaf function to complete the enclave creation process and finalize the enclave measurement to establish the enclave identity. Until an EINIT is executed, the enclave is not permitted to execute any enclave code (i.e. entering the enclave by executing EENTER would result in a fault).

...

31. Updates to Chapter 39, Volume 3D

Change bars show changes to Chapter 39 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

The following aspects of enclave operation are described in this chapter:

- Enclave creation: Includes loading code and data from outside of enclave into the EPC and establishing the enclave entity.
- Adding pages and measuring the enclave.
- Initialization of an enclave: Finalizes the cryptographic log and establishes the enclave identity and sealing identity.
- Enclave entry and exiting including:
  - — Controlled entry and exit.
  - — Asynchronous Enclave Exit (AEX) and resuming execution after an AEX.

39.1 CONSTRUCTING AN ENCLAVE

Figure 39-1 illustrates a typical Enclave memory layout.

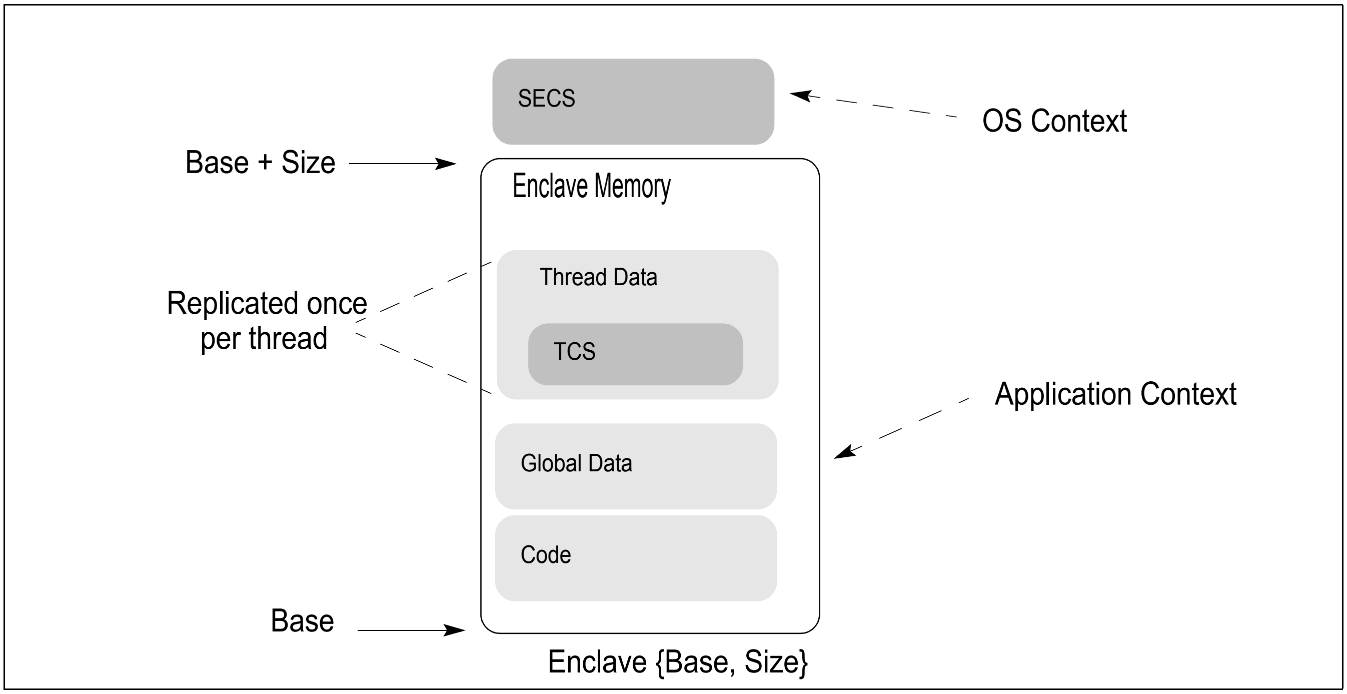


Figure 39-1 Enclave Memory Layout

The enclave creation, commitment of memory resources, and finalizing the enclave's identity with measurement comprises multiple phases. This process can be illustrated by the following exemplary steps:

...

39.1.2 EADD and EEXTEND Interaction

Once the SECS has been created, enclave pages can be added to the enclave via EADD. This involves converting a free EPC page into either a PT_REG or a PT_TCS page.

When EADD is invoked, the processor will update the EPCM entry with the type of page (PT_REG or PT_TCS), the linear address used by the enclave to access the page, and the enclave access permissions for the page. It associates the page to the SECS provided as input. The EPCM entry information is used by hardware to manage access control to the page. EADD records EPCM information in the cryptographic log stored in the SECS and copies 4 KBytes of data from unprotected memory outside the EPC to the allocated EPC page.

System software is responsible for selecting a free EPC page. System software is also responsible for providing the type of page to be added, the attributes the page, the contents of the page, and the SECS (enclave) to which the page is to be added as requested by the application. Incorrect data would lead to a failure of EADD or to an incorrect cryptographic log and a failure at EINIT time.

After a page has been added to an enclave, software can measure a 256 byte region as determined by the developer by invoking EEXTEND. Thus to measure an entire 4KB page, system software must execute EEXTEND 16 times. Each invocation of EEXTEND adds to the cryptographic log information about which region is being measured and the measurement of the section.

Entries in the cryptographic log define the measurement of the enclave and are critical in gaining assurance that the enclave was correctly constructed by the untrusted system software.

39.1.3 EINIT Interaction

Once system software has completed the process of adding and measuring pages, the enclave needs to be initialized by the EINIT leaf function. After an enclave is initialized, EADD and EEXTEND are disabled for that enclave (An attempt to execute EADD/EEXTEND to enclave after enclave initialization will result in a fault). The initialization process finalizes the cryptographic log and establishes the **enclave identity** and **sealing identity** used by EGETKEY and EREPORT.

A cryptographic hash of the log is stored as the **enclave identity**. Correct construction of the enclave results in the cryptographic hash matching the one built by the enclave owner and included as the ENCLAVEHASH field of SIGSTRUCT. The **enclave identity** provided by the EREPORT leaf function can be verified by a remote party.

The EINIT leaf function checks the EINIT token to validate that the enclave has been enabled on this platform. If the enclave is not correctly constructed, or the EINIT token is not valid for the platform, or SIGSTRUCT isn't properly signed, then EINIT will fail. See the EINIT leaf function for details on the error reporting.

The **enclave identity** is a cryptographic hash that reflects the enclave attributes and MISCSELECT value, content of the enclave, the order in which it was built, the addresses it occupies in memory, the security attributes, and access right permissions of each page. The **enclave identity** is established by the EINIT leaf function.

The **sealing identity** is managed by a sealing authority represented by the hash of the public key used to sign the SIGSTRUCT structure processed by EINIT. The sealing authority assigns a product ID (ISVPRODID) and security version number (ISVSVN) to a particular enclave identity.

EINIT establishes the sealing identity using the following steps:

1. Verifies that SIGSTRUCT is properly signed using the public key enclosed in the SIGSTRUCT.

2. Checks that the measurement of the enclave matches the measurement of the enclave specified in SIGSTRUCT.

3. Checks that the enclave’s attributes and MISCSELECT values are compatible with those specified in SIGSTRUCT.

4. Finalizes the measurement of the enclave and records the **sealing identity** (the sealing authority, product id and security version number) and **enclave identity** in the SECS.

5. Sets the ATTRIBUTES.INIT bit for the enclave.

39.1.4 Intel® SGX Launch Control Configuration

Intel® SGX Launch Control is a set of controls that govern the creation of enclaves. Before the EINIT leaf function will successfully initialize an enclave, a designated Launch Enclave must create an EINITTOKEN for that enclave. Launch Enclaves have SECS.ATTRIBUTES.EINITTOKENKEY = 1, granting them access to the EINITTOKENKEY from the EGETKEY leaf function. EINITTOKENKEY must be used by the Launch Enclave when computing EINITTOKEN.MAC, the Message Authentication Code of the EINITTOKEN.

The hash of the public key used to sign the SIGSTRUCT of the Launch Enclave must equal the value in the IA32_SGXLEPUBKEYHASH MSRs. Only Launch Enclaves are allowed to launch without a valid token.

The IA32_SGXLEPUBKEYHASH MSRs are provided to designate the platform's Launch Enclave. IA32_SGXLEPUBKEYHASH defaults to digest of Intel's launch enclave signing key after reset.

IA32_FEATURE_CONTROL bit 17 controls the permissions on the IA32_SGXLEPUBKEYHASH MSRs when CPUID.(EAX=12H, ECX=00H):EAX[0] = 1. If IA32_FEATURE_CONTROL is locked with bit 17 set, IA32_SGXLEPUBKEYHASH MSRs are reconfigurable (writeable). If either IA32_FEATURE_CONTROL is not locked or bit 17 is clear, the MSRs are read only. By leaving these MSRs writable, system SW or a VMM can support a plurality of Launch Enclaves for hosting multiple execution environments. See Section 42.3.2 for more details.

39.2 ENCLAVE ENTRY AND EXITING

39.2.1 Controlled Entry and Exit

The EENTER leaf function is the method to enter the enclave under program control. To execute EENTER, software must supply an address of a TCS that is part of the enclave to be entered. The TCS holds the location inside the enclave to transfer control to and a pointer to the SSA frame inside the enclave that an AEX should store the register state to.

When a logical processor enters an enclave, the TCS is considered busy until the logical processors exits the enclave. An attempt to enter an enclave through a busy TCS results in a fault. Intel® SGX allows an enclave builder to define multiple TCSs, thereby providing support for multithreaded enclaves.

Software must also supply to EENTER the Asynchronous Exit Pointer (AEP) parameter. AEP is an address external to the enclave which an exception handler will return to using IRET. Typically the location would contain the ERESUME instruction. ERESUME transfers control back to the enclave, to the address retrieved from the enclave thread's saved state.

EENTER performs the following operations:

1. Check that TCS is not busy and flush all cached linear-to-physical mappings.
2. Change the mode of operation to be in enclave mode.
3. Save the old RSP, RBP for later restore on AEX (Software is responsible for setting up the new RSP, RBP to be used inside enclave).
4. Save XCR0 and replace it with the XFRM value for the enclave.
5. Check if software wishes to debug (applicable to a debuggable enclave):
   - — If not debugging, then configure hardware so the enclave appears as a single instruction.
   - — If debugging, then configure hardware to allow traps, breakpoints, and single steps inside the enclave.
6. Set the TCS as busy.
7. Transfer control from outside enclave to predetermined location inside the enclave specified by the TCS.

The EEXIT leaf function is the method of leaving the enclave under program control. EEXIT receives the target address outside of the enclave that the enclave wishes to transfer control to. It is the responsibility of enclave software to erase any secret from the registers prior to invoking EEXIT. To allow enclave software to easily perform an external function call and re-enter the enclave (using EEXIT and EENTER leaf functions), EEXIT returns the value of the AEP that was used when the enclave was entered.

EEXIT performs the following operations:

1. Clear enclave mode and flush all cached linear-to-physical mappings.
2. Mark TCS as not busy.
3. Transfer control from inside the enclave to a location on the outside specified as parameter to the EEXIT leaf function.

...

39.2.3.1 ERESUME Interaction

ERESUME restores registers depending on the mode of the enclave (32 or 64 bit).

- In 32-bit mode (IA32_EFER.LMA = 0 || CS.L = 0), the low 32-bits of the legacy registers (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP and EFLAGS) are restored from the thread’s GPR area of the current SSA frame. Neither the upper 32 bits of the legacy registers nor the 64-bit registers (R8 … R15) are loaded.
- In 64-bit mode (IA32_EFER.LMA = 1 && CS.L = 1), all 64 bits of the general processor registers (RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8 … R15, RIP and RFLAGS) are loaded.

Extended features specified by SECS.ATTRIBUTES.XFRM are restored from the XSAVE area of the current SSA frame. The layout of the x87 area depends on the current values of IA32_EFER.LMA and CS.L:

- IA32_EFER.LMA = 0 || CS.L = 0
    - 32-bit load in the same format that XSAVE/FXSAVE uses with these values.
- IA32_EFER.LMA = 1 && CS.L = 1
    - 64-bit load in the same format that XSAVE/FXSAVE uses with these values as if REX.W = 1.

...

39.3.2 Register Preservation

As with most systems, it is the responsibility of the callee to preserve all registers except that used for returning a value. This is consistent with conventional usage and tends to optimize the number of register save/restore operations that need be performed. It has the additional security result that it ensures that data is scrubbed from any registers that were used by enclave to temporarily contain secrets.

...

39.4.2.1 Enclave Security Version

In the SIGSTRUCT, the MRSIGNER is associated with a 16-bit Product ID (ISVPRODID) and a 16 bit integer SVN (ISVSVN). Together they define a specific group of versions of a specific product. Most keys, including the Seal Key, can be bound to this pair.

To support upgrading from one release to another, EGETKEY will return keys corresponding to any value less than or equal to the software's ISVSVN.

39.4.2.2 Hardware Security Version

CPUSVN is a 128 bit value that reflects the microcode update version and authenticated code modules supported by the processor. Unlike ISVSVN, CPUSVN is not an integer and cannot be compared mathematically. Not all values are valid CPUSVNs.

Software must ensure that the CPUSVN provided to EGETKEY is valid. EREPORT will return the CPUSVN of the current environment. Software can execute EREPORT with TARGETINFO set to zeros to retrieve a CPUSVN from REPORTDATA. Software can access keys for a CPUSVN recorded previously, provided that each of the elements reflected in CPUSVN are the same or have been upgraded.

39.4.3 Keys

Intel® SGX provides software with access to keys unique to each processor and rooted in HW keys inserted into the processor during manufacturing.

Each enclave requests keys using the EGETKEY leaf function. The key is based on enclave parameters such as measurement, the enclave signing key, security attributes of the enclave, and the Hardware Security version of the processor itself. A full list of parameter options is specified in the KEYREQUEST structure, see details in Section 38.17.

By deriving keys using enclave properties, SGX guarantees that if two enclaves call EGETKEY, they will receive a unique key only accessible by the respective enclave. It also guarantees that the enclave will receive the same key on every future execution of EGETKEY. Some parameters are optional or configurable by software. For example, a Seal key can be based on the signer of the enclave, resulting in a key available to multiple enclaves signed by the same party.

The EGETKEY leaf function provides several key types. Each key is specific to the processor, CPUSVN, and the enclave that executed EGETKEY. The EGETKEY instruction definition details how each of these keys is derived, see Table 41-56. Additionally,

- SEAL Key: The Seal key is a general purpose key for the enclave to use to protect secrets. Typical uses of the Seal key are encrypting and calculating MAC of secrets on disk. There are 2 types of Seal Key described in Section 39.4.3.1.
- REPORT Key: This key is used to compute the MAC on the REPORT structure. The EREPORT leaf function is used to compute this MAC, and destination enclave uses the Report key to verify the MAC. The software usage flow is detailed in Section 39.4.3.2.
- EINITOKENKEY: This key is used by Launch Enclaves to compute the MAC on EINITTOKENs. These tokens are then verified in the EINIT leaf function. The key is only available to enclaves with ATTRIBUTE.EINITTOKENKEY set to 1.
- PROVISIONING Key and PROVISIONING SEAL Key: These keys are used by attestation key provisioning software to prove to remote parties that the processor is genuine and identify the currently executing TCB. These keys are only available to enclaves with ATTRIBUTE.PROVISIONKEY set to 1.

...

39.4.3.2 Using REPORTs for Local Attestation

SGX provides a means for enclaves to securely identify one another, this is referred to as “Local Attestation”. SGX provides a hardware assertion, REPORT that contains calling enclaves Attributes, Measurements and User supplied data (described in detail in Section 38.15). Figure 39-3 shows the basic flow of information.

1. The source enclave determines the identity of the target enclave to populate TARGETINFO.
2. The source enclave calls EREPORT instruction to generate a REPORT structure. The EREPORT instruction conducts the following:
   — Populates the REPORT with identify information about the calling enclave.

— Derives the Report Key that is returned when the target enclave executes the EGETKEY. TARGETINFO provides information about the target.

— Computes a MAC over the REPORT using derived target enclave Report Key.

3. Non-enclave software copies the REPORT from source to destination.
4. The target enclave executes the EGETKEY instruction to request its REPORT key, which is the same key used by EREPORT at the source.
5. The target enclave verifies the MAC and can then inspect the REPORT to identify the source.

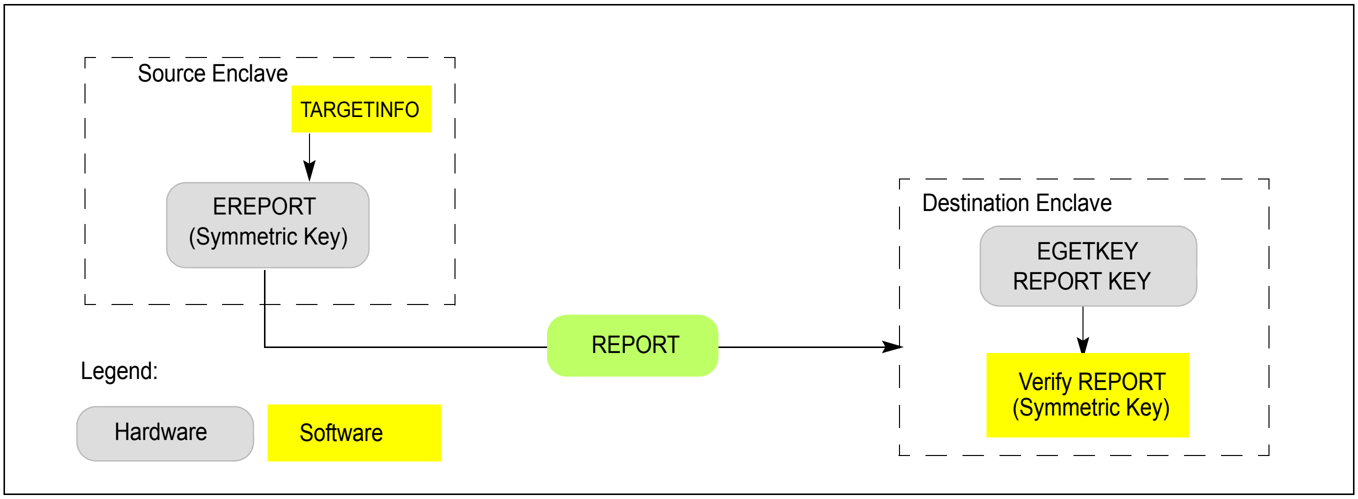


Figure 39-3 SGX Local Attestation

39.5 EPC AND MANAGEMENT OF EPC PAGES

EPC layout is implementation specific, and is enumerated through CPUID (see Table 37-6 for EPC layout). EPC is typically configured by BIOS at system boot time.

39.5.1 EPC Implementation

EPC must be properly protected against attacks. One example of EPC implementation could use a Memory Encryption Engine (MEE). An MEE provides a cost-effective mechanism of creating cryptographically protected volatile storage using platform DRAM. These units provide integrity, replay, and confidentiality protection. Details are implementation specific.

39.5.2 OS Management of EPC Pages

The EPC is a finite resource. SGX1 (i.e. CPUID.(EAX=12H, ECX=0):EAX.SGX1 = 1 but CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 0) provides the EPC manager with leaf functions to manage this resource and properly swap pages out of and into the EPC. For that, the EPC manager would need to keep track of all EPC entries, type and state, context affiliation, and SECS affiliation.

Enclave pages that are candidates for eviction should be moved to BLOCKED state using EBLOCK instruction that ensures no new cached virtual to physical address mappings can be created by attempts to reference a BLOCKED page.

Before evicting blocked pages, EPC manager should execute ETRACK leaf function on that enclave and ensure that there are no stale cached virtual to physical address mappings for the blocked pages remain on any thread on the platform.

After removing all stale translations from blocked pages, system software should use the EWB leaf function for securely evicting pages out of the EPC. EWB encrypts a page in the EPC, writes it to unprotected memory, and invalidates the copy in EPC. In addition, EWB also creates a cryptographic MAC (PCMD.MAC) of the page and stores it in unprotected memory. A page can be reloaded back to the processor only if the data and MAC match. To ensure that the only latest version of the evicted page can be loaded back, the version of the evicted page is stored securely in a Version Array (VA) in EPC.

SGX1 includes two instructions for reloading pages that have been evicted by system software: ELDU and ELDB. The difference between the two instructions is the value of the paging state at the end of the instruction. ELDU results in a page being reloaded and set to an UNBLOCKED state, while ELDB results in a page loaded to a BLOCKED state.

ELDB is intended for use by a Virtual Machine Monitor (VMM). When a VMM reloads an evicted page, it needs to restore it to the correct state of the page (BLOCKED vs. UNBLOCKED) as it existed at the time the page was evicted. Based on the state of the page at eviction, the VMM chooses either ELDB or ELDU.

39.5.3 Eviction of Enclave Pages

Intel SGX paging is optimized to allow the Operating System (OS) to evict multiple pages out of the EPC under a single synchronization.

The suggested flow for evicting a list of pages from the EPC is:

1. For each page to be evicted from the EPC:
   a. Select an empty slot in a Version Array (VA) page.
      - If no empty VA page slots exist, create a new VA page using the EPA leaf function.

   b. Remove linear-address to physical-address mapping from the enclave contexts's mapping tables (page table and EPT tables).

   c. Execute the EBLOCK leaf function for the target page. This sets the target page state to BLOCKED. At this point no new mappings of the page will be created. So any access which does not have the mapping cached in the TLB will generate a #PF.
2. For each enclave containing pages selected in step 1:
   — Execute an ETRACK leaf function pointing to that enclave's SECS. This initiates the tracking process that ensures that all caching of linear-address to physical-address translations for the blocked pages is cleared.
3. For all logical processors executing in processes (OS) or guests (VMM) that contain the enclaves selected in step 1:
   — Issue an IPI (inter-processor interrupt) to those threads. This causes those logical processors to asynchronously exit any enclaves they might be in, and as a result flush cached linear-address to physical-address translations that might hold stale translations to blocked pages. There is no need for additional measures such as performing a "TLB shootdown".
4. After enclaves exit, allow logical processors can resume normal operation, including enclave re-entry as the tracking logic keeps track of the activity.
5. For each page to be evicted:
   — Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents, and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

At this point, system software has the only copy of each page data encrypted with its page metadata in main memory.

39.5.4 Loading an Enclave Page

To reload a previously evicted page, system software needs four elements: the VA slot used when the page was evicted, a buffer containing the encrypted page contents, a buffer containing the page metadata, and the parent SECS to associate this page with. If the VA page or the parent SECS are not already in the EPC, they must be reloaded first.

1. Execute ELDB/ELDU (depending on the desired BLOCKED state for the page), passing as parameters: the EPC page linear address, the VA slot, the encrypted page, and the page metadata.
2. Create a mapping in the enclave context's mapping tables (page tables and EPT tables) to allow the application to access that page (OS: system page table; VMM: EPT).

The ELDB/ELDU instruction marks the VA slot empty so that the page cannot be replayed at a later date.

...

39.5.7 Allocating a Regular Page

On processors that support SGX2, allocating a new page to an already initialized enclave is accomplished by invoking the EAUG leaf function. Typically, the enclave requests that the OS allocate a new page at a particular location within the enclave's address space. Once allocated, the page remains in a pending state until the enclave executes the corresponding EACCEPT leaf function to accept the new page into the enclave. Page allocation operations may be batched to improve efficiency.

The typical process for allocating a regular page is as follows:

1. Enclave requests additional memory from OS when the current allocation becomes insufficient.
2. The OS invokes the EAUG leaf function to add a new memory page to the enclave.
   a. EAUG may only be called on a free EPC page.
   b. Successful completion of the EAUG instruction places the target page in the VALID and PENDING state.
   c. All dynamically created pages have the type PT_REG and content of all zeros.
3. The OS maps the page in the enclave context's mapping tables.
4. The enclave issues an EACCEPT instruction, which verifies the page's attributes and clears the PENDING state. At that point the page becomes accessible for normal enclave use.

39.5.8 Allocating a TCS Page

On processors that support SGX2, allocating a new TCS page to an already initialized enclave is a two-step process. First the OS allocates a regular page with a call to EAUG. This page must then be accepted and initialized by the enclave to which it belongs. Once the page has been initialized with appropriate values for a TCS page, the enclave requests the OS to change the page's type to PT_TCS. This change must also be accepted. As with allocating a regular page, TCS allocation operations may be batched.

A typical process for allocating a TCS page is as follows:

1. Enclave requests an additional page from the OS.
2. The OS invokes EAUG to add a new regular memory page to the enclave.
   a. EAUG may only be called on a free EPC page.

   b. Successful completion of the EAUG instruction places the target page in the VALID and PENDING state.
3. The OS maps the page in the enclave context's mapping tables.
4. The enclave issues an EACCEPT instruction, at which point the page becomes accessible for normal enclave use.
5. The enclave initializes the contents of the new page.
6. The enclave requests that the OS convert the page from type PT_REG to PT_TCS.
7. OS issues an EMODT instruction on the page.
   a. The parameters to EMODT indicate that the regular page should be converted into a TCS.
   b. EMODT forces all access rights to a page to be removed because TCS pages may not be accessed by enclave code.
8. The enclave issues an EACCEPT instruction to confirm the requested modification.

39.5.9 Trimming a Page

On processors that support SGX2, Intel SGX supports the trimming of an enclave page as a special case of EMODT. Trimming allows an enclave to actively participate in the process of removing a page from the enclave (deallocation) by splitting the process into first removing it from the enclave's access and then removing it from the EPC using the EREMOVE leaf function. The page type PT_TRIM indicates that a page has been trimmed from the enclave’s address space and that the page is no longer accessible to enclave software. Modifications to a page in the PT_TRIM state are not permitted; the page must be removed and then reallocated by the OS before the enclave may use the page again. Page deallocation operations may be batched to improve efficiency.

The typical process for trimming a page from an enclave is as follows:

1. Enclave signals OS that a particular page is no longer in use.
2. OS invokes the EMODT leaf function on the page, requesting that the page’s type be changed to PT_TRIM.
   a. SECS and VA pages cannot be trimmed in this way, so the initial type of the page must be PT_REG or PT_TCS.
   b. EMODT may only be called on valid enclave pages.
3. OS invokes the ETRACK leaf function on the enclave containing the page to track removal the TLB addresses from all the processors.
4. Issue an IPI (inter-processor interrupt) to flush the stale linear-address to physical-address translations for all logical processors executing in processes that contain the enclave.
5. Enclave issues an EACCEPT leaf function.
6. The OS may now permanently remove the page from the EPC (by issuing EREMOVE).

39.5.10 Restricting the EPCM Permissions of a Page

On processors that support SGX2, restricting the EPCM permissions associated with an enclave page is accomplished using the EMODPR leaf function. This operation requires the cooperation of the OS to flush stale entries to the page and to update the page-table permissions of the page to match. Permissions restriction operations may be batched.

The typical process for restricting the permissions of an enclave page is as follows:

1. Enclave requests that the OS to restrict the permissions of an EPC page.
2. OS performs permission restriction, flushing cached linear-address to physical-address translations, and page-table modifications.

   a. Invokes the EMODPR leaf function to restrict permissions (EMODPR may only be called on VALID pages).
   b. Invokes the ETRACK leaf function on the enclave containing the page to track removal of the TLB addresses from all the processor.
   c. Issue an IPI (inter-processor interrupt) to flush the stale linear-address to physical-address translations for all logical processors executing in processes that contain the enclave.
   d. Sends IPIs to trigger enclave thread exit and TLB shootdown.
   e. OS informs the Enclave that all logical processors should now see the new restricted permissions.
3. Enclave invokes the EACCEPT leaf function.
   a. Enclave may access the page throughout the entire process.
   b. Successful call to EACCEPT guarantees that no stale cached linear-address to physical-address translations are present.

39.5.11 Extending the EPCM Permissions of a Page

On processors that support SGX2, extending the EPCM permissions associated with an enclave page is accomplished directly be the enclave using the EMODPE leaf function. After performing the EPCM permission extension, the enclave requests the OS to update the page table permissions to match the extended permission. Security wise, permission extension does not require enclave threads to leave the enclave as TLBs with stale references to the more restrictive permissions will be flushed on demand, but to allow forward progress, an OS needs to be aware that an application might signal a page fault.

The typical process for extending the permissions of an enclave page is as follows:

1. Enclave invokes EMODPE to extend the EPCM permissions associated with an EPC page (EMODPE may only be called on VALID pages).
2. Enclave requests that OS update the page tables to match the new EPCM permissions.
3. Enclave code resumes.
   a. If cached linear-address to physical-address translations are present to the more restrictive permissions, the enclave thread will page fault. The SGX2-aware OS will see that the page tables permit the access and resume the thread, which can now successfully access the page because exiting cleared the TLB.
   b. If cached linear-address to physical-address translations are not present, access to the page with the new permissions will succeed without an enclave exit.

...

39.6.1 Illegal Instructions

The instructions listed in Table 38-1 are ring 3 instructions which become illegal when executed inside an enclave. Executing these instructions inside an enclave will generate an exception.

The first row of Table 38-1 enumerates instructions that may cause a VM exit for VMM emulation. Since a VMM cannot emulate enclave execution, execution of any these instructions inside an enclave results in an invalid-opcode exception (#UD) and no VM exit.

The second row of Table 38-1 enumerates I/O instructions that may cause a fault or a VM exit for emulation. Again, enclave execution cannot be emulated, so execution of any these instructions inside an enclave results in #UD.

The third row of Table 38-1 enumerates instructions that load descriptors from the GDT or the LDT or that change privilege level. The former class is disallowed because enclave software should not depend on the contents of the descriptor tables and the latter because enclave execution must be entirely with CPL = 3. Again, execution of any these instructions inside an enclave results in #UD.

The fourth row of Table 38-1 enumerates instructions that provide access to kernel information from user mode and can be used to aid kernel exploits from within enclave. Execution of any these instructions inside an enclave results in #UD

Table 39-1 Illegal Instructions Inside an Enclave

Instructions	Result	Comment
CPUID, GETSEC, RDPMC, SGDT, SIDT, SLDT, STR, VMCALL, VMFUNC	#UD	Might cause VM exit.
IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD	#UD	I/O fault may not safely recover. May require emulation.
Far call, Far jump, Far Ret, INT n/INTO, IRET, LDS/LES/LFS/LGS/LSS, MOV to DS/ES/SS/FS/GS, POP DS/ES/SS/FS/GS, SYSCALL, SYSENTER	#UD	Access segment register could change privilege level.
SMSW	#UD	Might provide access to kernel information.
ENCLU[EENTER], ENCLU[ERESUME]	#GP	Cannot enter an enclave from within an enclave.

RDTSC and RDTSCP are legal inside an enclave for processors that support SGX2 (subject to the value of CR4.TSD). For processors which support SGX1 but not SGX2, RDTSC and RDTSCP will cause #UD.

RDTSC and RDTSCP instructions may cause a VM exit when inside an enclave.

Software developers must take into account that the RDTSC/RDTSCP results are not immune to influences by other software, e.g. the TSC can be manipulated by software outside the enclave.

NOTE

Some early processor implementation of Intel SGX will generate a #UD when RDTSC and RDTSCP are executed inside an enclave. See the model-specific processor errata for details of which processors treat execution of RDTSC and RDTSCP inside an enclave as illegal.

...

32. Updates to Chapter 40, Volume 3D

Change bars show changes to Chapter 40 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3D:* System Programming Guide, Part 4.

Certain events, such as exceptions and interrupts, incident to (but asynchronous with) enclave execution may cause control to transition outside of enclave mode. (Most of these also cause a change of privilege level.) To protect the integrity and security of the enclave, the processor will exit the enclave (and enclave mode) before invoking the handler for such an event. For that reason, such events are called an **enclave-exiting events** (EEE); EEEs include external interrupts, non-maskable interrupts, system-management interrupts, exceptions, and VM exits.

The process of leaving an enclave in response to an EEE is called an **asynchronous enclave exit** (AEX). To protect the secrecy of the enclave, an AEX saves the state of certain registers within enclave memory and then loads those registers with fixed values called **synthetic state**.

40.1 COMPATIBLE SWITCH TO THE EXITING STACK OF AEX

AEXs load registers with a pre-determined synthetic state. These register may be later pushed onto the appropriate stack in a form as defined by the enclave-exiting event. To allow enclave execution to resume after the invoking handler has process the enclave exiting event, the asynchronous enclave exit loads the address of trampoline code outside of the enclave into RIP. This trampoline code eventually returns to the enclave by means of an ENCLU(ERESUME) leaf function. Prior to exiting the enclave the RSP and RBP registers are restored to their values prior to enclave entry.

The stack to be used is chosen using the same rules as for non-SGX mode:

- If there is a privilege level change, the stack will be the one associated with the new ring.
- If there is no privilege level change, the current application stack is used.
- If the IA-32e IST mechanism is used, the exit stack is chosen using that method.

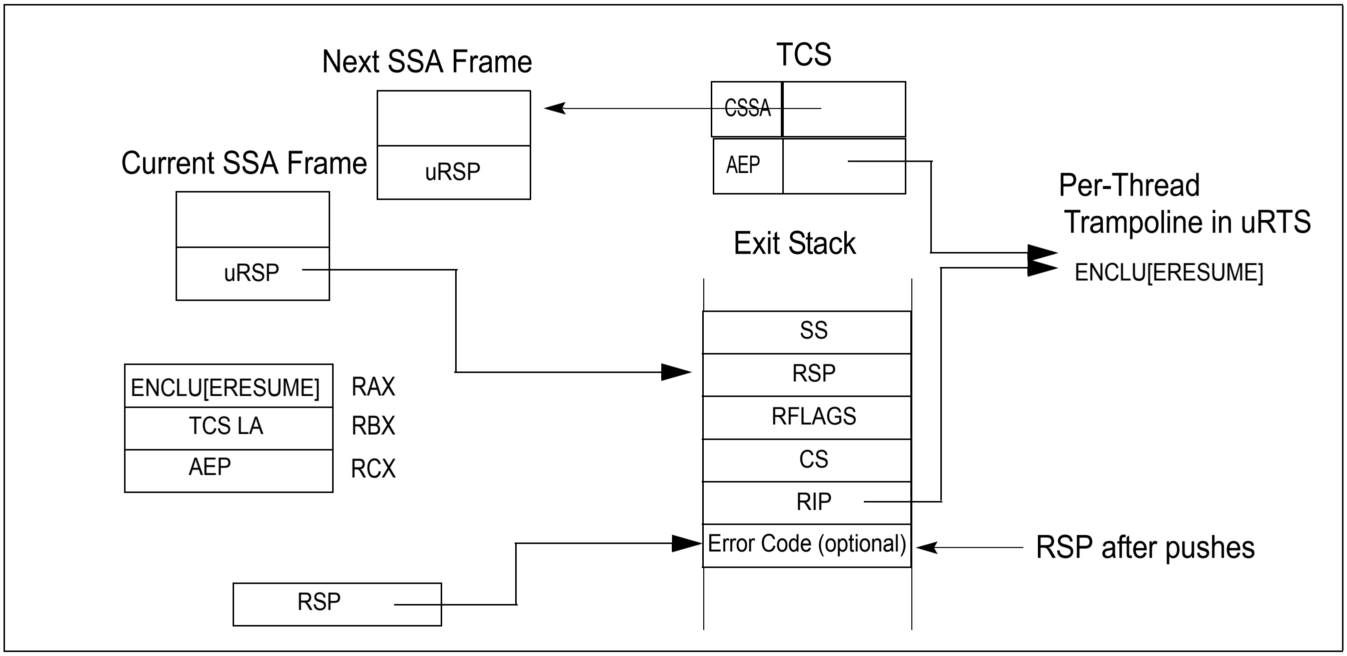


Figure 40-1 Exit Stack Just After Interrupt with Stack Switch

In all cases, the choice of exit stack and the information pushed onto it is consistent with non-SGX operation. Figure 40-1 shows the Application and Exiting Stacks after an exit with a stack switch. An exit without a stack switch uses the Application Stack. The ERESUME leaf index value is placed into RAX, the TCS pointer is placed in RBX and the AEP (see below) is placed into RCX to facilitate resuming the enclave after the exit.

Upon an AEX, the AEP (Asynchronous Exit Pointer) is loaded into the RIP. The AEP points to a trampoline code sequence which includes the ERESUME instruction that is later used to reenter the enclave.

The following bits of RFLAGS are cleared before RFLAGS is pushed onto the exit stack: CF, PF, AF, ZF, SF, OF, RF. The remaining bits are left unchanged.

40.2 STATE SAVING BY AEX

The State Save Area holds the processor state at the time of an AEX. To allow handling events within the enclave and re-entering it after an AEX, the SSA can be a stack of multiple SSA frames as illustrated in Figure 40-2.

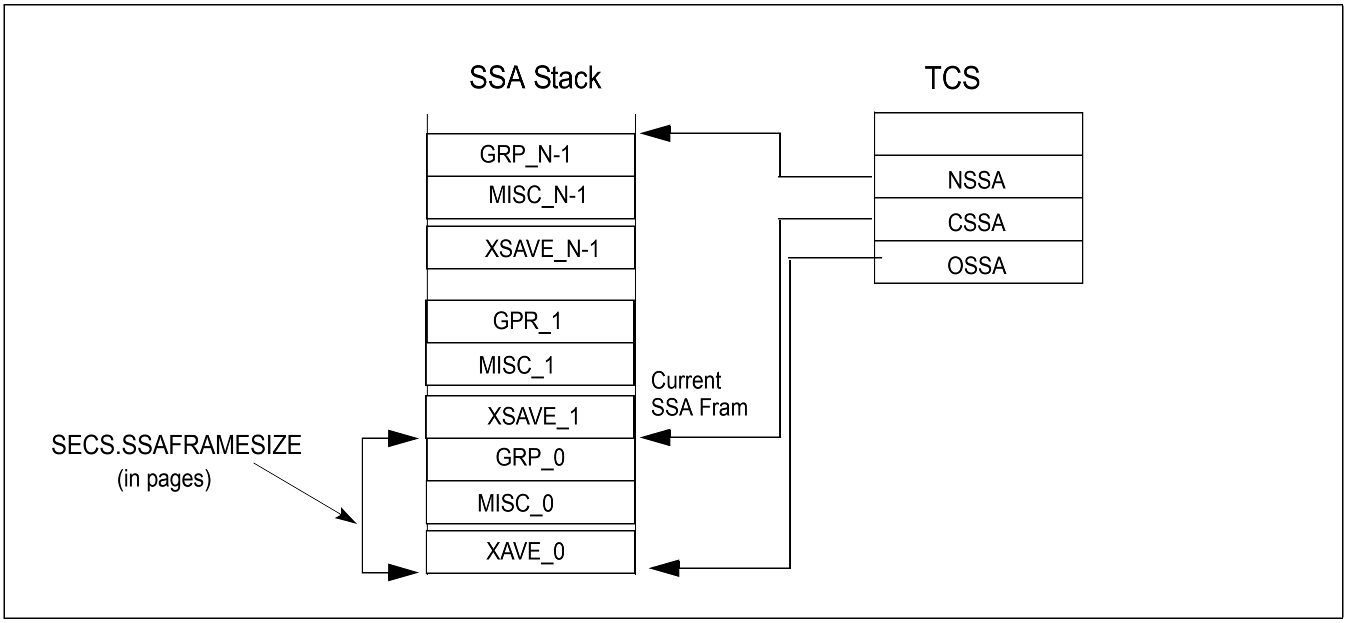


Figure 40-2 The SSA Stack

The location of the SSA frames to be used is controlled by the following variables in the TCS and the SECS:

- Size of a frame in the State Save Area (SECS.SSAFRAMESIZE): This defines the number of 4K Byte pages in a single frame in the State Save Area. The SSA frame size must be large enough to hold the GPR state, the XSAVE state, and the MISC state.
- Base address of the enclave (SECS.BASEADDR): This defines the enclave's base linear address from which the offset to the base of the SSA stack is calculated.
- Number of State Save Area Slots (TCS.NSSA): This defines the total number of slots (frames) in the State Save Area stack.
- Current State Save Area Slot (TCS.CSSA): This defines the slot to use on the next exit.
- State Save Area (TCS.OSSA): This defines the offset of the base address of a set of State Save Area slots from the enclave's base address.

When an AEX occurs, hardware selects the SSA frame to use by examining TCS.CSSA. Processor state is saved into the SSA frame (see Section 40.4) and loaded with a synthetic state (as described in Section 40.3.1)to avoid leaking secrets, RSP and RP are restored to their values prior to enclave entry, and TCS.CSSA is incremented. As will be described later, if an exception takes the last slot, it will not be possible to reenter the enclave to handle the exception from within the enclave. A subsequent ERESUME restores the processor state from the current SSA frame and frees the SSA frame.

The format of the XSAVE section of SSA is identical to the format used by the XSAVE/XRSTOR instructions. On EENTER, CSSA must be less than NSSA, ensuring that there is at least one State Save Area slot available for exits. If there is no free SSA frame when executing EENTER, the entry will fail.

...

40.3.1 Processor Synthetic State on Asynchronous Enclave Exit

Table 40-1 shows the synthetic state loaded on AEX. The values shown are the lower 32 bits when the processor is in 32 bit mode and 64 bits when the processor is in 64 bit mode.

Table 40-1 GPR, x87 Synthetic States on Asynchronous Enclave Exit

Register	Value
RAX	3 (ENCLU[3] is ERESUME).
RBX	Pointer to TCS of interrupted enclave thread.
RCX	AEP of interrupted enclave thread.
RDX, RSI, RDI	0.
RSP	Restored from SSA.uRSP.
RBP	Restored from SSA.uRBP.
R8-R15	0 in 64-bit mode; unchanged in 32-bit mode.
RIP	AEP of interrupted enclave thread.
RFLAGS	CF, PF, AF, ZF, SF, OF, RF bits are cleared. All other bits are left unchanged.
x87/SSE State	Unless otherwise listed here, all x87 and SSE state are set to the INIT state. The INIT state is the state that would be loaded by the XRSTOR instruction with bits 1:0 both set in the requested feature bitmask (RFBM), and both clear in XSTATE_BV the XSAVE header.
FCW	On #MF exception: set to 037EH. On all other exits: set to 037FH.
FSW	On #MF exception: set to 8081H. On all other exits: set to 0H.
MXCSR	On #XM exception: set to 1F01H. On all other exits: set to 1FB0H.
CR2	If the event that caused the AEX is a #PF, and the #PF does not directly cause a VM exit, then the low 12 bits are cleared. If the #PF leads directly to a VM exit, CR2 is not updated (usual IA behavior). Note: The low 12 bits are not cleared if a #PF is encountered during the delivery of the EEE that caused the AEX. This is because the #PF was not the EEE.
FS, GS	Restored to values as of most recent EENTER/ERESUME.

...

40.3.3 Synthetic State for MISC Features

State represented by SECS.MISCSELECT might also be overridden by synthetic state after it has been saved into the SSA. State represented by MISCSELECT[0] is not overridden but if the exiting event is a page fault then lower 12 bits of CR2 are cleared.

40.4 AEX FLOW

On Enclave Exiting Events (interrupts, exceptions, VM exits or SMIs), the processor state is securely saved inside the enclave, a synthetic state is loaded and the enclave is exited. The EEE then proceeds in the usual exit-defined fashion. The following sections describes the details of an AEX:

1. The exact processor state saved into the current SSA frame depends on whether the enclave is a 32-bit or a 64-bit enclave. In 32-bit mode (IA32_EFER.LMA = 0 || CS.L = 0), the low 32 bits of the legacy registers (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP and EFLAGS) are stored. The upper 32 bits of the legacy registers and the 64-bit registers (R8 … R15) are not stored.

In 64-bit mode (IA32_EFER.LMA = 1 && CS.L = 1), all 64 bits of the general processor registers (RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8 … R15, RIP and RFLAGS) are stored.

The state of those extended features specified by SECS.ATTRIBUTES.XFRM are stored into the XSAVE area of the current SSA frame. The layout of the x87 and XMM portions (the 1st 512 bytes) depends on the current values of IA32_EFER.LMA and CS.L:

If IA32_EFER.LMA = 0 || CS.L = 0, the same format (32-bit) that XSAVE/FXSAVE uses with these values.

If IA32_EFER.LMA = 1 && CS.L = 1, the same format (64-bit) that XSAVE/FXSAVE uses with these values when REX.W = 1.

The cause of the AEX is saved in the EXITINFO field. See Table 38-9 for details and values of the various fields.

The state of those miscellaneous features (see Section 38.7.2) specified by SECS.MISCSELECT are stored into the MISC area of the current SSA frame.

2. Synthetic state is created for a number of processor registers to present an opaque view of the enclave state. Table 40-1 shows the values for GPRs, x87, SSE, FS, GS, Debug and performance monitoring on AEX. The synthetic state for other extended features (those controlled by XCR0[62:2]) is set to their respective INIT states when their corresponding bit of SECS.ATTRIBUTES.XFRM is set. The INIT state is that state as defined by the behavior of the XRSTOR instruction when HEADER.XSTATE_BV[n] is 0. Synthetic state of those miscellaneous features specified by SECS.MISCSELECT depends on the miscellaneous feature. There is no synthetic state required for the miscellaneous state controlled by SECS.MISCSELECT[0].
3. Any code and data breakpoints that were suppressed at the time of enclave entry are unsuppressed when exiting the enclave.
4. RFLAGS.TF is set to the value that it had at the time of the most recent enclave entry (except for the situation that the entry was opt-in for debug; see Section 43.2). In the SSA, RFLAGS.TF is set to 0.
5. RFLAGS.RF is set to 0 in the synthetic state. In the SSA, the value saved is the same as what would have been saved on stack in the non-SGX case (architectural value of RF). Thus, AEXs due to interrupts, traps, and code breakpoints save RF unmodified into SSA, while AEXs due to other faults save RF as 1 in the SSA.

   If the event causing AEX happened on intermediate iteration of a REP-prefixed instruction, then RF=1 is saved on SSA, irrespective of its priority.
6. Any performance monitoring activity (including PEBS) or profiling activity (LBR, Tracing using Intel PT) on the exiting thread that was suppressed due to the enclave entry on that thread is unsuppressed. Any counting that had been demoted from AnyThread counting to MyThread counting (on one logical processor) is promoted back to AnyThread counting.

40.4.1 AEX Operational Detail

Temp Variables in AEX Operational Flow

Name	Type	Size (bits)	Description
TMP_RIP	Effective Address	32/64	Address of instruction at which to resume execution on ERESUME.
TMP_MODE64	binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1)).
TMP_BRANCH_RECORD	LBR Record	2x64	From/To address to be pushed onto LBR stack.

The pseudo code in this section describes the internal operations that are executed when an AEX occurs in enclave mode. These operations occur just before the normal interrupt or exception processing occurs.

```
(* Save RIP for later use *)
TMP_RIP = Linear Address of Resume RIP
```

```
(* Is the processor in 64-bit mode? *)
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Save all registers, When saving EFLAGS, the TF bit is set to 0 and
   the RF bit is set to what would have been saved on stack in the non-SGX case *)

 IF (TMP_MODE64 = 0)
   THEN
        Save EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EFLAGS, EIP into the current SSA frame using
CR_GPR_PA; (* see Table 41-4 for list of CREGs used to describe internal operation within Intel SGX *)
        SSA.RFLAGS.TF ← 0;
   ELSE   (* TMP_MODE64 = 1 *)
        Save RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8-R15, RFLAGS, RIP into the current SSA frame using
CR_GPR_PA;
        SSA.RFLAGS.TF ← 0;
FI;
Save FS and GS BASE into SSA using CR_GPR_PA;

(* store XSAVE state into the current SSA frame's XSAVE area using the physical addresses
   that were determined and cached at enclave entry time with CR_XSAVE_PAGE_i. *)
For each XSAVE state i defined by (SECS.ATTRIBUTES.XFRM[i] = 1, destination address cached in
CR_XSAVE_PAGE_i)
  SSA.XSAVE.i ← XSAVE_STATE_i;

(* Clear bytes 8 to 23 of XSAVE_HEADER, i.e. the next 16 bytes after XHEADER_BV *)

CR_XSAVE_PAGE_0.XHEADER_BV[191:64] ← 0;

(* Clear bits in XHEADER_BV[63:0] that are not enabled in ATTRIBUTES.XFRM *)

CR_XSAVE_PAGE_0.XHEADER_BV[63:0] ←
   CR_XSAVE_PAGE_0.XHEADER_BV[63:0] & SECS(CR_ACTIVE_SECS).ATTRIBUTES.XFRM;
   Apply synthetic state to GPRs, RFLAGS, extended features, etc.

(* Restore the RSP and RBP from the current SSA frame's GPR area using the physical address
   that was determined and cached at enclave entry time with CR_GPR_PA. *)
RSP ← CR_GPR_PA.URSP;
RBP ← CR_GPR_PA.URBP;
(* Restore the FS and GS *)
FS.selector ← CR_SAVE_FS.selector;
FS.base ← CR_SAVE_FS.base;
FS.limit ← CR_SAVE_FS.limit;
FS.access_rights ← CR_SAVE_FS.access_rights;
GS.selector ← CR_SAVE_GS.selector;
GS.base ← CR_SAVE_GS.base;
GS.limit ← CR_SAVE_GS.limit;
GS.access_rights ← CR_SAVE_GS.access_rights;

(* Examine exception code and update enclave internal states*)
exception_code ← Exception or interrupt vector;
```

```
(* Indicate the exit reason in SSA *)
IF (exception_code = (#DE OR #DB OR #BP OR #BR OR #UD OR #MF OR #AC OR #XM ))
   THEN
       CR_GPR_PA.EXITINFO.VECTOR ← exception_code;
       IF (exception code = #BP)
           THEN CR_GPR_PA.EXITINFO.EXIT_TYPE ← 6;
           ELSE CR_GPR_PA.EXITINFO.EXIT_TYPE ← 3;
       FI;
       CR_GPR_PA.EXITINFO.VALID ← 1;
   ELSE IF (exception_code is #PF or #GP )
       THEN
       (* Check SECS.MISCSELECT using CR_ACTIVE_SECS *)
       IF (SECS.MISCSELECT[0] is set)
           THEN
           CR_GPR_PA.EXITINFO.VECTOR ← exception_code;
           CR_GPR_PA.EXITINFO.EXIT_TYPE ← 3;
           IF (exception_code is #PF)
               THEN
                   SSA.MISC.EXINFO. MADDR ← CR2;
                   SSA.MISC.EXINFO.ERRCD ← PFEC;
                   SSA.MISC.EXINFO.RESERVED ← 0;
           ELSE
               SSA.MISC.EXINFO. MADDR ← 0;
               SSA.MISC.EXINFO.ERRCD ← GPEC;
               SSA.MISC.EXINFO.RESERVED ← 0;
           FI;
           CR_GPR_PA.EXITINFO.VALID ← 1;
       FI;
   ELSE
       CR_GPR_PA.EXITINFO.VECTOR ← 0;
       CR_GPR_PA.EXITINFO.EXIT_TYPE ← 0
       CR_GPR_PA.REASON.VALID ← 0;
FI;

(* Execution will resume at the AEP *)
RIP ← CR_TCS_PA.AEP;

(* Set EAX to the ERESUME leaf index *)
EAX ← 3;

(* Put the TCS LA into RBX for later use by ERESUME *)
RBX ← CR_TCS_LA;

(* Put the AEP into RCX for later use by ERESUME *)
RCX ← CR_TCS_PA.AEP;

(* Increment the SSA frame # *)
CR_TCS_PA.CSSA ← CR_TCS_PA.CSSA + 1;
```

```
(* Restore XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
    THEN XCR0 ← CR_SAVE_XCR0; FI;

Un-suppress all code breakpoints that are outside ELRANGE

(* Update the thread context to show not in enclave mode *)
CR_ENCLAVE_MODE ← 0;

(* Assure consistent translations. *)
Flush linear context including TLBs and paging-structure caches

IF (CR_DBGOPTIN = 0)
    THEN
         Un-suppress all breakpoints that overlap ELRANGE
         (* Clear suppressed breakpoint matches *)
         Restore suppressed breakpoint matches
         (* Restore TF *)
         RFLAGS.TF ← CR_SAVE_TF;
         Un-suppress monitor trap flag;
         Un-suppress branch recording facilities;
         Un-suppress all suppressed performance monitoring activity;
         Promote any sibling-thread counters that were demoted from AnyThread to MyThread during enclave
entry back to AnyThread;
FI;

IF the “monitor trap flag” VM-execution control is 1
    THEN Pend MTF VM Exit at the end of exit; FI;

(* Clear low 12 bits of CR2 on #PF *)
IF (Exception code is #PF)
    THEN CR2 ← CR2 & ~0xFFF; FI;

(* end_of_flow *)
(* Execution continues with normal event processing. *)
```

...

33. Updates to Chapter 41, Volume 3D

Change bars show changes to Chapter 41 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D:* System Programming Guide, Part 4.

...

41.1.5.1 Concurrency Tables of Intel® SGX Instructions

Concurrent restriction of an individual leaf function (ENCLS or ENCLU) with another Intel SGX instruction leaf functions is listed under the **Concurrency Restriction** paragraph of the respective reference pages of the leaf function.

Each cell in the table for a given Intel SGX Instruction leaf details the concurrency restriction when that instruction references the same EPC page (as an explicit or an implicit parameter) as referenced by a concurrent instruction leaf executed on another logical processor. The concurrency behavior of the instruction leaf if focus shown in a given row is denoted by the following:

- 'N': The instructions listed in a given row heading may not execute concurrently with the instruction leaf shown in the respective column. Software should serialize them. For example, multiple ETRACK operations on the same enclave are not allowed to execute concurrently on the same SECS page.
- 'Y': The instruction leaf listed in a given row may execute concurrently with the instruction leaf shown in the respective column. For instance, multiple ELDB/ELDUs are allowed to execute concurrently as long as the selected EPC page is not the same page.
- 'C': The instruction leaf listed in a given row heading may return an error code when executed concurrently with the instruction leaf shown in the respective column.
- 'U': These two instruction leaves may complete, but the occurrence these two simultaneous flows are considered a user program error for which the processor does not enforce any restriction.
- A grey cell indicates the concurrency behavior of the instruction in focus (in the row header) may be different than that of the concurrent instruction (in the column header). The concurrent instruction's behavior is detailed in its respective concurrency table. For example, EBLOCK's SECS parameter is implicit, thus it is always shown as 'Y' in the table. However a concurrent instruction may return an error code when accessing the same page.

For instance, multiple ELDB/ELDUs are allowed to execute as long as the selected EPC page is not the same page. Multiple ETRACK operations are not allowed to execute concurrently.

...

ENCLS—Execute an Enclave System Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 01 CF ENCLS	NP	V/V	SGX1	This instruction is used to execute privileged Intel SGX leaf functions that are used for managing and debugging the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
NP	NA	NA	NA	See Section 41.3

Description

The ENCLS instruction invokes the specified privileged Intel SGX leaf function for managing and debugging enclaves. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLS instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute the instruction when CPL > 0 results in #UD. The instruction produces a general-protection exception (#GP) if CR0.PG = 0 or if an attempt is made to invoke an undefined leaf function.

In VMX non-root operation, execution of ENCLS may cause a VM exit if the "enable ENCLS exiting" VM-execution control is 1. In this case, execution of individual leaf functions of ENCLS is governed by the ENCLS-exiting bitmap field in the VMCS. Each bit in that field corresponds to the index of an ENCLS leaf function (as provided in EAX).

Software in VMX root operation can thus intercept the invocation of various ENCLS leaf functions in VMX non-root operation by setting the “enable ENCLS exiting” VM-execution control and setting the corresponding bits in the ENCLS-exiting bitmap.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 || CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 || CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, and is the REX prefix in 64-bit mode.

Operation

```
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE = 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0
    THEN #UD; FI;

IF (CPL > 0)
    THEN #UD; FI;

IF in VMX non-root operation and the “enable ENCLS exiting“ VM-execution control is 1
    THEN
        IF EAX < 63 and ENCLS_exiting_bitmap[EAX] = 1 or EAX> 62 and ENCLS_exiting_bitmap[63] = 1
            THEN VM exit;
        FI;
FI;
IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;

IF EAX is invalid leaf number)
    THEN #GP(0); FI;

IF CR0.PG = 0
    THEN #GP(0); FI;

(* DS must not be an expanded down segment *)
IF not in 64-bit mode and DS.Type is expand-down data
    THEN #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	If any of the LOCK/OSIZE/REP/VEX prefix is used. If current privilege level is not 0. If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0. If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0. If IA32_FEATURE_CONTROL.SGX_ENABLE = 0. If input value in EAX encodes an unsupported leaf. If data segment expand down. If CR0.PG=0.

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD	ENCLS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/OSIZE/REP/VEX prefix is used. If current privilege level is not 0. If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0. If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0. If IA32_FEATURE_CONTROL.SGX_ENABLE = 0. If input value in EAX encodes an unsupported leaf.

...

ENCLU—Execute an Enclave User Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 01 D7 ENCLU	NP	V/V	SGX1	This instruction is used to execute non-privileged Intel SGX leaf functions.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
NP	NA	NA	NA	See Section 41.4

Description

The ENCLU instruction invokes the specified non-privileged Intel SGX leaf functions. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLU instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute this instruction when CPL < 3 results in #UD. The instruction produces a general-protection exception (#GP) if either CR0.PG or CR0.NE is 0, or if an attempt is made to invoke an undefined leaf function. The ENCLU instruction produces a device not available exception (#NM) if CR0.TS = 1.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 or CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 and CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, and is the REX prefix in 64-bit mode.

Operation

```
IN_64BIT_MODE← 0;
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE= 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0
    THEN #UD; FI;

IF CR0.TS = 1
    THEN #NM; FI;

IF CPL < 3
    THEN #UD; FI;

IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;

IF EAX is invalid leaf number
    THEN #GP(0); FI;

IF CR0.PG = 0 or CR0.NE = 0
    THEN #GP(0); FI;

IN_64BIT_MODE ← IA32_EFER.LMA AND CS.L ? 1 : 0;
(* Check not in 16-bit mode and DS is not a 16-bit segment *)
IF not in 64-bit mode and (CS.D = 0 or DS.B = 0)
    THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 1 and (EAX = 2 or EAX = 3) (* EENTER or ERESUME *)
    THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 0 and (EAX = 0 or EAX = 1 or EAX = 4 or EAX = 5 or EAX = 6 or EAX = 7)
(* EREPORT, EGETKEY, EEXIT, EACCEPT, EMODPE, or EACCEPTCOPY *)
    THEN #GP(0); FI;
```

Jump to leaf specific flow

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD If any of the LOCK/OSIZE/REP/VEX prefix is used.
If current privilege level is not 3.
If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
If logical processor is in SMM.

#GP(0) If IA32_FEATURE_CONTROL.LOCK = 0.
If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
If input value in EAX encodes an unsupported leaf.
If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1.
If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0.
If operating in 16-bit mode.
If data segment is in 16-bit mode.
If CR0.PG = 0 or CR0.NE= 0.

#NM If CR0.TS = 1.

Real-Address Mode Exceptions

#UD ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD ENCLS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD If any of the LOCK/OSIZE/REP/VEX prefix is used.
If current privilege level is not 3.
If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
If logical processor is in SMM.

#GP(0) If IA32_FEATURE_CONTROL.LOCK = 0.
If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
If input value in EAX encodes an unsupported leaf.
If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1.
If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0.
If CR0.NE= 0.

#NM If CR0.TS = 1.

...

EADD—Add a Page to an Uninitialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLS[EADD]	IR	V/V	SGX1	This leaf function adds a page to an uninitialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EADD (In)	Address of a PAGEINFO (In)	Address of the destination EPC page (In)

Description

This leaf function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. As part of the association, the enclave offset and the security attributes are measured and extended into the SECS.MRENCLAVE. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of EADD leaf function.

EADD Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EADD Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	If security attributes specifies a TCS and the source page specifies unsupported TCS values or fields.
The SECS has been initialized.	The specified enclave offset is outside of the enclave address space.

Concurrency Restrictions

Table 41-5 Concurrency Restrictions of EADD with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EADD	Targ				N		N		N	N		N			N				N	N	N	N	N
	SECS			N		N	Y	Y	N		Y			N		N		N	N			Y	N

Table 41-6 Concurrency Restrictions of EADD with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EADD	Targ	N				N	N	N		N	N			N		N							
	SECS	N	Y		N	Y	N		Y	N	N				N	N	N			N			

Operation

Temp Variables in EADD Operational Flow

Name	Type	Size (bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE ← DS:RBX.SRCPGE;
TMP_SECS ← DS:RBX.SECS;
TMP_SECINFO ← DS:RBX.SECINFO;
TMP_LINADDR ← DS:RBX.LINADDR;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECS is not 4KByte aligned or
    DS:TMP_SECINFO is not 64Byte aligned or TMP_LINADDR is not 4KByte aligned)
    THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
    THEN #PF(DS:TMP_SECS); FI;

SCRATCH_SECINFO ← DS:TMP_SECINFO;

(* Check for mis-configured SECINFO flags*)
```

```
IF (SCRATCH_SECINFO reserved fields are not zero or
    ! (SCRATCH_SECINFO.FLAGS.PT is PT_REG or SCRATCH_SECINFO.FLAGS.PT is PT_TCS) )
    THEN #GP(0); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EADD)
    THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] ← DS:TMP_SRCPGE[32767:0];

CASE (SCRATCH_SECINFO.FLAGS.PT)
{
    PT_TCS:
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
        IF ( (DS:TMP_SECS.ATTIBUTES.MODE64BIT = 0) and
            ((DS:TCS.FSLIMIT & 0FFFH ≠ 0FFFH) or (DS:TCS.GSLIMIT & 0FFFH ≠ 0FFFH) )) #GP(0); FI;
        BREAK;
    PT_REG:
        IF (SCRATCH_SECINFO.FLAGS.W = 1 and SCRATCH_SECINFO.FLAGS.R = 0) #GP(0); FI;
        BREAK;
ESAC;

(* Check the enclave offset is within the enclave linear address space *)
IF (TMP_LINADDR < DS:TMP_SECS.BASEADDR or TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE)
    THEN #GP(0); FI;

(* Check concurrency of measurement resource*)
IF (Measurement being updated)
    THEN #GP(0); FI;

(* Check if the enclave to which the page will be added is already in Initialized state *)
IF (DS:TMP_SECS already initialized)
    THEN #GP(0); FI;

(* For TCS pages, force EPCM.rwx bits to 0 and no debug access *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        SCRATCH_SECINFO.FLAGS.R ← 0;
        SCRATCH_SECINFO.FLAGS.W ← 0;
        SCRATCH_SECINFO.FLAGS.X ← 0;
        (DS:RCX).FLAGS.DBGOPTIN ← 0; // force TCS.FLAGS.DBGOPTIN off
        DS:RCX.CSSA ← 0;
        DS:RCX.AEP ← 0;
        DS:RCX.STATE ← 0;
FI;

(* Add enclave offset and security attributes to MRENCLAVE *)
TMP_ENCLAVEOFFSET ← TMP_LINADDR - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] ← 0000000044444145H; // "EADD"
TMPUPDATEFIELD[127:64] ← TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] ← SCRATCH_SECINFO[375:0]; // 48 bytes
DS:TMP_SECS.MRENCLAVE ← SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(* Add enclave offset and security attributes to MRENCLAVE *)
EPCM(DS:RCX).R ← SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W ← SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X ← SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT ← SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS ← TMP_LINADDR;

(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED ← 0;
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).VALID ← 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

...

EAUG—Add a Page to an Initialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0DH ENCLS[EAUG]	IR	V/V	SGX2	This leaf function adds a page to an initialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EAUG (In)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function zeroes a page of EPC memory, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in the EPCM. As part of the association, the security attributes are configured to prevent access to the EPC page until a corresponding invocation of the EACCEPT leaf or EACCEPTCOPY leaf confirms the addition of the new page into the enclave. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EAUG leaf function.

EAUG Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Must be zero	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EAUG Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	The specified enclave offset is outside of the enclave address space.
The SECS has been initialized.	

Concurrency Restrictions

Table 41-7 Concurrency Restrictions of EAUG with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EAUG	Targ				N	N	N		N	N		N			N				N	N	N	N	N
	SECS			Y	N	N		Y	N		Y			Y		N		Y	N	N		Y	N

Table 41-8 Concurrency Restrictions of EAUG with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EAUG	Targ	N				N	N	N		N	N			N		N							
	SECS	N	Y		Y	Y	N		Y	N	Y				Y	N	Y			Y			

Operation

Temp Variables in EAUG Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;
```

```
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SECS ← DS:RBX.SECS;
TMP_LINADDR ← DS:RBX.LINADDR;

IF ( DS:TMP_SECS is not 4KByte aligned or TMP_LINADDR is not 4KByte aligned )
    THEN #GP(0); FI;

IF ( (DS:RBX.SRCPAGE is not 0) or (DS:RBX:SECINFO is not 0) )
    THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
    THEN #PF(DS:SECS); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EAUG)
    THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

(* Check if the enclave to which the page will be added is in the Initialized state *)
IF (DS:TMP_SECS is not initialized)
    THEN #GP(0); FI;

(* Check the enclave offset is within the enclave linear address space *)
IF ( (TMP_LINADDR < DS:TMP_SECS.BASEADDR) or (TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE) )
    THEN #GP(0); FI;

(* Clear the content of EPC page*)
DS:RCX[32767:0] ← 0;

(* Set EPCM security attributes *)
EPCM(DS:RCX).R ← 1;
EPCM(DS:RCX).W ← 1;
EPCM(DS:RCX).X ← 0;
EPCM(DS:RCX).PT ← PT_REG;
EPCM(DS:RCX).ENCLAVEADDRESS ← TMP_LINADDR;
EPCM(DS:RCX).BLOCKED ← 0;
EPCM(DS:RCX).PENDING ← 1;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).PR ← 0;
```

```
(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;

(* Set EPCM valid fields *)
EPCM(DS:RCX).VALID ← 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If a memory operand is locked.
If the enclave is not initialized.

#PF(error code) If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If a memory operand is locked.
If the enclave is not initialized.

#PF(error code) If a page fault occurs in accessing memory operands.

...

EBLOCK—Mark a page in EPC as Blocked

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLS[EBLOCK]	IR	V/V	SGX1	This leaf function marks a page in the EPC as blocked.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	EBLOCK (In)	Return error code (Out)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be marked as BLOCKED. This instruction can only be executed when current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

An error code is returned in RAX.

The table below provides additional information on the memory parameter of EBLOCK leaf function.

EBLOCK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 41-9 EBLOCK Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EBLOCK successful
SGX_BLKSTATE	Page already blocked. This value is used to indicate to a VMM that the page was already in BLOCKED state as a result of EBLOCK and thus will need to be restored to this state when it is eventually reloaded (using ELDB).
SGX_ENTRYEPOCH_LOCKED	SECS locked for Entry Epoch update. This value indicates that an ETRACK is currently executing on the SECS. The EBLOCK should be reattempted.
SGX_NOTBLOCKABLE	Page type is not one which can be blocked
SGX_PG_INVLD	Page is not valid and cannot be blocked
SGX_LOCKFAIL	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB

Concurrency Restrictions

Table 41-10 Concurrency Restrictions of EBLOCK with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EBLOCK	Targ	Y	Y	Y	N	C	C	C	N	Y	C	Y	Y	C	Y	C	Y	C	Y	N	C	C	N
	SECS			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-11 Concurrency Restrictions of EBLOCK with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EBLOCK	Targ	N	C	Y	C	C	N	C	C	N	C	Y	Y	Y	C	N	C	Y	Y	C	Y	Y	Y
	SECS	Y	Y	Y	Y	U	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EBLOCK Operational Flow

Name	Type	Size (Bits)	Description
TMP_BLKSTATE	Integer	64	Page is already blocked.

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
```

```
        THEN #PF(DS:RCX); FI;

RFLAGS.ZF,CF,PF,AF,OF,SF ← 0;
RAX← 0;

IF (EPCM(DS:RCX). VALID = 0)
    THEN
        RFLAGS.ZF ← 1;
        RAX← SGX_PG_INVLD;
        GOTO DONE;
FI;

IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_TRIM) )
    THEN
        RFLAGS.CF ← 1;
        IF (EPCM(DS:RCX).PT = PT_SECS)
            THEN RAX← SGX_PG_IS_SECS;
            ELSE RAX← SGX_NOTBLOCKABLE;
        FI;
        GOTO DONE;
FI;

(* Check if the page is already blocked and report blocked state *)
TMP_BLKSTATE ← EPCM(DS:RCX).BLOCKED;

(* at this point, the page must be valid and PT_TCS or PT_REG or PT_TRIM*)
IF (TMP_BLKSTATE = 1) )
    THEN
        RFLAGS.CF ← 1;
        RAX← SGX_BLKSTATE;
    ELSE
        EPCM(DS:RCX).BLOCKED ← 1
FI;

DONE:
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Sets CF if page is BLOCKED or not blockable, otherwise cleared. Clears PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

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

	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

...

ECREATE—Create an SECS page in the Enclave Page Cache

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLS[ECREATE]	IR	V/V	SGX1	This leaf function begins an enclave build by creating an SECS page in EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	ECREATE (In)	Address of a PAGEINFO (In)	Address of the destination SECS page (In)

Description

ENCLS[ECREATE] is the first instruction executed in the enclave build process. ECREATE copies an SECS structure outside the EPC into an SECS page inside the EPC. The internal structure of SECS is not accessible to software.

ECREATE will set up fields in the protected SECS and mark the page as valid inside the EPC. ECREATE initializes or checks unused fields.

Software sets the following fields in the source structure: SECS:BASEADDR, SECS:SIZE in bytes, and ATTRIBUTES. SECS:BASEADDR must be naturally aligned on an SECS.SIZE boundary. SECS.SIZE must be at least 2 pages (8192).

The source operand RBX contains an effective address of a PAGEINFO structure. PAGEINFO contains an effective address of a source SECS and an effective address of an SECINFO. The SECS field in PAGEINFO is not used.

The RCX register is the effective address of the destination SECS. It is an address of an empty slot in the EPC. The SECS structure must be page aligned. SECINFO flags must specify the page as an SECS page.

ECREATE Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

ECREATE will fault if the SECS target page is in use; already valid; outside the EPC. It will also fault if addresses are not aligned; unused PAGEINFO fields are not zero.

If the amount of space needed to store the SSA frame is greater than the amount specified in SECS.SSAFRAMESIZE, a #GP(0) results. The amount of space needed for an SSA frame is computed based on DS:TMP_SECS.ATTRIBUTES.XFRM size. Details of computing the size can be found Section 42.7.

Concurrency Restrictions

Table 41-12 Concurrency Restrictions of ECREATE with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
ECREATE	SECS				N	N	N		N	N		N			N				N	N	N	N	N

Table 41-13 Concurrency Restrictions of ECREATE with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
ECREATE	SECS	N				N	N	N		N	N			N		N							

Operation

Temp Variables in ECREATE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the SECS source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the SECS page to be added.
TMP_XSIZE	SSA Size	64	The size calculation of SSA frame.
TMP_MISC_SIZE	MISC Field Size	64	Size of the selected MISC field components.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE ← DS:RBX.SRCPGE;
TMP_SECINFO ← DS:RBX.SECINFO;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECINFO is not 64Byte aligned)
    THEN #GP(0); FI;

IF (DS:RBX.LINADDR ! = 0 or DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

(* Check for misconfigured SECINFO flags*)
```

```
IF (DS:TMP_SECINFO reserved fields are not zero or DS:TMP_SECINFO.FLAGS.PT ≠ PT_SECS) )
    THEN #GP(0); FI;

TMP_SECS ← RCX;

IF (EPC entry in use)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] ← DS:TMP_SRCPGE[32767:0];

(* Check lower 2 bits of XFRM are set *)
IF ( ( DS:TMP_SECS.ATTRIBUTES.XFRM BitwiseAND 03H) ≠ 03H)
    THEN #GP(0); FI;

IF (XFRM is illegal)
    THEN #GP(0); FI;

(* Make sure that the SECS does not have any unsupported MISCSELECT options*)
IF ( !(CPUID.(EAX=12H, ECX=0):EBX[31:0] & DS:TMP_SECS.MISSELECT[31:0]) )
    THEN
        EPCM(DS:TMP_SECS).EntryLock.Release();
        #GP(0);
FI;

( * Compute size of MISC area *)
TMP_MISC_SIZE ← compute_misc_region_size();

(* Compute the size required to save state of the enclave on async exit, see Section 42.7.2.2*)
TMP_XSIZE ← compute_xsave_size(DS:TMP_SECS.ATTRIBUTES.XFRM) + GPR_SIZE + TMP_MISC_SIZE;

(* Ensure that the declared area is large enough to hold XSAVE and GPR stat *)
IF ( ( DS:TMP_SECS.SSAFRAMESIZE*4096 < TMP_XSIZE)
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.BASEADDR is not canonical) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.BASEADDR and 0FFFFFFFF00000000H) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[7:0]) ) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[15:8]) ) )
    THEN #GP(0); FI;
```

```
(* Enclave size must be at least 8192 bytes and must be power of 2 in bytes*)
IF (DS:TMP_SECS.SIZE < 8192 or popcnt(DS:TMP_SECS.SIZE) > 1)
    THEN #GP(0); FI;

(* Ensure base address of an enclave is aligned on size*)
IF ( ( DS:TMP_SECS.BASEADDR and (DS:TMP_SECS.SIZE-1) )
    THEN #GP(0); FI;

* Ensure the SECS does not have any unsupported attributes*)
IF ( ( DS:TMP_SECS.ATTRIBUTES and (~CR_SGX_ATTRIBUTES_MASK) )
    THEN #GP(0); FI;

IF ( ( DS:TMP_SECS reserved fields are not zero)
    THEN #GP(0); FI;

Clear DS:TMP_SECS to Uninitialized;
DS:TMP_SECS.MRENCLAVE ← SHA256INITIALIZE(DS:TMP_SECS.MRENCLAVE);
DS:TMP_SECS.ISVSVN ← 0;
DS:TMP_SECS.ISVPRODID ← 0;

(* Initialize hash updates etc*)
Initialize enclave's MRENCLAVE update counter;

(* Add "ECREATE" string and SECS fields to MRENCLAVE *)
TMPUPDATEFIELD[63:0] ← 0045544145524345H; // "ECREATE"
TMPUPDATEFIELD[95:64] ← DS:TMP_SECS.SSAFRAMESIZE;
TMPUPDATEFIELD[159:96] ← DS:TMP_SECS.SIZE;
TMPUPDATEFIELD[511:160] ← 0;
SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(* Set EID *)
DS:TMP_SECS.EID ← LockedXAdd(CR_NEXT_EID, 1);

(* Set the EPCM entry, first create SECS identifier and store the identifier in EPCM *)
EPCM(DS:TMP_SECS).PT ← PT_SECS;
EPCM(DS:TMP_SECS).ENCLAVEADDRESS ← 0;
EPCM(DS:TMP_SECS).R ← 0;
EPCM(DS:TMP_SECS).W ← 0;
EPCM(DS:TMP_SECS).X ← 0;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED ← 0;
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).PR ← 0;
EPCM(DS:RCX).VALID ← 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If the reserved fields are not zero.
If PAGEINFO.SECS is not zero.
If PAGEINFO.LINADDR is not zero.
If the SECS destination is locked.
If SECS.SSAFRAMESIZE is insufficient.

#PF(error code) If a page fault occurs in accessing memory operands.
If the SECS destination is outside the EPC.

64-Bit Mode Exceptions

#GP(0) If a memory address is non-canonical form.
If a memory operand is not properly aligned.
If the reserved fields are not zero.
If PAGEINFO.SECS is not zero.
If PAGEINFO.LINADDR is not zero.
If the SECS destination is locked.
If SECS.SSAFRAMESIZE is insufficient.

#PF(error code) If a page fault occurs in accessing memory operands.
If the SECS destination is outside the EPC.

...

EDBGRD—Read From a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLS[EDBGRD]	IR	V/V	SGX1	This leaf function reads a dword/quadword from a debug enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EDBGRD (In)	Data read from a debug enclave (Out)	Address of source memory in the EPC (In)

Description

This leaf function copies a quadword/doubleword from an EPC page belonging to a debug enclave into the RBX register. Eight bytes are read in 64-bit mode, four bytes are read in non-64-bit modes. The size of data read cannot be overridden.

The effective address of the source location inside the EPC is provided in the register RCX.

EDBGRD Memory Parameter Semantics

EPCQW
Read access permitted by Enclave

The error codes are:

Table 41-14 EDBGRD Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EDBGRD successful
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state

The instruction faults if any of the following:

EDBGRD Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is beyond the architectural size of the TCS (SGX_TCS_LIMIT).
An operand causing any segment violation.	May page fault.
CPL > 0.	

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDGBRD does not result in a #GP.

Concurrency Restrictions

Table 41-15 Concurrency Restrictions of EDBGRD with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EDBGRD	Targ	Y	Y		N		Y		N	Y		Y	Y		Y		Y		N	N	Y		N
	SECS			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-16 Concurrency Restrictions of EDBGRD with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EDBGRD	Targ	N		Y		N	N	Y		N		Y	Y	Y		N			Y			Y	Y
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EDBGRD Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1))
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs

```
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ( (TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned) )
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing EPCM *)
IF (Other EPCM modifying instructions executing)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (SOURCE) is pointing to a PT_REG or PT_TCS or PT_VA *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_VA))
    THEN #PF(DS:RCX); FI;
```

```
(* If source is a TCS, then make sure that the offset into the page is not beyond the TCS size*)
IF ( ( EPCM(DS:RCX). PT = PT_TCS) and ((DS:RCX) & FFFH ≥ SGX_TCS_LIMIT) )
    THEN #GP(0); FI;

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF ( (EPCM(DS:RCX).PT = PT_REG) or (EPCM(DS:RCX).PT = PT_TCS) )
    THEN
        TMP_SECS ← GET_SECS_ADDRESS;
        IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
            THEN #GP(0); FI;
        IF ( (TMP_MODE64 = 1) )
            THEN RBX[63:0] ← (DS:RCX)[63:0];
            ELSE EBX[31:0] ← (DS:RCX)[31:0];
        FI;
    ELSE
        TMP_64BIT_VAL[63:0] ← (DS:RCX)[63:0] & (~07H); // Read contents from VA slot
        IF (TMP_MODE64 = 1)
            THEN
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN RBX[63:0] ← 0FFFFFFFFFFFFFFFFH;
                    ELSE RBX[63:0] ← 0H;
                FI;
            ELSE
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN EBX[31:0] ← 0FFFFFFFFH;
                    ELSE EBX[31:0] ← 0H;
                FI;
FI;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0) If the address in RCS violates DS limit or access rights.

If DS segment is unusable.

If RCX points to a memory location not 4Byte-aligned.

If the address in RCX points to a page belonging to a non-debug enclave.

If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA.

If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.

#PF(error code) If a page fault occurs in accessing memory operands.

If the address in RCX points to a non-EPC page.

If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form. If RCX points to a memory location not 8Byte-aligned. If the address in RCX points to a page belonging to a non-debug enclave. If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA. If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.
#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

...

EDBGWR—Write to a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLS[EDBGWR]	IR	V/V	SGX1	This leaf function writes a dword/quadword to a debug enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EDBGWR (In)	Data to be written to a debug enclave (In)	Address of Target memory in the EPC (In)

Description

This leaf function copies the content in EBX/RBX to an EPC page belonging to a debug enclave. Eight bytes are written in 64-bit mode, four bytes are written in non-64-bit modes. The size of data cannot be overridden.

The effective address of the source location inside the EPC is provided in the register RCX

EDBGWR Memory Parameter Semantics

EPCQW
Write access permitted by Enclave

The instruction faults if any of the following:

EDBGWR Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is not the FLAGS word.
An operand causing any segment violation.	May page fault.
CPL > 0.	

The error codes are:

Table 41-17 EDBGWR Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EDBGWR successful
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDGBRD does not result in a #GP.

Concurrency Restrictions

Table 41-18 Concurrency Restrictions of EDBGWR with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EDBGWR	Targ	Y	Y	▨	N	▨	Y	▨	N	Y	▨	Y	Y	▨	Y	▨	Y	▨	N	N	Y	▨	N
	SECS	▨	▨	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-19 Concurrency Restrictions of EDBGWR with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EDBGWR	Targ	N	▨	Y	▨	N	N	Y	▨	N	▨	Y	Y	Y	▨	N	▨	▨	Y	▨	▨	Y	Y
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EDBGWR Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1)).
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.

```
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ( (TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned) )
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;
```

```
(* make sure no other Intel SGX instruction is accessing EPCM *)
IF (Other EPCM modifying instructions executing)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) )
    THEN #PF(DS:RCX); FI;

(* If destination is a TCS, then make sure that the offset into the page can only point to the FLAGS field*)
IF ( ( EPCM(DS:RCX). PT = PT_TCS) and ((DS:RCX) & FF8H ≠ offset_of_FLAGS & 0FF8H) )
    THEN #GP(0); FI;

(* Locate the SECS for the enclave to which the DS:RCX page belongs *)
TMP_SECS ← GET_SECS_PHYS_ADDRESS(EPCM(DS:RCX).ENCLAVESCES);

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 1) )
    THEN (DS:RCX)[63:0] ← RBX[63:0];
    ELSE (DS:RCX)[31:0] ← EBX[31:0];
FI;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights.
	If DS segment is unusable.
	If RCX points to a memory location not 4Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a location inside TCS that is not the FLAGS word.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.
	If RCX points to a memory location not 8Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a location inside TCS that is not the FLAGS word.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

...

EEXTEND—Extend Uninitialized Enclave Measurement by 256 Bytes

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLS[EEXTEND]	IR	V/V	SGX1	This leaf function measures 256 bytes of an uninitialized enclave page.

Instruction Operand Encoding

Op/En	EAX	EBX	RCX
IR	EEXTEND (In)	Effective address of the SECS of the data chunk (In)	Effective address of a 256-byte chunk in the EPC (In)

Description

This leaf function updates the MRENCLAVE measurement register of an SECS with the measurement of an EXTEND string compromising of “EEXTEND” || ENCLAVEOFFSET || PADDING || 256 bytes of the enclave page. This instruction can only be executed when current privilege level is 0 and the enclave is uninitialized.

RBX contains the effective address of the SECS of the region to be measured. The address must be the same as the one used to add the page into the enclave.

RCX contains the effective address of the 256 byte region of an EPC page to be measured. The DS segment is used to create linear addresses. Segment override is not supported.

EEXTEND Memory Parameter Semantics

EPC[RCX]
Read access by Enclave

The instruction faults if any of the following:

EEXTEND Faulting Conditions

RBX points to an address not 4KBytes aligned.	RBX does not resolve to an SECS.
RBX does not point to an SECS page.	RBX does not point to the SECS page of the data chunk.
RCX points and address not 256B aligned.	RCX points to an unused page or a SECS.
RCX does not resolve in an EPC page.	If SECS is locked.
If the SECS is already initialized.	May page fault.
CPL > 0.	

Concurrency Restrictions

Table 41-20 Concurrency Restrictions of EEXTEND with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EEXTEND	Targ	N	N		N		Y		N	Y					Y				N	N			N
	SECS	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	N	N	Y	Y	Y	Y

Table 41-21 Concurrency Restrictions of EEXTEND with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EEXTEND	Targ	N					N			N				N		N							
	SECS	Y	Y	Y	N	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y	N	Y	Y	N	Y	Y	Y

Operation

Temp Variables in EEXTEND Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF (DS:RBX does resolve to an EPC page)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX is not 256Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
```

```
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing EPCM *)
IF (Other instructions accessing EPCM)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) )
    THEN #PF(DS:RCX); FI;

TMP_SECS ← Get_SECS_ADDRESS();

IF (DS:RBX does not resolve to TMP_SECS)
    THEN #GP(0); FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUETS.INIT *)
IF ( (Other instruction accessing MRENCLAVE) or (Other instructions checking or updating the initialized state of the SECS))
    THEN #GP(0); FI;

(* Calculate enclave offset *)
TMP_ENCLAVEOFFSET ←   EPCM(DS:RCX).ENCLAVEADDRESS - TMP_SECS.BASEADDR;
TMP_ENCLAVEOFFSET ←   TMP_ENCLAVEOFFSET + (DS:RCX & 0FFFH)

(* Add EEXTEND message and offset to MRENCLAVE *)
TMPUPDATEFIELD[63:0] ← 00444E4554584545H; // "EEXTEND"
TMPUPDATEFIELD[127:64] ← TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] ← 0; // 48 bytes
TMP_SECS.MRENCLAVE ← SHA256UPDATE(TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(*Add 256 bytes to MRENCLAVE, 64 byte at a time *)
TMP_SECS.MRENCLAVE ← SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[511:0] );
TMP_SECS.MRENCLAVE ← SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1023: 512] );
TMP_SECS.MRENCLAVE ← SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1535: 1024] );
TMP_SECS.MRENCLAVE ← SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[2047: 1536] );
INC enclave's MRENCLAVE update counter by 4;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0) If the address in RBX is outside the DS segment limit.
If RBX points to an SECS page which is not the SECS of the data chunk.
If the address in RCX is outside the DS segment limit.
If RCX points to a memory location not 256Byte-aligned.
If another instruction is accessing MRENCLAVE.
If another instruction is checking or updating the SECS.
If the enclave is already initialized.

#PF(error code) If a page fault occurs in accessing memory operands.
If the address in RBX points to a non-EPC page.
If the address in RCX points to a page which is not PT_TCS or PT_REG.
If the address in RCX points to a non-EPC page.
If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0) If RBX is non-canonical form.
If RBX points to an SECS page which is not the SECS of the data chunk.
If RCX is non-canonical form.
If RCX points to a memory location not 256 Byte-aligned.
If another instruction is accessing MRENCLAVE.
If another instruction is checking or updating the SECS.
If the enclave is already initialized.

#PF(error code) If a page fault occurs in accessing memory operands.
If the address in RBX points to a non-EPC page.
If the address in RCX points to a page which is not PT_TCS or PT_REG.
If the address in RCX points to a non-EPC page.
If the address in RCX points to an invalid EPC page.

...

EINIT—Initialize an Enclave for Execution

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLS[EINIT]	IR	V/V	SGX1	This leaf function initializes the enclave and makes it ready to execute enclave code.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EINIT (In)	Error code (Out)	Address of SIGSTRUCT (In)	Address of SECS (In)	Address of EINITTOKEN (In)

Description

This leaf function is the final instruction executed in the enclave build process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to start user code execution using the EENTER instruction.

EINIT takes the effective address of a SIGSTRUCT and EINITTOKEN. The SIGSTRUCT describes the enclave including MRENCLAVE, ATTRIBUTES, ISVSVN, a 3072 bit RSA key, and a signature using the included key. SIGSTRUCT must be populated with two values, q1 and q2. These are calculated using the formulas shown below:

q1 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

The EINITTOKEN contains the MRENCLAVE, MRSIGNER, and ATTRIBUTES. These values must match the corresponding values in the SECS. If the EINITTOKEN was created with a debug launch key, the enclave must be in debug mode as well.

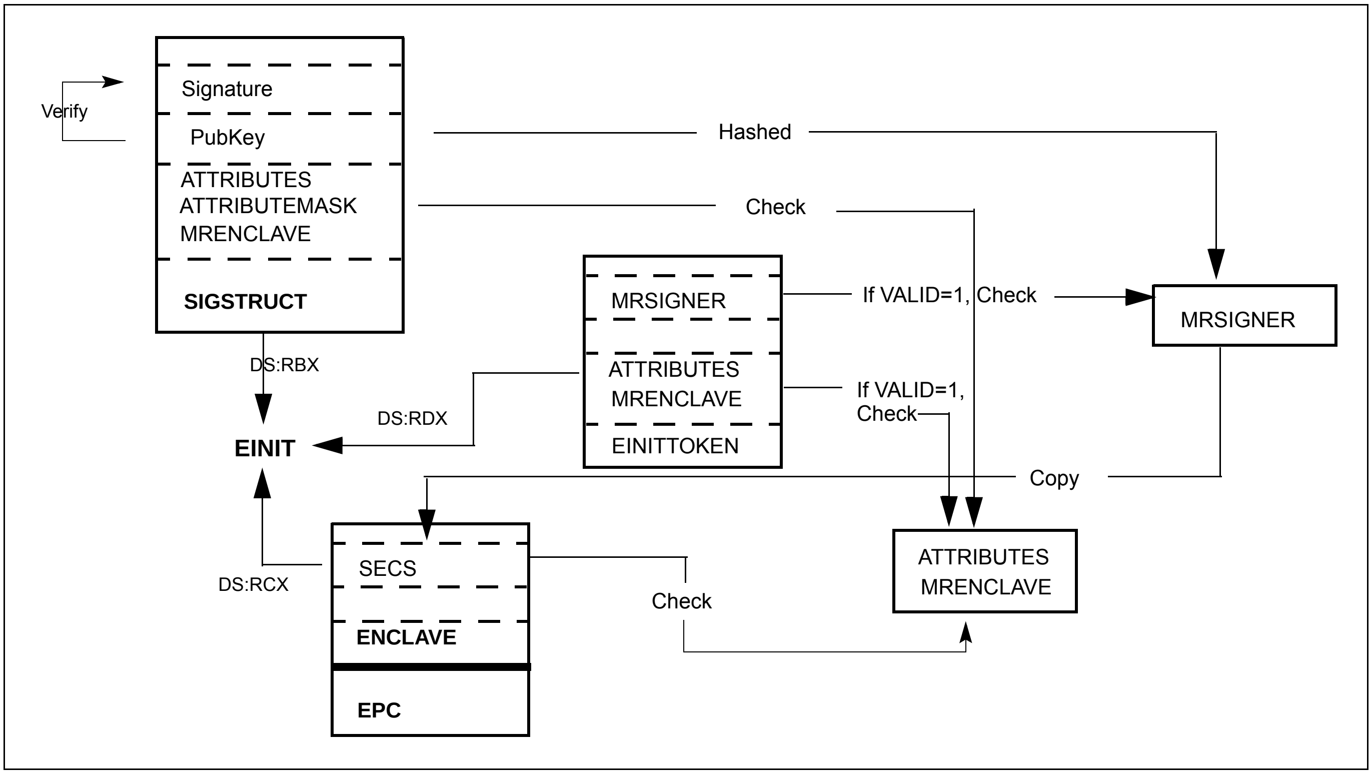


Figure 41-1 Relationships Between SECS, SIGSTRUCT and EINITTOKEN

EINIT Memory Parameter Semantics

SIGSTRUCT	SECS	EINITTOKEN
Access by non-Enclave	Read/Write access by Enclave	Access by non-Enclave

EINIT performs the following steps, which can be seen in Figure 41-1:

Validates that SIGSTRUCT is signed using the enclosed public key.

Checks that the completed computation of SECS.MRENCLAVE equals SIGSTRUCT.HASHENCLAVE.

Checks that no reserved bits are set to 1 in SIGSTRUCT.ATTRIBUTES and no reserved bits in SIGSTRUCT.ATTRIBUTESMASK are set to 0.

Checks that no controlled ATTRIBUTES bits are set in SIGSTRUCT.ATTRIBUTES unless the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.

Checks that SIGSTRUCT.ATTRIBUTES equals the result of logically and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.

If EINITTOKEN.VALID is 0, checks that the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.

If EINITTOKEN.VALID is 1, checks the validity of EINITTOKEN.

If EINITTOKEN.VALID is 1, checks that EINITTOKEN.MRENCLAVE equals SECS.MRENCLAVE.

If EINITTOKEN.VALID is 1 and EINITTOKEN.ATTRIBUTES.DEBUG is 1, SECS.ATTRIBUTES.DEBUG must be 1.

Commits SECS.MRENCLAVE, and sets SECS.MRSIGNER, SECS.ISVSVN, and SECS.ISVPRODID based on SIGSTRUCT.

Update the SECS as Initialized.

Periodically, EINIT polls for certain asynchronous events. If such an event is detected, it completes with failure code (ZF=1 and RAX = SGX_UNMASKED_EVENT), and RIP is incremented to point to the next instruction. These events includes external interrupts, non-maskable interrupts, system-management interrupts, machine checks, INIT signals, and the VMX-preemption timer. EINIT does not fail if the pending event is inhibited (e.g., external interrupts could be inhibited due to blocking by MOV SS blocking or by STI).

The following bits in RFLAGS are cleared: CF, PF, AF, OF, and SF. When the instruction completes with an error, RFLAGS.ZF is set to 1, and the corresponding error bit is set in RAX. If no error occurs, RFLAGS.ZF is cleared and RAX is set to 0.

The error codes are:

Table 41-22 EINIT Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EINIT successful
SGX_INVALID_SIG_STRUCT	If SIGSTRUCT contained an invalid value
SGX_INVALID_ATTRIBUTE	If SIGSTRUCT contains an unauthorized attributes mask
SGX_INVALID_MEASUREMENT	If SIGSTRUCT contains an incorrect measurement If EINITTOKEN contains an incorrect measurement
SGX_INVALID_SIGNATURE	If signature does not validate with enclosed public key
SGX_INVALID_LICENSE	If license is invalid
SGX_INVALID_CPUSVN	If license SVN is unsupported
SGX_UNMASKED_EVENT	If an unmasked event is received before the instruction completes its operation

Concurrency Restrictions

Table 41-23 Concurrency Restrictions of EINIT with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EINIT	SECS			N	N	N	Y	Y	N	N	Y			N	N	N		N	N	N		Y	N

Table 41-24 Concurrency Restrictions of EINIT with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EINIT	SECS	N	Y		N	Y	N		Y	N	N				N	N	N			N			

Operation

Temp Variables in EINIT Operational Flow

Name	Type	Size	Description
TMP_SIG	SIGSTRUCT	1808Bytes	Temp space for SIGSTRUCT.
TMP_TOKEN	EINITTOKEN	304Bytes	Temp space for EINITTOKEN.
TMP_MRENCLAVE		32Bytes	Temp space for calculating MRENCLAVE.
TMP_MRSIGNER		32Bytes	Temp space for calculating MRSIGNER.
CONTROLLED_ATTRIBUTES	ATTRIBUTES	16Bytes	Constant mask of all ATTRIBUTE bits that can only be set for authorized enclaves.
TMP_KEYDEPENDENCIES	Buffer	224Bytes	Temp space for key derivation.
TMP_EINITTOKENKEY		16Bytes	Temp space for the derived EINITTOKEN Key.
TMP_SIG_PADDING	PKCS Padding Buffer	352Bytes	The value of the top 352 bytes from the computation of Signature^3 modulo MRSIGNER.

```
(* make sure SIGSTRUCT and SECS are aligned *)
IF ( (DS:RBX is not 4KByte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* make sure the EINITTOKEN is aligned *)
IF (DS:RDX is not 512Byte Aligned)
    THEN #GP(0); FI;

(* make sure the SECS is inside the EPC *)
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SIG[14463:0] ← DS:RBX[14463:0]; // 1808 bytes
TMP_TOKEN[2423:0] ← DS:RDX[2423:0]; // 304 bytes

(* Verify SIGSTRUCT Header. *)
IF ( (TMP_SIG.HEADER ≠ 06000000E10000000000010000000000h) or
    ((TMP_SIG.VENDOR ≠ 0) and (TMP_SIG.VENDOR ≠ 00008086h) ) or
    (TMP_SIG HEADER2 ≠ 01010000600000006000000001000000h) or
    (TMP_SIG.EXPONENT   ≠ 00000003h) or (Reserved space is not 0’s) )
    THEN
        RFLAGS.ZF ← 1;
        RAX ← SGX_INVALID_SIG_STRUCT;
        GOTO EXIT;
FI;

(* Open “Event Window” Check for Interrupts. Verify signature using embedded public key, q1, and q2. Save upper 352 bytes of the
PKCS1.5 encoded message into the TMP_SIG_PADDING*)
IF (interrupt was pending) {
    RFLAG.ZF ← 1;
```

```
        RAX ← SGX_UNMASKED_EVENT;
        GOTO EXIT;
    FI
    IF (signature failed to verify) {
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_SIGNATURE;
        GOTO EXIT;
    FI;
    (*Close “Event Window” *)

    (* make sure no other Intel SGX instruction is modifying SECS*)
    IF (Other instructions modifying SECS)
        THEN #GP(0); FI;

    IF ( (EPCM(DS:RCX). VALID = 0) or (EPCM(DS:RCX).PT ≠ PT_SECS) )
        THEN #PF(DS:RCX); FI;

    (* make sure no other instruction is accessing MRENCLAVE or ATTRIBUETS.INIT *)
    IF ( (Other instruction modifying MRENCLAVE) or (Other instructions modifying the SECS’s Initialized state))
        THEN #GP(0); FI;

    (* Calculate finalized version of MRENCLAVE *)
    (* SHA256 algorithm requires one last update that compresses the length of the hashed message into the output SHA256 digest *)
    TMP_ENCLAVE ←   SHA256FINAL( (DS:RCX).MRENCLAVE, enclave’s MRENCLAVE update count *512);

    (* Verify MRENCLAVE from SIGSTRUCT *)
    IF (TMP_SIG.ENCLAVEHASH ≠ TMP_MRENCLAVE)
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_MEASUREMENT;
        GOTO EXIT;
    FI;

    TMP_MRSIGNER ← SHA256(TMP_SIG.MODULUS)

    (* if controlled ATTRIBUTES are set, SIGSTRUCT must be signed using an authorized key *)
    CONTROLLED_ATTRIBUTES ← 0000000000000020H;
    IF ( ( (DS:RCX.ATTRIBUTES & CONTROLLED_ATTRIBUTES) ≠ 0) and (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH) )
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_ATTRIBUTE;
        GOTO EXIT;
    FI;

    (* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
    IF ( (DS:RCX.ATTRIBUTES & TMP_SIG.ATTRIBUTEMASK) ≠ (TMP_SIG.ATTRIBUTE & TMP_SIG.ATTRIBUTEMASK) )
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_ATTRIBUTE;
        GOTO EXIT;
    FI;

    ( *Verify SIGSTRUCT.MISCSELECT requirements are met *)
```

```
IF ( (DS:RCX.MISCSELECT & TMP_SIG.MISCMASK) ≠ (TMP_SIG.MISCSELECT & TMP_SIG.MISCMASK) )
    THEN
        RFLAGS.ZF ← 1;
        RAX ← SGX_INVALID_ATTRIBUTE;
    GOTO EXIT
FI;

(* if EINITTOKEN.VALID[0] is 0, verify the enclave is signed by an authorized key *)
IF (TMP_TOKEN.VALID[0] = 0)
    IF (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH)
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_EINITTOKEN;
        GOTO EXIT;
    FI;
    GOTO COMMIT;
FI;

(* Debug Launch Enclave cannot launch Production Enclaves *)
IF ( (DS:RDX.MASKEDATTRIBUTESLE.DEBUG = 1) and (DS:RCX.ATTRIBUTES.DEBUG = 0) )
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;

(* Check reserve space in EINIT token includes reserved regions and upper bits in valid field *)
IF (TMP_TOKEN reserved space is not clear)
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;

(* EINIT token must be ≤ CR_CPUSVN *)
IF (TMP_TOKEN.CPUSVN > CR_CPUSVN)
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_CPUSVN;
    GOTO EXIT;
FI;

(* Derive Launch key used to calculate EINITTOKEN.MAC *)
HARDCODED_PKCS1_5_PADDING[15:0] ← 0100H;
HARDCODED_PKCS1_5_PADDING[2655:16] ← SignExtend330Byte(-1); // 330 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[2815:2656] ← 2004000501020403650148866009060D30313000H;

TMP_KEYDEPENDENCIES.KEYNAME ← EINITTOKEN_KEY;
TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_TOKEN.ISVPRODIDLE;
TMP_KEYDEPENDENCIES.ISVSVN ← TMP_TOKEN.ISVSVN;
TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_TOKEN.MASKEDATTRIBUTESLE;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
TMP_KEYDEPENDENCIES.MRSIGNER ← IA32_SGXLEPUBKEYHASH;
TMP_KEYDEPENDENCIES.KEYID ← TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← TMP_TOKEN.CPUSVN;
TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_TOKEN.MASKEDMISCSELECTLE;
TMP_KEYDEPENDENCIES.MISCMASK ← 0;
TMP_KEYDEPENDENCIES.PADDING ← HARDCODED_PKCS1_5_PADDING;

(* Calculate the derived key*)
TMP_EINITTOKENKEY ← derivekey(TMP_KEYDEPENDENCIES);

(* Verify EINITTOKEN was generated using this CPU's Launch key and that it has not been modified since issuing by the Launch
Enclave. Only 192 bytes of EINITOKEN are CMACed *)
IF (TMP_TOKEN.MAC ≠ CMAC(TMP_EINITTOKENKEY, TMP_TOKEN[1535:0] ) )
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_EINIT_TOKEN;
    GOTO EXIT;
FI;

(* Verify EINITTOKEN (RDX) is for this enclave *)
IF (TMP_TOKEN.MRENCLAVE ≠ TMP_MRENCLAVE) or (TMP_TOKEN.MRSIGNER ≠ TMP_MRSIGNER) )
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_MEASUREMENT;
    GOTO EXIT;
FI;

(* Verify ATTRIBUTES in EINITTOKEN are the same as the enclave's *)
IF (TMP_TOKEN.ATTRIBUTES ≠ DS:RCX.ATTRIBUTES)
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_EINIT_ATTRIBUTE;
    GOTO EXIT;
FI;

COMMIT:
(* Commit changes to the SECS; Set ISVPRODID, ISVSVN, MRSIGNER, INIT ATTRIBUTE fields in SECS (RCX) *)
DS:RCX.MRENCLAVE ← TMP_MRENCLAVE;
(* MRSIGNER stores a SHA256 in little endian implemented natively on x86 *)
DS:RCX.MRSIGNER ← TMP_MRSIGNER;
DS:RCX.ISVPRODID ← TMP_SIG.ISVPRODID;
DS:RCX.ISVSVN ← TMP_SIG.ISVSVN;
DS:RCX.PADDING ← TMP_SIG_PADDING;

(* Mark the SECS as initialized *)
Update DS:RCX to initialized;

(* Set RAX and ZF for success*)
    RFLAG.ZF ← 0;
    RAX ← 0;
EXIT:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set and RAX contains the error code. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand is not properly aligned. If another instruction is modifying the SECS. If the enclave is already initialized. If the SECS.MRENCLAVE is in use.
#PF(error code)	If a page fault occurs in accessing memory operands. If RCX does not resolve in an EPC page. If the memory address is not a valid, uninitialized SECS.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is not properly aligned. If another instruction is modifying the SECS. If the enclave is already initialized. If the SECS.MRENCLAVE is in use.
#PF(error code)	If a page fault occurs in accessing memory operands. If RCX does not resolve in an EPC page. If the memory address is not a valid, uninitialized SECS.

...

ELDB/ELDU—Load an EPC page and Marked its State

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLS[ELDB]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as blocked.
EAX = 08H ENCLS[ELDU]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as unblocked.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	ELDB/ELDU (In)	Return error code (Out)	Address of the PAGEINFO (In)	Address of the EPC page (In)	Address of the version-array slot (In)

Description

This leaf function copies a page from regular main memory to the EPC. As part of the copying process, the page is cryptographically authenticated and decrypted. This instruction can only be executed when current privilege level is 0.

The ELDB leaf function sets the BLOCK bit in the EPCM entry for the destination page in the EPC after copying. The ELDU leaf function clears the BLOCK bit in the EPCM entry for the destination page in the EPC after copying.

RBX contains the effective address of a PAGEINFO structure; RCX contains the effective address of the destination EPC page; RDX holds the effective address of the version array slot that holds the version of the page.

The table below provides additional information on the memory parameter of ELDB/ELDU leaf functions.

EBLDB/ELDBU Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	PAGEINFO.SECS	EPCPAGE	Version-Array Slot
Non-enclave read access	Non-enclave read access	Non-enclave read access	Enclave read/write access	Read/Write access permitted by Enclave	Read/Write access permitted by Enclave

The error codes are:

Table 41-25 ELDB/ELDU Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	ELDB/ELDU successful
SGX_MAC_COMPARE_FAIL	If the MAC check fails

Concurrency Restrictions

Table 41-26 Concurrency Restrictions of ELDB/ELDU with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECRE ATE	EDBGRD/ WR		EENTER/ ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
ELDB/ ELDU	Targ				N		N		N	N		N			N				N	N	N	N	N
	VA				N				N	Y										N	Y		N
	SECS			Y	N	Y		Y	N		Y			Y		Y		Y	Y	N		Y	

Table 41-27 Concurrency Restrictions of ELDB/ELDU with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRA CK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECI NFO	Targ	SECS	Targ	SECS	Targ	SECI NFO	SECS	Targ	SRC	SECI NFO
ELDB/ ELDU	Targ	N				N	N	N		N	N			N		N							
	VA	N					N	Y		N						N							
	SECS	N	Y		Y	Y			Y	N	Y				Y		Y			Y			

Operation

Temp Variables in ELDB/ELDU Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Memory page	4KBytes	
TMP_SECS	Memory page	4KBytes	
TMP_PCMD	PCMD	128 Bytes	
TMP_HEADER	MACHEADER	128 Bytes	
TMP_VER	UINT64	64	
TMP_MAC	UINT128	128	
TMP_PK	UINT128	128	Page encryption/MAC key.
SCRATCH_PCMD	PCMD	128 Bytes	

```
(* Check PAGEINFO and EPCPAGE alignment *)
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check VASLOT alignment *)
IF (DS:RDX is not 8Byte aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

TMP_SRCPGE ← DS:RBX.SRCPGE;
TMP_SECS ← DS:RBX.SECS;
TMP_PCMD ← DS:RBXPCMD;

(* Check alignment of PAGEINFO (RBX)linked parameters. Note: PCMD pointer is overlaid on top of PAGEINFO.SECINFO field *)
IF ( (DS:TMP_PCMD is not 128Byte aligned) or (DS:TMP_SRCPGE is not 4KByte aligned) )
    THEN #GP(0); FI;

(* Check concurrency of EPC and VASLOT by other Intel SGX instructions *)
IF ( (other instructions accessing EPC) or (Other instructions modifying VA slot) )
    THEN #GP(0); FI;

(* Verify EPCM attributes of EPC page, VA, and SECS *)
IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~0FFFH).PT ≠ PT_VA) )
    THEN #PF(DS:RDX); FI;
```

```
(* Copy PCMD into scratch buffer *)
SCRATCH_PCMD[1023: 0]← DS:TMP_PCMD[1023:0];

(* Zero out TMP_HEADER*)
TMP_HEADER[sizeof(TMP_HEADER)-1: 0]← 0;

TMP_HEADER.SECINFO ← SCRATCH_PCMD.SECINFO;
TMP_HEADER.RSVD ← SCRATCH_PCMD.RSVD;
TMP_HEADER.LINADDR ← DS:RBX.LINADDR;

(* Verify various attributes of SECS parameter *)
IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) )
    THEN
        IF ( DS:TMP_SECS is not 4KByte aligned)
            THEN #GP(0) FI;
        IF (DS:TMP_SECS does not resolve within an EPC)
            THEN #PF(DS:TMP_SECS) FI;
        IF ( Other instructions modifying SECS)
            THEN #GP(0) FI;
        IF ( (EPCM(DS:TMP_SECS).VALID = 0) or (EPCM(DS:TMP_SECS).PT ≠ PT_SECS) )
            THEN #PF(DS:TMP_SECS) FI;
    ELSIF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_SECS) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_VA) )
        IF ( ( TMP_SECS ≠ 0) )
            THEN #GP(0) FI;
    ELSE
            #GP(0)
FI;

IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) )
    THEN
        TMP_HEADER.EID ← DS:TMP_SECS.EID;
    ELSE
        (* These pages do not have any parent, and hence no EID binding *)
        TMP_HEADER.EID ← 0;
FI;

(* Copy 4KBytes SRCPGE to secure location *)
DS:RCX[32767: 0]← DS:TMP_SRCPGE[32767: 0];
TMP_VER ← DS:RDX[63:0];

(* Decrypt and MAC page. AES_GCM_DEC has 2 outputs, {plain text, MAC} *)
(* Parameters for AES_GCM_DEC {Key, Counter, ..} *)
{DS:RCX, TMP_MAC} ← AES_GCM_DEC(CR_BASE_PK, TMP_VER << 32, TMP_HEADER, 128, DS:RCX, 4096);

IF ( (TMP_MAC ≠ DS:TMP_PCMD.MAC) )
    THEN
        RFLAGS.ZF ← 1;
        RAX← SGX_MAC_COMPARE_FAIL;
        GOTO ERROR_EXIT;
FI;

(* Check version before committing *)
IF (DS:RDX ≠ 0)
    THEN #GP(0);
    ELSE
        DS:RDX← TMP_VER;
FI;

(* Commit EPCM changes *)
EPCM(DS:RCX).PT ← TMP_HEADER.SECINFO.FLAGS.PT;
EPCM(DS:RCX).RWX ← TMP_HEADER.SECINFO.FLAGS.RWX;
EPCM(DS:RCX).PENDING ← TMP_HEADER.SECINFO.FLAGS.PENDING;
EPCM(DS:RCX).MODIFIED ← TMP_HEADER.SECINFO.FLAGS.MODIFIED;
EPCM(DS:RCX).PR ← TMP_HEADER.SECINFO.FLAGS.PR;
EPCM(DS:RCX).ENCLAVEADDRESS ← TMP_HEADER.LINADDR;

IF ( (EAX = 07H) and (TMP_HEADER.SECINFO.FLAGS.PT is NOT PT_SECS or PT_VA))
    THEN
        EPCM(DS:RCX).BLOCKED ← 1;
    ELSE
        EPCM(DS:RCX).BLOCKED ← 0;
FI;

EPCM(DS:RCX). VALID ← 1;

RAX← 0;
RFLAGS.ZF ← 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If the instruction's EPC resource is in use by others.
If the instruction fails to verify MAC.
If the version-array slot is in use.
If the parameters fail consistency checks.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand expected to be in EPC does not resolve to an EPC page.
If one of the EPC memory operands has incorrect page type.
If the destination EPC page is already valid.

64-Bit Mode Exceptions

#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If the instruction's EPC resource is in use by others.
If the instruction fails to verify MAC.
If the version-array slot is in use.
If the parameters fail consistency checks.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand expected to be in EPC does not resolve to an EPC page.
If one of the EPC memory operands has incorrect page type.
If the destination EPC page is already valid.

...

EMODPR—Restrict the Permissions of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0EH ENCLS[EMODPR]	IR	V/V	SGX2	This leaf function restricts the access rights associated with a EPC page in an initialized enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODPR (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function restricts the access rights associated with an EPC page in an initialized enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not restrict the page permissions will have no effect. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPR leaf function.

EMODPR Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODPR Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 41-28 EMODPR Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EMODPR successful
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state
SGX_LOCKFAIL	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB

Concurrency Restrictions

Table 41-29 Concurrency Restrictions of EMODPR with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EMODPR	Targ		Y		N		Y		N	Y			Y		N		Y			N			N
	SECS			Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y

Table 41-30 Concurrency Restrictions of EMODPR with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EMODPR	Targ	N		Y	Y		N			N		C	Y	C		C		C	Y		C	Y	Y
	SECS	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EMODPR Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or
    !(SCRATCH_SECINFO.FLAGS.R is 0 or SCRATCH_SECINFO.FLAGS.W is not 0) )
    THEN #GP(0); FI;

(* Check concurrency with SGX1 or SGX2 instructions on the EPC page *)
IF (SGX1 or other SGX2 instructions accessing EPC page)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID is 0 )
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN
        RFLAGS ← 1;
        RAX ← SGX_LOCKFAIL;
        GOTO DONE;
FI;

IF ( (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS ← 1;
        RAX ← SGX_PAGE_NOT_MODIFIABLE;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT is not PT_REG)
    THEN #PF(DS:RCX); FI;

TMP_SECS ← GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
  THEN #GP(0); FI;

(* Set the PR bit to indicate that permission restriction is in progress *)
EPCM(DS:RCX).PR ← 1;

(* Update EPCM permissions *)
EPCM(DS:RCX).R ← EPCM(DS:RCX).R & SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W ← EPCM(DS:RCX).W & SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X ← EPCM(DS:RCX).X & SCRATCH_SECINFO.FLAGS.X;

RFLAGS.ZF ← 0;
RAX ← 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

...

EMODT—Change the Type of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0FH ENCLS[EMODT]	IR	V/V	SGX2	This leaf function changes the type of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function modifies the type of an EPC page. The security attributes are configured to prevent access to the EPC page at its new type until a corresponding invocation of the EACCEPT leaf confirms the modification. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODT leaf function.

EMODT Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODT Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 41-31 EMODT Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EMODT successful
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state
SGX_LOCKFAIL	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODPR, or EWB

Concurrency Restrictions

Table 41-32 Concurrency Restrictions of EMODT with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EMODT	Targ	Y	Y		C	C	C		C	C		C	Y		C		Y		N	C	C		C
	SECS			Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y

Table 41-33 Concurrency Restrictions of EMODT with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EMODT	Targ	C		Y			C	C		C	C	C	Y	C		C	Y	C	Y		C	Y	Y
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EMODT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;
```

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or
    !(SCRATCH_SECINFO.FLAGS.PT is PT_TCS or SCRATCH_SECINFO.FLAGS.PT is PT_TRIM) )
    THEN #GP(0); FI;

(* Check concurrency with SGX1 instructions on the EPC page *)
IF (other SGX1 instructions accessing EPC page)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID is 0 or
    !(EPCM(DS:RCX).PT is PT_REG or EPCM(DS:RCX).PT is PT_TCS))
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN #GP(0); FI;

(* Check for mis-configured SECINFO flags*)
IF ( (EPCM(DS:RCX).R = 0) and (SCRATCH_SECINFO.FLAGS.R = 0) and (SCRATCH_SECINFO.FLAGS.W ≠ 0) ))
    THEN
        RFLAGS ← 1;
        RAX ← SGX_LOCKFAIL;
        GOTO DONE;
FI;

IF ( (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS ← 1;
        RAX ← SGX_PAGE_NOT_MODIFIABLE;
        GOTO DONE;
FI;

TMP_SECS ← GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
  THEN #GP(0); FI;

(* Check concurrency with ETRACK *)
IF (ETRACK executed concurrently)
    THEN #GP(0); FI;

(* Update EPCM fields *)
```

```
EPCM(DS:RCX).PR ← 0;
EPCM(DS:RCX).MODIFIED ← 1;
EPCM(DS:RCX).R ← 0;
EPCM(DS:RCX).W ← 0;
EPCM(DS:RCX).X ← 0;
EPCM(DS:RCX).PT ← SCRATCH_SECINFO.FLAGS.PT;

RFLAGS.ZF ← 0;
RAX ← 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.

...

EPA—Add Version Array

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0AH ENCLS[EPA]	IR	V/V	SGX1	This leaf function adds a Version Array to the EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EPA (In)	PT_VA (In, Constant)	Effective address of the EPC page (In)

Description

This leaf function creates an empty version array in the EPC page whose logical address is given by DS:RCX, and sets up EPCM attributes for that page. At the time of execution of this instruction, the register RBX must be set to PT_VA.

The table below provides additional information on the memory parameter of EPA leaf function.

EPA Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

Concurrency Restrictions

Table 41-34 Concurrency Restrictions of EPA with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EPA	VA				N	N	N		N	N		N			N				N	N	N		N

Table 41-35 Concurrency Restrictions of EPA with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EPA	VA	N				N	N	N		N	N			N		N							

Operation

```
IF (RBX ≠ PT_VA or DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions accessing the page)
    THEN #GP(0); FI;

(* Check EPC page must be empty *)
IF (EPCM(DS:RCX). VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Clears EPC page *)
DS:RCX[32767:0] ← 0;

EPCM(DS:RCX).PT ← PT_VA;
EPCM(DS:RCX).ENCLAVEADDRESS ← 0;
EPCM(DS:RCX).BLOCKED ← 0;
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).PR ← 0;
EPCM(DS:RCX).RWX ← 0;
EPCM(DS:RCX).VALID ← 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If another Intel SGX instruction is accessing the EPC page.
If RBX is not set to PT_VA.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If another Intel SGX instruction is accessing the EPC page.
If RBX is not set to PT_VA.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If the EPC page is valid.

...

EREMOVE—Remove a page from the EPC

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLS[EREMOVE]	IR	V/V	SGX1	This leaf function removes a page from the EPC.

Instruction Operand Encoding

Op/En	EAX	RCX
IR	EREMOVE (In)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be un-associated with its SECS and be marked as unused. This instruction leaf can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

The instruction fails if the operand is not properly aligned or does not refer to an EPC page or the page is in use by another thread, or other threads are running in the enclave to which the page belongs. In addition the instruction fails if the operand refers to an SECS with associations.

EREMOVE Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

The instruction faults if any of the following:

EREMOVE Faulting Conditions

The memory operand is not properly aligned.	The memory operand does not resolve in an EPC page.
Refers to an invalid SECS.	Refers to an EPC page that is locked by another thread.
Another Intel SGX instruction is accessing the EPC page.	RCX does not contain an effective address of an EPC page.
the EPC page refers to an SECS with associations.	

The error codes are:

Table 41-36 EREMOVE Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EREMOVE successful
SGX_CHILD_PRESENT	If the SECS still have enclave pages loaded into EPC
SGX_ENCLAVE_ACT	If there are still logical processors executing inside the enclave

Concurrency Restrictions

Table 41-37 Concurrency Restrictions of EREMOVE with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EREMOVE	Targ	C	C	C	N	N	N	C	N	N	C	N	C	C	N	C	C	C	N	N	N	N	N
	SECS			C	Y	Y	Y	Y	Y	Y	Y	Y	Y	C	Y	Y	Y	C	Y	Y	Y	Y	Y

Table 41-38 Concurrency Restrictions of EREMOVE with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EREMOVE	Targ	N	C	C	C	N	N	N	C	N	N	C	C	N	C	N	C	C	C	C	C	C	C
	SECS	Y	Y	Y	C	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	C	C	C	Y	Y	Y

Operation

Temp Variables in EREMOVE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve to an EPC page)
    THEN #PF(DS:RCX); FI;

TMP_SECS ← Get_SECS_ADDRESS();

(* Check the EPC page for concurrency *)
IF (EPC page being referenced by another Intel SGX instruction)
    THEN #GP(0); FI;

(* if DS:RCX is already unused, nothing to do*)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PT = PT_TRIM AND EPCM(DS:RCX).MODIFIED = 0))
    THEN GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT = PT_VA)
    THEN
        EPCM(DS:RCX).VALID ← 0;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT = PT_SECS)
    THEN
        IF (DS:RCX has an EPC page associated with it)
            THEN
                RFLAGS.ZF ← 1;
                RAX← SGX_CHILD_PRESENT;
                GOTO ERROR_EXIT;
        FI;
        EPCM(DS:RCX).VALID ← 0;
        GOTO DONE;
FI;

TEMP_SECS ← Get_SECS_ADDRESS();

IF (Other threads active using SECS)
    THEN
        RFLAGS.ZF ← 1;
        RAX← SGX_ENCLAVE_ACT;
        GOTO ERROR_EXIT;
```

```
FI;

DONE:
RAX← 0;
RFLAGS.ZF ← 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the page.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If the memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the page.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the memory operand is not an EPC page.

...

ETRACK—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0CH ENCLS[ETRACK]	IR	V/V	SGX1	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	ETRACK (In)	Return error code (Out)	Pointer to the SECS of the EPC page (In)

Description

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

The table below provides additional information on the memory parameter of EBLOCK leaf function.

ETRACK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 41-39 ETRACK Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	ETRACK successful
SGX_PREV_TRK_INCMPL	All processors did not complete the previous shoot-down sequence

Concurrency Restrictions

Table 41-40 Concurrency Restrictions of ETRACK with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
ETRACK	SECS			Y	N	Y		Y	N	N	Y			Y		Y		Y	Y	N		Y	N

Table 41-41 Concurrency Restrictions of ETRACK with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
ETRACK	SECS	N	Y		Y	N	N		Y	N	Y				Y		Y			Y			

Operation

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions using tracking facility on this SECS)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).PT ≠ PT_SECS)
    THEN #PF(DS:RCX); FI;

(* All processors must have completed the previous tracking cycle*)
IF ( (DS:RCX).TRACKING ≠ 0) )
    THEN
        RFLAGS.ZF ← 1;
        RAX← SGX_PREV_TRK_INCMPL;
```

```
            GOTO DONE;
        ELSE
            RAX← 0;
            RFLAGS.ZF ← 0;
FI;

DONE:
RFLAGS.ZF,CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If another thread is concurrently using the tracking facility on this SECS.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form. If a memory operand is not properly aligned. If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands. If a memory operand is not an EPC page.

Table 41-42 ETRACK Return Value in RAX

Error Code (see Table 38-3)	Value	Description
No Error	0	ETRACK successful
SGX_PREV_TRK_INCMPL		All processors did not complete the previous shoot-down sequence

...

EWB—Invalidate an EPC Page and Write out to Main Memory

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0BH ENCLS[EWB]	IR	V/V	SGX1	This leaf function invalidates an EPC page and writes it out to main memory.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EWB (In)	Error code (Out)	Address of an PAGEINFO (In)	Address of the EPC page (In)	Address of a VA slot (In)

Description

This leaf function copies a page from the EPC to regular main memory. As part of the copying process, the page is cryptographically protected. This instruction can only be executed when current privilege level is 0.

The table below provides additional information on the memory parameter of EPA leaf function.

EWB Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	EPCPAGE	VASLOT
Non-EPC R/W access	Non-EPC R/W access	Non-EPC R/W access	EPC R/W access	EPC R/W access

The error codes are:

Table 41-43 EWB Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EWB successful
SGX_PAGE_NOT_BLOCKED	If page is not marked as blocked
SGX_NOT_TRACKED	If EWB is racing with ETRACK instruction
SGX_VA_SLOT_OCCUPIED	Version array slot contained valid entry
SGX_CHILD_PRESENT	Child page present while attempting to page out enclave

Concurrency Restrictions

Table 41-44 Concurrency Restrictions of EWB with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EWB	Src	C	C	C	N	N	N	C	N	N	C	N	C	C	N	C	C	C	N	N	N		N
	VA				N				N	Y										N	Y		N
	SECS			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-45 Concurrency Restrictions of EWB with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EWB	Src	N	C	C	C	N	N	N	C	N	N	C	C	N	C	N	C	C	C	C	C	C	C
	VA	N					N	Y		N						N							
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EWB Operational Flow

Name	Type	Size (Bytes)	Description
TMP_SRCPGE	Memory page	4096	
TMP_PCMD	PCMD	128	
TMP_SECS	SECS	4096	
TMP_BPEPOCH	UINT64	8	
TMP_BPREFCOUNT	UINT64	8	
TMP_HEADER	MAC Header	128	
TMP_PCMD_ENCLAVEID	UINT64	8	
TMP_VER	UINT64	8	
TMP_PK	UINT128	16	

```
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (DS:RDX is not 8Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

(* EPCPAGE and VASLOT should not resolve to the same EPC page*)
IF (DS:RCX and DS:RDX resolve to the same EPC page)
    THEN #GP(0); FI;

TMP_SRCPGE ← DS:RBX.SRCPGE;
(* Note PAGEINFO.PCMD is overlaid on top of PAGEINFO.SECINFO *)
TMP_PCMD ← DS:RBX.PCMD;

If (DS:RBX.LINADDR ≠ 0) OR (DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

IF ( (DS:TMP_PCMD is not 128Byte Aligned) or (DSTMP_SRCPGE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Check for concurrent Intel SGX instruction access to the page *)
IF (Other Intel SGX instruction is accessing page)
    THEN #GP(0); FI;

(*Check if the VA Page is being removed or changed*)
```

```
IF (VA Page is being modified)
    THEN #GP(0); FI;

(* Verify that EPCPAGE and VASLOT page are valid EPC pages and DS:RDX is VA *)
IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~FFFH).PT is not PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Perform page-type-specific exception checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM ) )
    THEN
        TMP_SECS = Obtain SECS through EPCM(DS:RCX)
    (* Check that EBLOCK has occurred correctly *)
    IF (EBLOCK is not correct)
        THEN #GP(0); FI;
FI;

RFLAGS.ZF,CF,PF,AF,OF,SF ← 0;
RAX ← 0;

(* Perform page-type-specific checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM ))
    THEN
        (* check to see if the page is evictable *)
        IF (EPCM(DS:RCX).BLOCKED = 0)
            THEN
                RAX ← SGX_PAGE NOT_BLOCKED;
                RFLAGS.ZF ← 1;
                GOTO ERROR_EXIT;
        FI;
        (* Check if tracking done correctly *)
        IF (Tracking not correct)
            THEN
                RAX ← SGX_NOT_TRACKED;
                RFLAGS.ZF ← 1;
                GOTO ERROR_EXIT;
        FI;

        (* Obtain EID to establish cryptographic binding between the paged-out page and the enclave *)
        TMP_HEADER.EID ← TMP_SECS.EID;

        (* Obtain EID as an enclave handle for software *)
        TMP_PCMD_ENCLAVEID ← TMP_SECS.EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_SECS)
        (*check that there are no child pages inside the enclave *)
        IF (DS:RCX has an EPC page associated with it)
            THEN
                RAX ← SGX_CHILD_PRESENT;
```

```
                    RFLAGS.ZF ← 1;
                    GOTO ERROR_EXIT;
            FI:
            TMP_HEADER.EID ← 0;
            (* Obtain EID as an enclave handle for software *)
            TMP_PCMD_ENCLAVEID ← (DS:RCX).EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_VA)
            TMP_HEADER.EID ← 0; // Zero is not a special value
            (* No enclave handle for VA pages*)
            TMP_PCMD_ENCLAVEID ← 0;
FI;

(* Zero out TMP_HEADER*)
TMP_HEADER[ sizeof(TMP_HEADER)-1 : 0] ← 0;

TMP_HEADER.LINADDR ← EPCM(DS:RCX).ENCLAVEADDRESS;
TMP_HEADER.SECINFO.FLAGS.PT ← EPCM(DS:RCX).PT;
TMP_HEADER.SECINFO.FLAGS.RWX ← EPCM(DS:RCX).RWX;
TMP_HEADER.SECINFO.FLAGS.PENDING ← EPCM(DS:RCX).PENDING;
TMP_HEADER.SECINFO.FLAGS.MODIFIED ← EPCM(DS:RCX).MODIFIED;
TMP_HEADER.SECINFO.FLAGS.PR ← EPCM(DS:RCX).PR;

(* Encrypt the page, DS:RCX could be encrypted in place. AES-GCM produces 2 values, {ciphertext, MAC}. *)
(* AES-GCM input parameters: key, GCM Counter, MAC_HDR, MAC_HDR_SIZE, SRC, SRC_SIZE)*)
{DS:TMP_SRCPGE, DS:TMP_PCMD.MAC} ← AES_GCM_ENC(CR_BASE_PK), (TMP_VER << 32),
    TMP_HEADER, 128, DS:RCX, 4096);

(* Write the output *)
Zero out DS:TMP_PCMD.SECINFO
DS:TMP_PCMD.SECINFO.FLAGS.PT ← EPCM(DS:RCX).PT;
DS:TMP_PCMD.SECINFO.FLAGS.RWX ← EPCM(DS:RCX).RWX;
DS:TMP_PCMD.SECINFO.FLAGS.PENDING ← EPCM(DS:RCX).PENDING;
DS:TMP_PCMD.SECINFO.FLAGS.MODIFIED ← EPCM(DS:RCX).MODIFIED;
DS:TMP_PCMD.SECINFO.FLAGS.PR ← EPCM(DS:RCX).PR;
DS:TMP_PCMD.RESERVED ← 0;
DS:TMP_PCMD.ENCLAVEID ← TMP_PCMD_ENCLAVEID;
DS:RBX.LINADDR ← EPCM(DS:RCX).ENCLAVEADDRESS;

(*Check if version array slot was empty *)
IF ([DS.RDX])
    THEN
            RAX ← SGX_VA_SLOT_OCCUPIED
            RFLAGS.CF ← 1;
FI;

(* Write version to Version Array slot *)
[DS.RDX] ← TMP_VER;

(* Free up EPCM Entry *)
EPCM.(DS:RCX).VALID ← 0;
```

EXIT:

Flags Affected

ZF is set if page is not blocked, not tracked, or a child is present. Otherwise cleared.

CF is set if VA slot is previously occupied, Otherwise cleared.

Protected Mode Exceptions

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

If a memory operand is not properly aligned.

If the EPC page and VASLOT resolve to the same EPC page.

If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.

If the tracking resource is in use.

If the EPC page or the version array page is invalid.

If the parameters fail consistency checks.

#PF(error code) If a page fault occurs in accessing memory operands.

If a memory operand is not an EPC page.

If one of the EPC memory operands has incorrect page type.

64-Bit Mode Exceptions

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

If a memory operand is not properly aligned.

If the EPC page and VASLOT resolve to the same EPC page.

If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.

If the tracking resource is in use.

If the EPC page or the version array page in invalid.

If the parameters fail consistency checks.

#PF(error code) If a page fault occurs in accessing memory operands.

If a memory operand is not an EPC page.

If one of the EPC memory operands has incorrect page type.

...

41.4 INTEL® SGX USER LEAF FUNCTION REFERENCE

41.4.1 Instruction Column in the Instruction Summary Table

Leaf functions available with the ENCLU instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of the implicitly-encoded register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EACCEPT—Accept Changes to an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLU[EACCEPT]	IR	V/V	SGX2	This leaf function accepts changes made by system software to an EPC page in the running enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EACCEPT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function accepts changes to a page in the running enclave by verifying that the security attributes specified in the SECINFO match the security attributes of the page in the EPCM. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPT leaf function.

EACCEPT Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPT Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is not valid.	RCX does not contain an effective address of an EPC page in the running enclave.
SECINFO contains an invalid request.	Page type is PT_REG and MODIFIED bit is 0.
	Page type is PT_TCS or PT_TRIM and PENDING bit is 0 and MODIFIED bit is 1.

The error codes are:

Table 41-46 EACCEPT Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EACCEPT successful
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values
SGX_NOT_TRACKED	The OS did not complete an ETRACK on the target page

Concurrency Restrictions

Table 41-47 Concurrency Restrictions of EACCEPT with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EACCEPT	Targ	C	Y							Y		C	Y				Y						
	SECINFO		U							Y			U				U						
	SECS			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-48 Concurrency Restrictions of EACCEPT with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EACCEPT	Targ			Y								N	Y	N		N		N	Y		N	Y	C
	SECINFO			U								Y	Y					Y	Y			U	Y
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EACCEPT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operands belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX is not within CR_ELRANGE)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
    (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or
    (EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ (DS:RBX & FFFH)) )
    THEN #PF(DS:RBX); FI;

(* Copy 64 bytes of contents *)
SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero ) )
```

```
        THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not within CR_ELRANGE)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check that the combination of requested PT, PENDING and MODIFIED is legal *)
IF (NOT ( ((SCRATCH_SECINFO.FLAGS.PT is PT_REG) and (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) or
    ((SCRATCH_SECINFO.FLAGS.PT is PT_TCS or PT_TRIM) and (SCRATCH_SECINFO.FLAGS.PENDING is 0) and
    (SCRATCH_SECINFO.FLAGS.MODIFIED is 1)) ) )
THEN #GP(0); FI

(* Check security attributes of the destination EPC page *)
If ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).BLOCKED is not 0) or
    ((EPCM(DS:RCX).PT is not PT_REG) and (EPCM(DS:RCX).PT is not PT_TCS) and (EPCM(DS:RCX).PT is not PT_TRIM)) or
    (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS))
    THEN #PF((DS:RCX); FI;

(* Check the destination EPC page for concurrency *)
IF ( EPC page in use )
    THEN #GP(0); FI;

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN #PF(DS:RCX); FI;

(* Verify that accept request matches current EPC page settings *)
IF ( (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX) or (EPCM(DS:RCX).PENDING ≠ SCRATCH_SECINFO.FLAGS.PENDING) or
    (EPCM(DS:RCX).MODIFIED ≠ SCRATCH_SECINFO.FLAGS.MODIFIED) or (EPCM(DS:RCX).R ≠ SCRATCH_SECINFO.FLAGS.R) or
    (EPCM(DS:RCX).W ≠ SCRATCH_SECINFO.FLAGS.W) or (EPCM(DS:RCX).X ≠ SCRATCH_SECINFO.FLAGS.X) or
    (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) )
    THEN
        RFLAGS ← 1;
        RAX ← SGX_PAGE_ATTRIBUTES_MISMATCH;
        GOTO DONE;
FI;
(* Check that all required threads have left enclave *)
IF (Tracking not correct)
    THEN
        RFLAGS.ZF ← 1;
        RAX ← SGX_NOT_TRACKED;
        GOTO DONE;
FI;

(* Get pointer to the SECS to which the EPC page belongs *)
```

```
TMP_SECS = << Obtain physical address of SECS through EPCM(DS:RCX)>>
(* For TCS pages, perform additional checks *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
FI;

(* Check that TCS.FLAGS.DBGOPTIN, TCS stack, and TCS status are correctly initialized *)
IF ( ((DS:RCX).FLAGS.DBGOPTIN is not 0) or ((DS:RCX).CSSA ≥ (DS:RCX).NSSA) or ((DS:RCX).AEP is not 0) or ((DS:RCX).STATE is not 0)
    THEN #GP(0); FI;

(* Check consistency of FS & GS Limit *)
IF ( (TMP_SECS.ATTRIBUTES.MODE64BIT is 0) and ((DS:RCX.FSLIMIT & FFFH ≠ FFFH) or (DS:RCX.GSLIMIT & FFFH ≠ FFFH)) )
    THEN #GP(0); FI;

(* Clear PENDING/MODIFIED flags to mark accept operation complete *)
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).PR ← 0;

(* Clear EAX and ZF to indicate successful completion *)
RFLAGS.ZF ← 0;
RAX ← 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if page cannot be accepted, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

...

EACCEPTCOPY—Initialize a Pending Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLU[EACCEPTCOPY]	IR	V/V	SGX2	This leaf function initializes a dynamically allocated EPC page from another page in the EPC.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EACCEPTCOPY (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)	Address of the source EPC page (In)

Description

This leaf function copies the contents of an existing EPC page into an uninitialized EPC page (created by EAUG). After initialization, the instruction may also modify the access rights associated with the destination EPC page. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX and RDX each contain the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPTCOPY leaf function.

EACCEPTCOPY Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)	EPCPAGE (Source)
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPTCOPY Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	If security attributes of the source EPC page make the page inaccessible.
The EPC page is not valid.	RBX does not contain an effective address in an EPC page in the running enclave.
SECINFO contains an invalid request.	RCX/RDX does not contain an effective address of an EPC page in the running enclave.

The error codes are:

Table 41-49 EACCEPTCOPY Return Value in RAX

Error Code (see Table 38-3)	Description
No Error	EACCEPTCOPY successful
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values

Concurrency Restrictions

Table 41-50 Concurrency Restrictions of EACCEPTCOPY with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EACCEPTCOPY	Targ																						
	Src		U							Y			U				Y						
	SECINFO		U							Y			U				U						

Table 41-51 Concurrency Restrictions of EACCEPTCOPY with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EACCEPTCOPY	Targ													N		N		N			N		
	Src			Y								Y	Y					Y	U			Y	Y
	SECINFO			U								Y	Y					Y	Y			Y	Y

Operation

Temp Variables in EACCEPTCOPY Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF ( (DS:RCX is not 4KByte Aligned) or (DS:RDX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) or (DS:RDX is not within CR_ELRANGE))
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

IF ( (EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
    (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or (EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or
    (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ DS:RBX) )
    THEN #PF(DS:RBX); FI;

(* Copy 64 bytes of contents *)
SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or ((SCRATCH_SECINFO.FLAGS.R=0) AND(SCRATCH_SECINFO.FLAGS.W≠0 ) or
    (SCRATCH_SECINFO.FLAGS.PT is not PT_REG) )
    THEN #GP(0); FI;

(* Check security attributes of the source EPC page *)
IF ( (EPCM(DS:RDX).VALID = 0) or (EPCM(DS:RDX).PENDING ≠ 0) or (EPCM(DS:RDX).MODIFIED ≠ 0) or
    (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RDX).ENCLAVEADDRESS ≠ DS:RDX))
    THEN #PF(DS:RDX); FI;

(* Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN
         RFLAGS ← 1;
         RAX ← SGX_PAGE_ATTRIBUTE_MISMATCH;
         GOTO DONE;
FI;

(* Check the destination EPC page for concurrency *)
IF (destination EPC page in use )
    THEN #GP(0); FI;

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).R ≠ 1) or (EPCM(DS:RCX).W ≠ 1) or (EPCM(DS:RCX).X ≠ 0) or
    (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
    THEN #PF(DS:RCX); FI;

(* Copy 4KBbytes form the source to destination EPC page*)
DS:RCX[32767:0] ← DS:RDX[32767:0];

(* Update EPCM permissions *)
EPCM(DS:RCX).R ← EPCM(DS:RCX).R | SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W ← EPCM(DS:RCX).W | SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X ← EPCM(DS:RCX).X | SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PENDING ← 0;

RFLAGS.ZF ← 0;
RAX ← 0;
```

```
DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if page is not modifiable, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If EPC page has incorrect page type or security attributes.

...

EENTER—Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLU[EENTER]	IR	V/V	SGX1	This leaf function is used to enter an enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	
IR	EENTER (In)	Content of RBX.CSSA (Out)	Address of a TCS (In)	Address of AEP (In)	Address of IP following EENTER (Out)

Description

The ENCLU[EENTER] instruction transfers execution to an enclave. At the end of the instruction, the logical processor is executing in enclave mode at the RIP computed as EnclaveBase + TCS.OENTRY. If the target address is not within the CS segment (32-bit) or is not canonical (64-bit), a #GP(0) results.

EENTER Memory Parameter Semantics

TCS
Enclave access

EENTER is a serializing instruction. The instruction faults if any of the following occurs:

Address in RBX is not properly aligned.	Any TCS.FLAGS's must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use.	Either of TCS-specified FS and GS segment is not a subsets of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

The following operations are performed by EENTER:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or interrupt.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 43.1.2):
  - On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 43.2.5).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 43.2.2).
- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.
- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 43.2.3):
  - All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
  - PEBS is suppressed.
  - AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set
  - If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 41-52 Concurrency Restrictions of EENTER with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EENTER	TCS	N			N				N	Y		N								N			N
	SSA		U							Y		Y	U				U						
	SECS			Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y

Table 41-53 Concurrency Restrictions of EENTER with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EENTER	TCS	N					N			N						N							
	SSA			U								Y	U					Y	U			U	U
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y

Operation

Temp Variables in EENTER Operational Flow

Name	Type	Size (Bits)	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.

```
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)
IF (TMP_MODE64 = 0 and (DS not usable or ( ( DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1) ) ) )
    THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)
IF (TMP_MODE64 = 0)
    THEN
        IF(CS.base ≠ 0 or DS.base ≠ 0) #GP(0); FI;
        IF(ES usable and ES.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.B = 0) #GP(0); FI;
FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (DS:RCX is not canonical) )
    THEN #GP(0); FI;
```

```
(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions is operating on TCS)
    THEN #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) or (EPCM(DS:RBX).PT ≠ PT_TCS) )
    THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS ← Address of SECS for TCS;

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    THEN
        TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_FSLIMIT ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
        TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        TMP_GSLIMIT ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
        (* if FS wrap-around, make sure DS has no holes*)
        IF (TMP_FSLIMIT < TMP_FSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
        (* if GS wrap-around, make sure DS has no holes*)
        IF (TMP_GSLIMIT < TMP_GSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
    ELSE
        TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
```

```
            TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
            IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
                THEN #GP(0); FI;
FI;

(* Ensure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & & FFFFFFFFFFFFFFFEH) ≠ 0)
    THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    THEN #GP(0); FI;

(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
    THEN #GP(0); FI;

IF (CR4.OSFXSR = 0)
    THEN #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    THEN
        IF (TMP_SECS.ATTRIBUES.XFRM ≠ 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUES.XFRM & XCR0) ≠ TMP_SECS.ATTRIBUES.XFRM) THEN #GP(0); FI;
FI;

(* Make sure the SSA contains at least one more frame *)
IF ( (DS:RBX).CSSA ≥ (DS:RBX).NSSA)
    THEN #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA ← (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * (DS:RBX).CSSA;
TMP_XSIZE ← compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;

    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
                (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
                (EPCM(DS:TMP_SECS).R = 0) or (EPCM(DS:TMP_SECS).W = 0) )
                THEN #PF(DS:TMP_SSA_PAGE); FI;
        CR_XSAVE_PAGE_n ← Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR ← TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE -- sizeof(GPRSGX_AREA);
If a fault occurs; release locks, abort and deliver that fault;

IF (DS:TMP_GPR does not resolve to EPC page)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS EPCM(DS:RBX).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
    THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)
    THEN
        IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
FI;

CR_GPR_PA ← Physical_Address (DS: TMP_GPR);

(* Validate TCS.OENTRY *)
TMP_TARGET ← (DS:RBX).OENTRY + TMP_SECS.BASEADDR;
IF (TMP_MODE64 = 1)
    THEN
        IF (TMP_TARGET is not canonical) THEN #GP(0); FI;
    ELSE
        IF (TMP_TARGET > CS limit) THEN #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE))
    THEN #GP(0); FI;

CR_ENCALVE_MODE ← 1;
CR_ACTIVE_SECS ← TMP_SECS;
CR_ELRANGE ← (TMPSECS.BASEADDR, TMP_SECS.SIZE);

(* Save state for possible AEXs *)
CR_TCS_PA ← Physical_Address (DS:RBX);
CR_TCS_LA ← RBX;
```

```
CR_TCS_LA.AEP ← RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector ← FS.selector;
CR_SAVE_FS_base ← FS.base;
CR_SAVE_FS_limit ← FS.limit;
CR_SAVE_FS_access_rights ← FS.access_rights;
CR_SAVE_GS_selector ← GS.selector;
CR_SAVE_GS_base ← GS.base;
CR_SAVE_GS_limit ← GS.limit;
CR_SAVE_GS_access_rights ← GS.access_rights;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 ← XCR0;
    XCR0 ← TMP_SECS.ATTRIBUTES.XFRM;
FI;

(* Set CR_ENCLAVE_ENTRY_IP *)
CR_ENCLAVE_ENTRY_IP ← CRIP”
RIP ← NRIP;
RAX ← (DS:RBX).CSSA;
(* Save the outside RSP and RBP so they can be restored on interrupt or EEXIT *)
DS:TMP_SSA.U_RSP ← RSP;
DS:TMP_SSA.U_RBP ← RBP;

(* Do the FS/GS swap *)
FS.base ← TMP_FSBASE;
FS.limit ← DS:RBX.FSLIMIT;
FS.type ← 0001b;
FS.W ← DS.W;
FS.S ← 1;
FS.DPL ← DS.DPL;
FS.G ← 1;
FS.B ← 1;
FS.P ← 1;
FS.AVL ← DS.AVL;
FS.L ← DS.L;
FS.unusable ← 0;
FS.selector ← 0BH;

GS.base ← TMP_GSBASE;
GS.limit ← DS:RBX.GSLIMIT;
GS.type ← 0001b;
GS.W ← DS.W;
GS.S ← 1;
GS.DPL ← DS.DPL;
GS.G ← 1;
GS.B ← 1;
GS.P ← 1;
GS.AVL ← DS.AVL;
GS.L ← DS.L;
GS.unusable ← 0;
GS.selector ← 0BH;

CR_DBGOPTIN ← TSC.FLAGS.DBGOPTIN;
Suppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        Suppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        CR_SAVE_TF ← RFLAGS.TF;
        RFLAGS.TF ← 0;
        Suppress_monitor_trap_flag for the source of the execution of the enclave;
        Suppress any pending debug exceptions;
        Suppress any pending MTF VM exit;
    ELSE
        IF RFLAGS.TF = 1
            THEN pend a single-step #DB at the end of EENTER; FI;
        IF the “monitor trap flag” VM-execution control is set
            THEN pend an MTF VM exit at the end of EENTER; FI;
FI;

Flush_linear_context;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0) If DS:RBX is not page aligned.

If the enclave is not initialized.

If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.

If the thread is not in the INACTIVE state.

If CS, DS, ES or SS bases are not all zero.

If executed in enclave mode.

If any reserved field in the TCS FLAG is set.

If the target address is not within the CS segment.

If CR4.OSFXSR = 0.

If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.

If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

#PF(error code) If a page fault occurs in accessing memory.

If DS:RBX does not point to a valid TCS.

If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
	If the target address is not canonical.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

...

EEXIT—Exits an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLU[EEXIT]	IR	V/V	SGX1	This leaf function is used to exit an enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EEXIT (In)	Target address outside the enclave (In)	Address of the current AEP (In)

Description

The ENCLU[EEXIT] instruction exits the currently executing enclave and branches to the location specified in RBX. RCX receives the current AEP. If RBX is not within the CS (32-bit mode) or is not canonical (64-bit mode) a #GP(0) results.

EEXIT Memory Parameter Semantics

Target Address
Non-Enclave read and execute access

If RBX specifies an address that is inside the enclave, the instruction will complete normally. The fetch of the next instruction will occur in non-enclave mode, but will attempt to fetch from inside the enclave. This has the effect of abort page semantics on the next destination.

If secrets are contained in any registers, it is responsibility of enclave software to clear those registers.

If XCR0 was modified on enclave entry, it is restored to the value it had at the time of the most recent EENTER or ERESUME.

If the enclave is opt-out, RFLAGS.TF is loaded from the value previously saved on EENTER.

Code and data breakpoints are unsuppressed.

Performance monitoring counters are unsuppressed.

Concurrency Restrictions

Table 41-54 Concurrency Restrictions of EEXIT with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EEXIT	TCS				Y	Y	Y		Y	Y		N	Y		Y	Y	Y		Y	Y	Y	Y	Y
	SSA		U		Y	Y	Y		Y	Y		Y	U		Y	Y	U		Y	Y	Y	Y	Y
	SECS			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-55 Concurrency Restrictions of EEXIT with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EEXIT	TCS	Y		Y		Y	Y	Y		Y	Y	Y	Y	Y		Y		Y	Y		Y	Y	Y
	SSA	Y				Y	Y	Y		Y	Y	Y	U	Y		Y		Y	U		Y	U	U
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EEXIT Operational Flow

Name	Type	Size (Bits)	Description
TMP_RIP	Effective Address	32/64	Saved copy of CRIP for use when creating LBR.

```
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF (TMP_MODE64 = 1)
    THEN
        IF (RBX is not canonical) THEN #GP(0); FI;
    ELSE
        IF (RBX > CS limit) THEN #GP(0); FI;
FI;

TMP_RIP ← CRIP;
RIP ← RBX;

(* Return current AEP in RCX *)
RCX ← CR_TCS_PA.AEP;

(* Do the FS/GS swap *)
FS.selector ← CR_SAVE_FS.selector;
FS.base ← CR_SAVE_FS.base;
FS.limit ← CR_SAVE_FS.limit;
FS.access_rights ← CR_SAVE_FS.access_rights;
GS.selector ← CR_SAVE_GS.selector;
GS.base ← CR_SAVE_GS.base;
GS.limit ← CR_SAVE_GS.limit;
GS.access_rights ← CR_SAVE_GS.access_rights;

(* Restore XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
    XCR0 ← CR_SAVE__XCR0;
FI;

Unsuppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        UnSuppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        Restore suppressed breakpoint matches;
        RFLAGS.TF ← CR_SAVE_TF;
        UnSuppress_montior_trap_flag;
        UnSuppress_LBR_Generation;
        UnSuppress_performance monitoring_activity;
        Restore performance monitoring counter AnyThread demotion to MyThread in enclave back to AnyThread
FI;

IF RFLAGS.TF = 1
    THEN Pend Single-Step #DB at the end of EEXIT;
FI;

IF the "monitor trap flag" VM-execution control is set
    THEN pend a MTF VM exit at the end of EEXIT;
FI;

CR_ENCLAVE_MODE ← 0;
CR_TCS_PA.STATE ← INACTIVE;

(* Assure consistent translations *)
Flush_linear_context;
```

Flags Affected

RFLAGS.TF is restored from the value previously saved in EENTER or ERESUME.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave. If RBX is outside the CS segment.
#PF(error code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RBX is not canonical.
#PF(error code)	If a page fault occurs in accessing memory operands.

...

EGETKEY—Retrieves a Cryptographic Key

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLU[EGETKEY]	IR	V/V	SGX1	This leaf function retrieves a cryptographic key.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EGETKEY (In)	Address to a KEYREQUEST (In)	Address of the OUTPUTDATA (In)

Description

The ENCLU[EGETKEY] instruction returns a 128-bit secret key from the processor specific key hierarchy. The register RBX contains the effective address of a KEYREQUEST structure, which the instruction interprets to determine the key being requested. The Requesting Keys section below provides a description of the keys that can be requested. The RCX register contains the effective address where the key will be returned. Both the addresses in RBX & RCX should be locations inside the enclave.

EGETKEY derives keys using a processor unique value to create a specific key based on a number of possible inputs. This instruction leaf can only be executed inside an enclave.

EEGETKEY Memory Parameter Semantics

KEYREQUEST	OUTPUTDATA
Enclave read access	Enclave write access

After validating the operands, the instruction determines which key is to be produced and performs the following actions:

- The instruction assembles the derivation data for the key based on the Table 41-56
- Computes derived key using the derivation data and package specific value
- Outputs the calculated key to the address in RCX

The instruction fails with #GP(0) if the operands are not properly aligned. Successful completion of the instruction will clear RFLAGS.{ZF, CF, AF, OF, SF, PF}. The instruction returns an error code if the user tries to request a key based on an invalid CPUSVN or ISVSVN (when the user request is accepted, see the table below), requests a key for which it has not been granted the attribute to request, or requests a key that is not supported by the hardware. These checks may be performed in any order. Thus, an indication by error number of one cause (for example, invalid attribute) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same Enclave. The correctness of software should not rely on the order resulting from the checks documented in this section. In such cases the ZF flag is set and the corresponding error bit (SGX_INVALID_SVN, SGX_INVALID_ATTRIBUTE, SGX_INVALID_KEYNAME) is set in RAX and the data at the address specified by RCX is unmodified.

Requesting Keys

The KEYREQUEST structure (see Section 38.17.1) identifies the key to be provided. The Keyrequest.KeyName field identifies which type of key is requested.

Deriving Keys

Key derivation is based on a combination of the enclave specific values (see Table 41-56) and a processor key. Depending on the key being requested a field may either be included by definition or the value may be included from the KeyRequest. A “yes” in Table 41-56 indicates the value for the field is included from its default location, identified in the source row, and a “request” indicates the values for the field is included from its corresponding KeyRequest field.

Table 41-56 Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODID	MRENCLAVE	MRSIGNER	RAND
Source	Key Dependent Constant	Y← SECS.ATTRIBUTES and SECS.MISCSELECT;	CSR_SEOWNEREPOCH	Y← CPUSVN Register;	R← Req.ISVSVN;	SECS. ISVID	SECS. MRENCLAVE	SECS. MRSIGNER	Req. KEYID
		R←AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT;		R← Req.CPUSVN;					
EINITTOKEN	Yes	Request	Yes	Request	Request	Yes	No	Yes	Request
Report	Yes	Yes	Yes	Yes	No	No	Yes	No	Request
Seal	Yes	Request	Yes	Request	Request	Yes	Request	Request	Request
Provisioning	Yes	Request	No	Request	Request	Yes	No	Yes	Yes
Provisioning Seal	Yes	Request	No	Request	Request	Yes	No	Yes	Yes

Keys that permit the specification of a CPU or ISV's code's SVNs have additional requirements. The caller may not request a key for an SVN beyond the current CPU or ISV SVN, respectively.

Several keys are access controlled. Access to the Provisioning Key and Provisioning Seal key requires the enclave's ATTRIBUTES.PROVISIONKEY be set. The EINITTOKEN Key requires ATTRIBUTES.EINITTOKENKEY be set and SECS.MRSIGNER equal IA32_SGXLEPUBKEYHASH.

Some keys are derived based on a hardcode PKCS padding constant (352 byte string):

```
HARDCODED_PKCS1_5_PADDING[15:0] ← 0100H;
HARDCODED_PKCS1_5_PADDING[2655:16] ← SignExtend330Byte(-1); // 330 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[2815:2656] ← 2004000501020403650148866009060D30313000H;
```

The error codes are:

Table 41-57 EGETKEY Return Value in RAX

Error Code (see Table 38-3)	Value	Description
No Error	0	EGETKEY successful
SGX_INVALID_ATTRIBUTE		The KEYREQUEST contains a KEYNAME for which the enclave is not authorized
SGX_INVALID_CPUSVN		If KEYREQUEST.CPUSVN is an unsupported platforms CPUSVN value
SGX_INVALID_ISVSVN		If KEYREQUEST.ISVSVN is greater than the enclave's ISV_SVN
SGX_INVALID_KEYNAME		If KEYREQUEST.KEYNAME is an unsupported value

Concurrency Restrictions

Table 41-58 Concurrency Restrictions of EGETKEY with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EGETKEY	Param		U							Y			U				U						
	SECS			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-59 Concurrency Restrictions of EGETKEY with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EGETKEY	Param			U								Y	U					Y	U			Y	U
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EGETKEY Operational Flow

Name	Type	Size (Bits)	Description
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_ATTRIBUTES		128	Temp Space for the calculation of the sealable Attributes.
TMP_OUTPUTKEY		128	Temp Space for the calculation of the key.

```
(* Make sure KEYREQUEST is properly aligned and inside the current enclave *)
IF ( (DS:RBX is not 128Byte aligned) or (DS:RBX is within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RBX is an EPC address and the EPC page is valid *)
IF ( (DS:RBX does not resolve to an EPC address) or (EPCM(DS:RBX).VALID = 0) )
```

```
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1) )
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #PF(DS:RBX);
FI;

(* Make sure OUTPUTDATA is properly aligned and inside the current enclave *)
IF ( (DS:RCX is not 16Byte aligned) or (DS:RCX is within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RCX is an EPC address and the EPC page is valid *)
IF ( (DS:RCX does not resolve to an EPC address) or (EPCM(DS:RCX).VALID = 0) )
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1) )
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).W = 0) )
    THEN #PF(DS:RCX);
FI;

(* Verify RESERVED spaces in KEYREQUEST are valid *)
IF ( (DS:RBX).RESERVED ≠ 0) or (DS:RBX.KEYPOLICY.RESERVED ≠ 0) )
    THEN #GP(0); FI;

TMP_CURRENTSECS ← CR_ACTIVE_SECS;

(* Determine which enclave attributes that must be included in the key. Attributes that must always be include INIT & DEBUG *)
REQUIRED_SEALING_MASK[127:0] ← 00000000 00000000 00000000 00000003H;
TMP_ATTRIBUTES ← (DS:RBX.ATTRIBUTEMASK | REQUIRED_SEALING_MASK) & TMP_CURRENTSECS.ATTRIBUTES;

(* Compute MISCSELECT fields to be included *)
TMP_MISCSELECT ← DS:RBX.MISCMASK & TMP_CURRENTSECS.MISCSELECT

CASE (DS:RBX.KEYNAME)
    SEAL_KEY:
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
```

```
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        // Include enclave identity?
        TMP_MRENCLAVE ← 0;
        IF (DS:RBX.KEYPOLICY.MRENCLAVE = 1)
            THEN TMP_MRENCLAVE ← TMP_CURRENTSECS.MRENCLAVE;
        FI;
        // Include enclave author?
        TMP_MRSIGNER ← 0;
        IF (DS:RBX.KEYPOLICY.MRSIGNER = 1)
            THEN TMP_MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
        FI;
        //Determine values key is based on
        TMP_KEYDEPENDENCIES.KEYNAME ← SEAL_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SEOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← TMP_MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK ← ~DS:RBX.MISCMASK;
        BREAK;
    REPORT_KEY:
        //Determine values key is based on
        TMP_KEYDEPENDENCIES.KEYNAME ← REPORT_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← 0;
        TMP_KEYDEPENDENCIES.ISVSVN ← 0;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SEOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_CURRENTSECS.ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← TMP_CURRENTSECS.MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
        TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← CR_CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← HARDCODED_PKCS1_5_PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_CURRENTSECS.MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK ← 0;
        BREAK;
    EINITTOKEN_KEY:
        (* Check ENCLAVE has LAUNCH capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.LAUNCHKEY = 0)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME ← EINITTOKEN_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SEOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
        TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK ← 0;
        BREAK;
    PROVISION_KEY:
    (* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
```

```
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME ← PROVISION_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← 0;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
        TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID ← 0;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← 0;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK ← ~DS:RBX.MISCMASK;
        BREAK;
    PROVISION_SEAL_KEY:
        (* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME ← PROVISION_SEAL_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← 0;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
        TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID ← 0;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK ← ~DS:RBX.MISCMASK;
        BREAK;
    DEFAULT:
        (* The value of KEYNAME is invalid *)
        RFLAGS.ZF ← 1;
        RAX ← SGX_INVALID_KEYNAME;
        GOTO EXIT:
ESAC;

(* Calculate the final derived key and output to the address in RCX *)
TMP_OUTPUTKEY ← derivekey(TMP_KEYDEPENDENCIES);
DS:RCX[15:0] ← TMP_OUTPUTKEY;
RAX ← 0;
RFLAGS.ZF ← 0;

EXIT:
RFLAGS.CF ← 0;
RFLAGS.PF ← 0;
RFLAGS.AF ← 0;
RFLAGS.OF ← 0;
RFLAGS.SF ← 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the current enclave.
If an effective address is not properly aligned.
If an effective address is outside the DS segment limit.
If KEYREQUEST format is invalid.

#PF(error code) If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0) If a memory operand effective address is outside the current enclave.
If an effective address is not properly aligned.
If an effective address is not canonical.
If KEYREQUEST format is invalid.

#PF(error code) If a page fault occurs in accessing memory operands.

...

EMODPE—Extend an EPC Page Permissions

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLU[EMODPE]	IR	V/V	SGX2	This leaf function extends the access rights of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EMODPE (In)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function extends the access rights associated with an existing EPC page in the running enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not extend the page permissions will have no effect. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPE leaf function.

EMODPE Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EMODPE Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is not valid.	RCX does not contain an effective address of an EPC page in the running enclave.
SECINFO contains an invalid request.	

Concurrency Restrictions

Table 41-60 Concurrency Restrictions of EMODPE with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EMODPE	Targ		Y							Y			Y				Y						
	SECINFO		U							Y			U				U						

Table 41-61 Concurrency Restrictions of EMODPE with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EMODPE	Targ			Y								N	Y	N		N		N	Y			Y	Y
	SECINFO			U								Y	Y					Y	Y			Y	Y

Operation

Temp Variables in EMODPE Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RBX).VALID = 0) or (EPCM(DS:RBX).R = 0) or (EPCM(DS:RBX).PENDING ≠ 0) or (EPCM(DS:RBX).MODIFIED ≠ 0) or
    (EPCM(DS:RBX).BLOCKED ≠ 0) or (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) )
    THEN #PF(DS:RBX); FI;

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero )
    THEN #GP(0); FI;

(* Check security attributes of the EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
```

```
     THEN #GP(0); FI;

(* Re-Check security attributes of the EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
     (EPCM(DS:RCX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
     (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
     THEN #PF(DS:RCX); FI;

(* Check for mis-configured SECINFO flags*)
IF ( (EPCM(DS:RCX).R = 0) and (SCRATCH_SECINFO.FLAGS.R = 0) and (SCRATCH_SECINFO.FLAGS.W ≠ 0) ))
     THEN #GP(0); FI;

(* Update EPCM permissions *)
EPCM(DS:RCX).R ← EPCM(DS:RCX).R | SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W ← EPCM(DS:RCX).W | SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X ← EPCM(DS:RCX).X | SCRATCH_SECINFO.FLAGS.X;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.

...

EREPORT—Create a Cryptographic Report of the Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLU[EREPORT]	IR	V/V	SGX1	This leaf function creates a cryptographic report of the enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX	RDX
IR	EREPORT (In)	Address of TARGETINFO (In)	Address of REPORTDATA (In)	Address where the REPORT is written to in an OUTPUTDATA (In)

Description

This leaf function creates a cryptographic REPORT that describes the contents of the enclave. This instruction leaf can only be executed when inside the enclave. The cryptographic report can be used by other enclaves to determine that the enclave is running on the same platform.

RBX contains the effective address of the MRENCLAVE value of the enclave that will authenticate the REPORT output, using the REPORT key delivered by EGETKEY command for that enclave. RCX contains the effective address of a 64-byte REPORTDATA structure, which allows the caller of the instruction to associate data with the enclave from which the instruction is called. RDX contains the address where the REPORT will be output by the instruction.

EREPORT Memory Parameter Semantics

TARGETINFO	REPORTDATA	OUTPUTDATA
Read access by Enclave	Read access by Enclave	Read/Write access by Enclave

This instruction leaf perform the following:

1. Validate the 3 operands (RBX, RCX, RDX) are inside the enclave.
2. Compute a report key for the target enclave, as indicated by the value located in RBX(TARGETINFO).
3. Assemble the enclave SECS data to complete the REPORT structure (including the data provided using the RCX (REPORTDATA) operand).
4. Computes a crytpographic hash over REPORT structure.
5. Add the computed hash to the REPORT structure.
6. Output the completed REPORT structure to the address in RDX (OUTPUTDATA).

The instruction fails if the operands are not properly aligned.

CR_REPORT_KEYID, used to provide key wearout protection, is populated with a statistically unique value on boot of the platform by a trusted entity within the SGX TCB.

The instruction faults if any of the following:

EREPORT Faulting Conditions

An effective address not properly aligned.	An memory address does not resolve in an EPC page.
If accessing an invalid EPC page.	If the EPC page is blocked.
May page fault.	

Concurrency Restrictions

Table 41-62 Concurrency Restrictions of EREPORT with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EREPORT	Param		U							Y			U				U						
	SECS			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 41-63 Concurrency Restrictions of EREPORT with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EREPORT	Param			U								Y	U					Y	U			Y	U
	SECS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Operation

Temp Variables in EREPORT Operational Flow

Name	Type	Size (bits)	Description
TMP_ATTRIBUTES		32	Physical address of SECS of the enclave to which source operand belongs.
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_REPORTKEY		128	REPORTKEY generated by the instruction.
TMP_REPORT		3712	

```
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Address verification for TARGETINFO (RBX) *)
IF ( (DS:RBX is not 128Byte Aligned) or (DS:RBX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX). VALID = 0)
    THEN #PF(DS:RBX); FI;
```

```
IF (EPCM(DS:RBX).BLOCKED = 1) )
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #PF(DS:RBX);
FI;

(* Address verification for REPORTDATA (RCX) *)
IF ( (DS:RCX is not 128Byte Aligned) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #P(DS:RCX); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1) )
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).R = 0) )
    THEN #PF(DS:RCX);
FI;

(* Address verification for OUTPUTDATA (RDX) *)
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RDX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX). VALID = 0)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX).BLOCKED = 1) )
    THEN #PF(DS:RDX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RDX).ENCLAVEADDRESS ≠ (DS:RDX & ~0FFFH) ) or (EPCM(DS:RDX).W = 0) )
    THEN #PF(DS:RDX);
FI;

(* REPORT MAC needs to be computed over data which cannot be modified *)
TMP_REPORT.CPUSVN ← CR_CPUSVN;
TMP_REPORT.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
TMP_REPORT.ISVSVN ← TMP_CURRENTSECS..ISVSVN;
TMP_REPORT.ATTRIBUTES ← TMP_CURRENTSECS.ATTRIBUTES;
TMP_REPORT.REPORTDATA ← DS:RCX[511:0];
TMP_REPORT.MRENCLAVE ← TMP_CURRENTSECS.MRENCLAVE;
TMP_REPORT.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
TMP_REPORT.MRRESERVED ← 0;
TMP_REPORT.KEYID[255:0] ← CR_REPORT_KEYID;
TMP_REPORT.MISCSELECT ← TMP_CURRENTSECS.MISCSELECT;
(* Derive the report key *)
TMP_KEYDEPENDENCIES.KEYNAME ← REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVPRODID ← 0;
TMP_KEYDEPENDENCIES.ISVSVN ← 0;
TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SGX_OWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← DS:RBX.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
TMP_KEYDEPENDENCIES.MRENCLAVE ← DS:RBX.MEASUREMENT;
TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
TMP_KEYDEPENDENCIES.KEYID ← TMP_REPORT.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT ← DS:RBX.MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK ← 0;

(* Calculate the derived key*)
TMP_REPORTKEY ← derive_key(TMP_KEYDEPENDENCIES);

(* call cryptographic CMAC function, CMAC data are not including MAC&KEYID *)
TMP_REPORT.MAC ← cmac(TMP_REPORTKEY, TMP_REPORT[3071:0] );
DS:RDX[3455: 0] ← TMP_REPORT;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If the address in RCS is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

...

ERESUME—Re-Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLU[ERESUME]	IR	V/V	SGX1	This leaf function is used to re-enter an enclave after an interrupt.

Instruction Operand Encoding

Op/En	RAX	RBX	RCX
IR	ERESUME (In)	Address of a TCS (In)	Address of AEP (In)

Description

The ENCLU[ERESUME] instruction resumes execution of an enclave that was interrupted due to an exception or interrupt, using the machine state previously stored in the SSA.

ERESUME Memory Parameter Semantics

TCS
Enclave read/write access

The instruction faults if any of the following:

Address in RBX is not properly aligned.	Any TCS.FLAGS’s must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use by another enclave.	Either of TCS-specified FS and GS segment is not a subset of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
Offsets 520-535 of the XSAVE area not 0.	The bit vector stored at offset 512 of the XSAVE area must be a subset of SECS.ATTRIBUTES.XFRM.
The SSA frame is not valid or in use.	

The following operations are performed by ERESUME:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or an asynchronous exit due to any Interrupt event.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 43.1.2):
    - — On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 43.2.5).

- On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 43.2.3).

- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.
- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 43.2.3):
  - All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
  - PEBS is suppressed.
  - AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set.
  - If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 41-64 Concurrency Restrictions of ERESUME with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Param	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
ERESUME	TCS	N			N				N	Y		N								N			N
	SSA		U							Y		Y	U				U						
	SECS			Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y

Table 41-65 Concurrency Restrictions of ERESUME with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Param	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
ERESUME	TCS	N					N			N						N							
	SSA			U								Y	U					Y	U			U	U
	SECS	Y	Y	Y	Y	Y	Y		Y		Y				Y		Y			Y			

Operation

Temp Variables in ERESUME Operational Flow

Name	Type	Size	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_TARGET	Effective Address	32/64	Address of first instruction inside enclave at which execution is to resume.
TMP_SECS	Effective Address	32/64	Physical address of SECS for this enclave.
TMP_SSA	Effective Address	32/64	Address of current SSA frame.

Name	Type	Size	Description
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.
TMP_BRANCH_RECORD	LBR Record		From/to addresses to be pushed onto the LBR stack.

```
TMP_MODE64 ← ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)
IF (TMP_MODE64 = 0 and (DS not usable or ( ( DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1) ) ) )
    THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)
IF (TMP_MODE64 = 0)
    THEN
        IF(CS.base ≠ 0 or DS.base ≠ 0) #GP(0); FI;
        IF(ES usable and ES.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.B = 0) #GP(0); FI;
FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (DS:RCX is not canonical) )
    THEN #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions is operating on TCS)
    THEN #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) or (EPCM(DS:RBX).PT ≠ PT_TCS) )
    THEN #PF(DS:RBX); FI;
```

```
IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS ← Address of SECS for TCS;

(* Make sure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & & FFFFFFFFFFFFFFFEH) ≠ 0)
    THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    THEN #GP(0); FI;

(* make sure the logical processor’s operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
    THEN #GP(0); FI;

IF (CR4.OSFXSR = 0)
    THEN #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    THEN
        IF (TMP_SECS.ATTRIBUES.XFRM ≠ 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUES.XFRM & XCR0) ≠ TMP_SECS.ATTRIBUES.XFRM) THEN #GP(0); FI;
FI;

(* Make sure the SSA contains at least one active frame *)
IF ( (DS:RBX).CSSA = 0)
    THEN #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA ← (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * ( (DS:RBX).CSSA - 1);
TMP_XSIZE ← compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;
    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
```

```
            THEN #PF(DS:TMP_SSA_PAGE); FI;
        IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE_.MODIFIED = 1))
            THEN #PF(DS:TMP_SSA_PAGE); FI;
        IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
            (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
            (EPCM(DS:TMP_SECS).R = 0) or (EPCM(DS:TMP_SECS).W = 0) )
            THEN #PF(DS:TMP_SSA_PAGE); FI;
        CR_XSAVE_PAGE_n ← Physical_Address(DS:TMP_SSA_PAGE);
    ENDFOR

    (* Compute address of GPR area*)
    TMP_GPR ← TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE -- sizeof(GPRSGX_AREA);
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;
    IF (DS:TMP_GPR does not resolve to EPC page)
        THEN #PF(DS:TMP_GPR); FI;
    IF (EPCM(DS:TMP_GPR).VALID = 0)
        THEN #PF(DS:TMP_GPR); FI;
    IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
        THEN #PF(DS:TMP_GPR); FI;
    IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
        THEN #PF(DS:TMP_GPR); FI;
    IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
        (EPCM(DS:TMP_GPR).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
        THEN #PF(DS:TMP_GPR); FI;

    IF (TMP_MODE64 = 0)
        THEN
            IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
    FI;

    CR_GPR_PA ← Physical_Address (DS: TMP_GPR);

    TMP_TARGET ← (DS:TMP_GPR).RIP;
    IF (TMP_MODE64 = 1)
        THEN
            IF (TMP_TARGET is not canonical) THEN #GP(0); FI;
        ELSE
            IF (TMP_TARGET > CS limit) THEN #GP(0); FI;
    FI;

    (* Check proposed FS/GS segments fall within DS *)
    IF (TMP_MODE64 = 0)
        THEN
            TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
            TMP_FSLIMIT ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
            TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
            TMP_GSLIMIT ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
            (* if FS wrap-around, make sure DS has no holes*)
            IF (TMP_FSLIMIT < TMP_FSBASE)
                THEN
                    IF (DS.limit < 4GB) THEN #GP(0); FI;
                ELSE
                    IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
            FI;
            (* if GS wrap-around, make sure DS has no holes*)
            IF (TMP_GSLIMIT < TMP_GSBASE)
                THEN
                    IF (DS.limit < 4GB) THEN #GP(0); FI;
                ELSE
                    IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
            FI;
        ELSE
            TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
            TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
            IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
                THEN #GP(0); FI;
    FI;

    (* Ensure the enclave is not already active and this thread is the only one using the TCS*)
    IF (DS:RBX.STATE = ACTIVE))
        THEN #GP(0); FI;

    (* SECS.ATTRIBUTES.XFRM selects the features to be saved. *)
    (* CR_XSAVE_PAGE_n: A list of 1 or more physical address of pages that contain the XSAVE area. *)
    XRSTOR(TMP_MODE64, SECS.ATTRIBUTES.XFRM, CR_XSAVE_PAGE_n);

    IF (XRSTOR failed with #GP)
        THEN
            DS:RBX.STATE ← INACTIVE;
            #GP(0);
    FI;

    CR_ENCALVE_MODE ← 1;
    CR_ACTIVE_SECS ← TMP_SECS;
    CR_ELRANGE ← (TMP_SECS.BASEADDR, TMP_SECS.SIZE);

    (* Save sate for possible AEXs *)
    CR_TCS_PA ← Physical_Address (DS:RBX);
    CR_TCS_LA ← RBX;
    CR_TCS_LA.AEP ← RCX;

    (* Save the hidden portions of FS and GS *)
    CR_SAVE_FS_selector ← FS.selector;
    CR_SAVE_FS_base ← FS.base;
    CR_SAVE_FS_limit ← FS.limit;
    CR_SAVE_FS_access_rights ← FS.access_rights;
    CR_SAVE_GS_selector ← GS.selector;
    CR_SAVE_GS_base ← GS.base;
```

```
CR_SAVE_GS_limit ← GS.limit;
CR_SAVE_GS_access_rights ← GS.access_rights;

(* Set CR_ENCLAVE_ENTRY_IP *)
CR_ENCLAVE_ENTRY_IP ← CRIP”
RIP ← TMP_TARGET;

Restore_GPRs from DS:TMP_GPR;

(*Restore the RFLAGS values from SSA*)
RFLAGS.CF ← DS:TMP_GPR.RFLAGS.CF;
RFLAGS.PF ← DS:TMP_GPR.RFLAGS.PF;
RFLAGS.AF ← DS:TMP_GPR.RFLAGS.AF;
RFLAGS.ZF ← DS:TMP_GPR.RFLAGS.ZF;
RFLAGS.SF ← DS:TMP_GPR.RFLAGS.SF;
RFLAGS.DF ← DS:TMP_GPR.RFLAGS.DF;
RFLAGS.OF ← DS:TMP_GPR.RFLAGS.OF;
RFLAGS.NT ← DS:TMP_GPR.RFLAGS.NT;
RFLAGS.AC ← DS:TMP_GPR.RFLAGS.AC;
RFLAGS.ID ← DS:TMP_GPR.RFLAGS.ID;
RFLAGS.RF ← DS:TMP_GPR.RFLAGS.RF;
RFLAGS.VM ← 0;
IF (RFLAGS.IOPL = 3)
    THEN RFLAGS.IF = DS:TMP_GPR.IF; FI;

IF (TCS.FLAGS.OPTIN = 0)
    THEN RFLAGS.TF = 0; FI;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 ← XCR0;
    XCR0 ← TMP_SECS.ATTRIBUTES.XFRM;
FI;

(* Pop the SSA stack*)
(DS:RBX).CSSA ← (DS:RBX).CSSA -1;

(* Do the FS/GS swap *)
FS.base ← TMP_FSBASE;
FS.limit ← DS:RBX.FSLIMIT;
FS.type ← 0001b;
FS.W ← DS.W;
FS.S ← 1;
FS.DPL ← DS.DPL;
FS.G ← 1;
FS.B ← 1;
FS.P ← 1;
FS.AVL ← DS.AVL;
FS.L ← DS.L;
FS.unusable ← 0;
FS.selector ← 0BH;

GS.base ← TMP_GSBASE;
GS.limit ← DS:RBX.GSLIMIT;
GS.type ← 0001b;
GS.W ← DS.W;
GS.S ← 1;
GS.DPL ← DS.DPL;
GS.G ← 1;
GS.B ← 1;
GS.P ← 1;
GS.AVL ← DS.AVL;
GS.L ← DS.L;
GS.unusable ← 0;
GS.selector ← 0BH;

CR_DBGOPTIN ← TSC.FLAGS.DBGOPTIN;
Suppress all code breakpoints that are outside ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        Suppress all code breakpoints that overlap with ELRANGE;
        CR_SAVE_TF ← RFLAGS.TF;
        RFLAGS.TF ← 0;
        Suppress any MTF VM exits during execution of the enclave;
        Clear all pending debug exceptions;
        Clear any pending MTF VM exit;
    ELSE
        Clear all pending debug exceptions;
        Clear pending MTF VM exits;
FI;

(* Assure consistent translations *)
Flush_linear_context;
Clear_Monitor_FSM;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0) If DS:RBX is not page aligned.

If the enclave is not initialized.

If the thread is not in the INACTIVE state.

If CS, DS, ES or SS bases are not all zero.

If executed in enclave mode.

If part or all of the FS or GS segment specified by TCS is outside the DS segment.

If any reserved field in the TCS FLAG is set.

If the target address is not within the CS segment.

If CR4.OSFXSR = 0.

If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.

If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

#PF(error code) If a page fault occurs in accessing memory.

If DS:RBX does not point to a valid TCS.

If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0) If DS:RBX is not page aligned.

If the enclave is not initialized.

If the thread is not in the INACTIVE state.

If CS, DS, ES or SS bases are not all zero.

If executed in enclave mode.

If part or all of the FS or GS segment specified by TCS is outside the DS segment.

If any reserved field in the TCS FLAG is set.

If the target address is not canonical.

If CR4.OSFXSR = 0.

If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.

If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

#PF(error code) If a page fault occurs in accessing memory operands.

If DS:RBX does not point to a valid TCS.

If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

...

34. Updates to Chapter 42, Volume 3D

Change bars show changes to Chapter 42 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

42.2 IA32_FEATURE_CONTROL

IA32_FEATURE_CONTROL MSR provides two new bits related to two aspects of Intel SGX: using the instruction extensions and launch control configuration.

42.2.1 Availability of Intel SGX

IA32_FEATURE_CONTROL[bit 18] allows BIOS to control the availability of Intel SGX extensions. For Intel SGX extensions to be available on a logical processor, bit 18 in the IA32_FEATURE_CONTROL MSR on that logical processor must be set, and IA32_FEATURE_CONTROL MSR on that logical processor must be locked (bit 0 must be set). See Section 37.7.1 for additional details. OS is expected to examine the value of bit 18 prior to enabling Intel SGX on the thread, as the settings of bit 18 is not reflected by CPUID.

42.2.2 Intel SGX Launch Control Configuration

The IA32_SGXLEPUBKEYHASHn MSRs used to configure authorized launch enclaves MRSIGNER digest value are present on logical processors that support the collection of SGX1 leaf functions (i.e. CPUID.(EAX=12H, ECX=00H):EAX[0] = 1). IA32_FEATURE_CONTROL[bit 17] allows to BIOS to enable write access to these MSRs. If IA32_FEATURE_CONTROL.LE_WR (bit 17) is set to 1 and IA32_FEATURE_CONTROL is locked on that logical processor, IA32_SGXLEPUBKEYHASH MSRs on that logical processor then the IA32_SGXLEPUBKEYHASHn MSR are writeable. If this bit 17 is not set or IA32_FEATURE_CONTROL is not locked, IA32_SGXLEPUBKEYHASH MSRs are read only. See Section 39.1.4 for additional details.

...

42.3.3 Interaction of Intel® SGX Instructions with Segmentation

All leaf functions of ENCLU and ENCLS instructions require that the DS segment be usable, and be an expand-up segment. Failing this check results in generation of a #GP(0) exception.

The Intel SGX leaf functions used for entering the enclave (ENCLU[EENTER] and ENCLU[ERESUME]) operate as follows:

- All usable segment registers except for FS and GS have a zero base.
- The contents of the FS/GS segment registers (including the hidden portion) is saved in the processor.
- New FS and GS values compatible with enclave security are loaded from the TCS
- The linear ranges and access rights available under the newly-loaded FS and GS must abide to OS policies by ensuring they are subsets of the linear-address range and access rights available for the DS segment.
- The CS segment mode (64-bit, compatible, or 32 bit modes) must be consistent with the segment mode for which the enclave was created, as indicated by the SECS.ATTRIBUTES.MODE64 bit, and that the CPL of the logical processor is 3

An exit from the enclave either via ENCLU[EEXIT] or via an AEX restores the saved values of FS/GS segment registers.

42.3.4 Interactions of Enclave Execution with Segmentation

During the course of execution, enclave code abides by all segmentation policies as dictated by IA32 and Intel 64 Architectures, and generates appropriate exceptions on violations.

Additionally, any attempt by software executing inside an enclave to modify the processor's segmentation state (e.g. via MOV seg register, POP seg register, LDS, far jump, etc; excluding WRFSBASE/WRGSBASE) results in the generation of a #UD. See Section 39.6.1 for more information.

Upon enclave entry via the EENTER leaf function, FS is loaded from the (TCS.OFSBASE + SECS.BASEADDR) and TCS.FSLIMIT fields and GS is loaded from the (TCS.OGSBASE + SECS.BASEADDR) and TCS.GSLIMIT fields.

Execution of WRFSBASE and WRGSBASE from inside a 64-bit enclave is allowed. The processor will save the new values into the current SSA frame on an asynchronous exit (AEX) and restore them back on enclave entry via ENCLU[ERESUME] instruction.

...

42.4 INTERACTIONS WITH PAGING

Intel SGX instructions are available only when the processor is executing in a protected mode of operation. Additionally, all Intel SGX leaf functions except for EDBGRD and EDBGWR are available only if paging is enabled. Any attempt to execute these leaf functions with paging disabled results in an invalid-opcode exception (#UD). As with segmentation, enclaves abide by all the paging policies set up by the OS, but they can be more restrictive than the OS.

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the DS segment, and the linear addresses generated by combining these offsets with DS segment register are subject to paging-based access control if paging is enabled at the time of the execution of the leaf function.

Since the ENCLU[EENTER] and ENCLU[ERESUME] can only be executed when paging is enabled, and since paging cannot be disabled by software running inside an enclave (recall that enclaves always run with CPL = 3), enclave execution is always subject to paging-based access control. The Intel SGX access control itself is implemented as an extension to the existing paging modes. See Section 38.5 for details.

Execution of Intel SGX instructions may set accessed and dirty flags on accesses to EPC pages that do not fault even if the instruction later causes a fault for some other reason.

42.5 INTERACTIONS WITH VMX

Intel SGX functionality (including SGX1 and SGX2) can be made available to software running in either VMX root operation or VMX non-root operation, as long as the processor is using a legal mode of operation (see Section 42.1).

A VMM has the flexibility to configure a VMCS to permit a guest to use any subset of the ENCLS leaf functions. Availability of the ENCLU leaf functions in VMX non-root operation has the same requirement as ENCLU leaf functions outside of a virtualized environment.

Details of the VMCS control to allow VMM to configure support of Intel SGX in VMX non-root operation is described in Section 42.5.1

42.5.1 VMM Controls to Configure Guest Support of Intel® SGX

Intel SGX capabilities are primarily exposed to the software via the CPUID instruction. VMMs can virtualize CPUID instruction to expose/hide this capability to/from guests.

Some of Intel SGX resources are exposed/controlled via model-specific registers (see Section 37.7). VMMs can virtualize these MSRs for the guests using the MSR bitmaps referenced by pointers in the VMCS.

The VMM can partition the Enclave Page Cache, and assign various partitions to (a subset of) its guests via the usual memory-virtualization techniques such as paging or the extended page table mechanism (EPT).

The VMM can set the "enable ENCLS exiting" VM-execution controls to cause a VM exit when the ENCLS instruction is executed in VMX non-root operation. If the "enable ENCLS exiting" control is 0, all of the ENCLS leaf functions are permitted in VMX non-root operation. If the "enable ENCLS exiting" control is 1, execution of ENCLS leaf functions in VMX non-root operation is governed by consulting the bits in a new 64-bit VM-execution control field called the ENCLS-exiting bitmap (Each bit in the bitmap corresponds to an ENCLS leaf function with an EAX value that is identical to the bit's position). When bits in the "ENCLS-exiting bitmap" are set, attempts to execute the corresponding ENCLS leaf functions in VMX non-root operation causes VM exits. The checking for these VM exits occurs immediately after checking that CPL = 0.

42.5.2 Interactions with the Extended Page Table Mechanism (EPT)

Intel SGX instructions are fully compatible with the extended page-table mechanism (EPT; see Section 28.2).

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the DS segment, and the linear addresses generated by combining these offsets with DS segment register are subject to paging and EPT. As with paging, enclaves abide by all the policies set up by the VMM.

The Intel SGX access control itself is implemented as an extension to paging and EPT, and may be more restrictive. See Section 42.4 for details of this extension.

An execution of an Intel SGX instruction may set accessed and dirty flags for EPT (when enabled; see Section 28.2.4) on accesses to EPC pages that do not fault or cause VM exits even if the instruction later causes a fault or VM exit for some other reason.

42.5.3 Interactions with APIC Virtualization

This section applies to Intel SGX in VMX non-root operation when the "virtualize APIC accesses" VM-execution control is 1.

A memory access by an enclave instruction that implicitly uses a cached physical address is never checked for overlap with the APIC-access page. Such accesses never cause APIC-access VM exits and are never redirected to the virtual-APIC page. Implicit memory accesses can only be made to the SECS, the TCS, or the SSA of an enclave (see Section 38.5.3.2).

An explicit Enclave Access (a linear memory access which is either from within an enclave into its ELRANGE, or an access by an Intel SGX instruction that is expected to be in the EPC) that overlaps with the APIC-access page causes a #PF exception (APIC page is expected to be outside of EPC).

Non-Enclave accesses made either by an Intel SGX instruction or by a logical processor inside an enclave to an address that without SGX would have caused redirection to the virtual-APIC page instead cause an APIC-access VM exit.

Other than implicit accesses made by Intel SGX instructions, guest-physical and physical accesses are not considered "enclave accesses"; consequently, such accesses result in undefined behavior if these accesses eventually reach EPC. This applies to any non-enclave physical accesses.

While a logical processor is executing inside an enclave, an attempt to execute an instruction outside of ELRANGE results in a #GP(0), even if the linear address would translate to a physical address that overlaps the APIC-access page.

42.6 INTEL® SGX INTERACTIONS WITH ARCHITECTURALLY-VISIBLE EVENTS

All architecturally visible vectored events (IA32 exceptions, interrupts, SMI, NMI, INIT, VM exit) can be detected while inside an enclave and will cause an asynchronous enclave exit if they are not blocked. Additionally, INT3, and the SignalTXTMsg[SENTER] (i.e. GETSEC[SENTER]'s rendezvous event message) events also cause asynchronous enclave exits. Note that SignalTXTMsg[SEXIT] (i.e. GETSEC[SEXIT]'s teardown message) does not cause an AEX.

On an AEX, information about the event causing the AEX is stored in the SSA (see Section 40.4 for details of AEX). The information stored in the SSA only describes the first event that triggered the AEX. If parsing/delivery of the first event results in detection of further events (e.g. VM exit, double fault, etc.), then the event information in the SSA is not updated to reflect these subsequently detected events.

...

42.7.1 Requirements and Architecture Overview

Processor extended states are the ISA features that are enabled by the settings of CR4.OSXSAVE and the XCR0 register. Processor extended states are normally saved/restored by software via XSAVE/XRSTOR instructions. Details of discovery of processor extended states and management of these states are described in CHAPTER 13 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Additionally, the following requirements apply to Intel SGX:

- On an AEX, the Intel SGX architecture must protect the processor extended state and miscellaneous state by saving them in the enclave's state-save area (SSA), and clear the secrets from the processor extended state that is used by an enclave.
- Intel SGX architecture must verify that the SSA frame size is large enough to contain all the processor extended states and miscelaneous state used by the enclave.
- Intel SGX architecture must ensure that enclaves can only use processor extended state that is enabled by system software in XCR0.
- Enclave software should be able to discover only those processor extended state and miscellaneous state for which such protection is enabled.
- The processor extended states that are enabled inside the enclave must be approved by the enclave developer:
  - — Certain processor extended state (e.g., Memory Protection Extensions, see Chapter 9 of *Intel® MPX* of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*) modify the behavior of the legacy ISA software. If such features are enabled for enclaves that do not understand those features, then such a configuration could lead to a compromise of the enclave's security.
- The processor extended states that are enabled inside the enclave must form an integral part of the enclave's identity. This requirement has two implications:
  - — Service providers may decide to assign different trust level to the same enclave depending on the ISA features the enclave is using.

To meet these requirements, the Intel SGX architecture defines a sub-field called X-Feature Request Mask (XFRM) in the ATTRIBUTES field of the SECS. On enclave creation (ENCLS[ECREATE] leaf function), the required SSA frame size is calculated by the processor from the list of enabled extended and miscellaneous states and verified against the actual SSA frame size defined by SECS.SSAFRAMESIZE.

On enclave entry, after verifying that XFRM is only enabling features that are already enabled in XCR0, the value in the XCR0 is saved internally by the processor, and is replaced by the XFRM. On enclave exit, the original value of XCR0 is restored. Consequently, while inside the enclave, the processor extended states enabled in XFRM are in enabled state, and those that are disabled in XFRM are in disabled state.

The entire ATTRIBUTES field, including the XFRM subfield is integral part of enclave's identity (i.e., its value is included in reports generated by ENCLU[EREPORT], and select bits from this field can be included in key-derivation for keys obtained via the ENCLU[EGETKEY] leaf function).

Enclave developers can create their enclave to work with certain features and fallback to another code path in case those features aren't available (e.g. optimize for AVX and fallback to SSE). For this purpose Intel SGX provides the following fields in SIGSTRUCT: ATTRIBUTES, ATTRIBUTESMASK, MISCSELECT, and MISCMASK. EINIT ensures that the final SECS.ATTRIBUTES and SECS.MISCSELECT comply with the enclave developer's requirements as follows:

SIGSTRUCT.ATTRIBUTES & SIGSTRUCT.ATTRIBUTEMASK = SECS.ATTRIBUTES & SIGSTRUCT.ATTRIBUTEMASK

SIGSTRUCT.MISCSELECT & SIGSTRUCT.MISCMASK = SECS.MISCSELECT & SIGSTRUCT.MISCMASK.

On an asynchronous enclave exit, the processor extended states enabled by XFRM are saved in the current SSA frame, and overwritten by synthetic state (see Section 40.3 for the definition of the synthetic state). When the interrupted enclave is resumed via the ENCLU[ERESUME] leaf function, the saved state for processor extended states enabled by XFRM is restored.

...

42.7.2.1 SECS.ATTRIBUTES.XFRM

The ATTRIBUTES field of the SECS data structure (see Section 38.7) contains a sub-field called XSAVE-Feature Request Mask (XFRM). Software populates this field at the time of enclave creation according to the features that are enabled by the operating system and approved by the enclave developer.

Intel SGX architecture guarantees that during enclave execution, the processor extended state configuration of the processor is identical to what is required by the XFRM sub-field. All the processor extended states enabled in XFRM are saved on AEX from the enclave and restored on ERESUME.

The XFRM sub-field has the same layout as XCR0, and has consistency requirements that are similar to those for XCR0. Specifically, the consistency requirements on XFRM values depend on the processor implementation and the set of features enabled in CR4.

Legal values for SECS.ATTRIBUTES.XFRM conform to these requirements:

- XFRM[1:0] must be set to 0x3.
- If the processor does not support XSAVE, or if the system software has not enabled XSAVE, then XFRM[63:2] must be zero.
- If the processor does support XSAVE, XFRM must contain a value that would be legal if loaded into XCR0.

The various consistency requirements are enforced at different times in the enclave's life cycle, and the exact enforcement mechanisms are elaborated in Section 42.7.3 through Section 42.7.6.

On processors not supporting XSAVE, software should initialize XFRM to 0x3. On processors supporting XSAVE, software should initialize XFRM to be a subset of XCR0 that would be present at the time of enclave execution. Because bits 0 and 1 of XFRM must always be set, the use of Intel SGX requires that SSE be enabled (CR4.OSFXSR = 1).

42.7.2.2 SECS.SSAFRAMESIZE

The SSAFRAMESIZE field in the SECS data structure specifies the number of pages which software allocated[1] for each SSA frame, including both the GPRSGX area, MISC area, the XSAVE area (x87 and XMM states are stored in the latter area), and optionally padding between the MISC and XSAVE area. The GPRSGX area must hold all the general-purpose registers and additional Intel SGX specific information. The MISC area must hold the Miscellaneous state as specified by SECS.MISCSELECT, the XSAVE area holds the set of processor extended states speci-

[1] It is the responsibility of the enclave to actually allocate this memory.

fied by SECS.ATTRIBUTES.XFRM (see Section 38.9 for the layout of SSA and Section 42.7.3 for ECREATE's consistency checks). The SSA is always in non-compacted format.

If the processor does not support XSAVE, the XSAVE area will always be 576 bytes; a copy of XFRM (which will be set to 0x3) is saved at offset 512 on an AEX.

If the processor does support XSAVE, the length of the XSAVE area depends on SECS.ATTRIBUTES.XFRM. The length would be equal to what CPUID.(EAX=0DH, ECX= 0):EBX would return if XCR0 were set to XFRM. The following pseudo code illustrates how software can calculate this length using XFRM as the input parameter without modifying XCR0:

```
offset = 576;
size_last_x = 0;
For x=2 to 63
IF (XFRM[x] != 0) Then
    tmp_offset = CPUID.(EAX=0DH, ECX= x):EBX[31:0];
    IF (tmp_offset >= offset + size_last_x) Then
        offset = tmp_offset;
        size_last_x = CPUID.(EAX=0DH, ECX= x):EAX[31:0];
     FI;
FI;
EndFor
return (offset + size_last_x); (* compute_xsave_size(XFRM), see “ECREATE—Create an SECS page in the
Enclave Page Cache”*)
```

Where the non-zero bits in XFRM are a subset of non-zero bit fields in XCR0.

The size of the MISC region depends on the setting of SECS.MISCSELECT and can be calculated using the layout information described in Section 38.9.2

...

42.7.2.4 MISC Area in SSA

The MISC area of an SSA frame is positioned immediately before the GPRSGX region.

42.7.2.5 SIGSTRUCT Fields

Intel SGX provides the flexibility for an enclave developer to choose the enclave's code path according to the features that are enabled on the platform (e.g. optimize for AVX and fallback to SSE). See Section 42.7.1 for details.

SIGSTRUCT includes the following fields:

SIGSTRUCT.ATTRIBUTES, SIGSTRUCT.ATTRIBUTEMASK, SIGSTRUCT.MISCSELECT, SIGSTRUCT.MISCMASK.

42.7.2.6 REPORT.ATTRIBUTES.XFRM and REPORT.MISCSELECT

The processor extended states and miscellaneous states that are enabled inside the enclave form an integral part of the enclave's identity and are therefore included in the enclave's report, as provided by the ENCLU[EREPORT] leaf function. The REPORT structure includes the enclave's XFRM and MISCSELECT configurations.

42.7.2.7 KEYREQUEST

An enclave developer can specify which bits out of XFRM and MISCSELECT ENCLU[EGETKEY] should include in the derivation of the sealing key by specifying ATTRIBUTESMASK and MISCMASK in the KEYREQUEST structure.

42.7.3 Processor Extended States and ENCLS[ECREATE]

The ECREATE leaf function of the ENCLS instruction enforces a number of consistency checks described earlier. The execution of ENCLS[ECREATE] leaf function results in a #GP(0) in any of the following cases:

- SECS.ATTRIBUTES.XFRM[1:0] is not 3.
- The processor does not support XSAVE and any of the following is true:
  - SECS.ATTRIBUTES.XFRM[63:2] is not 0.
  - SECS.SSAFRAMESIZE is 0.
- The processor supports XSAVE and any of the following is true:
  - XSETBV would fault on an attempt to load XFRM into XCR0.
  - XFRM[63]=1.
  - The SSAFRAME is too small to hold required, enabled states (see Section 42.7.2.2).

...

42.7.4.1 Fault Checking

The EENTER leaf function of the ENCLU instruction enforces a number of consistency requirements described earlier. The execution of the ENCLU[EENTER] leaf function results in a #GP(0) in any of the following cases:

- If CR4.OSFXSR=0.
- If The processor supports XSAVE and either of the following is true:
  - CR4.OSXSAVE=0 and SECS.ATTRIBUTES.XFRM is not 3.
  - (SECS.ATTRIBUTES.XFRM & XCR0) != SECS.ATTRIBUTES.XFRM

...

42.7.6.1 Fault Checking

The ERESUME leaf function of the ENCLU instruction enforces a number of consistency requirements described earlier. Specifically, the ENCLU[ERESUME] leaf function results in a #GP(0) in any of the following cases:

- CR4.OSFXSR=0.
- The processor supports XSAVE and either of the following is true:
  - CR4.OSXSAVE=0 and SECS.ATTRIBUTES.XFRM is not 3.
  - (SECS.ATTRIBUTES.XFRM & XCR0) != SECS.ATTRIBUTES.XFRM.

A successful execution of ENCLU[ERESUME] loads state from the XSAVE area of the SSA frame in a fashion similar to that used by the XRSTOR instruction. Data in the XSAVE area that would cause the XRSTOR instruction to fault will cause the ENCLU[ERESUME] leaf function to fault. Examples include, but are not restricted to the following:

- A bit is set in the XSTATE_BV field and clear in XFRM.
- The required bytes in the header are not clear.
- Loading data would set a reserved bit in MXCSR.

Any of these conditions will cause ERESUME to fault, even if CR4.OSXSAVE=0.

...

42.7.7 Processor Extended States and ENCLU[EEXIT]

The ENCLU[EEXIT] leaf function does not perform any X-feature specific consistency checks, nor performs any state synthesis. It is the responsibility of enclave software to clear any sensitive data from the registers before executing EEXIT. However, successful execution of the ENCLU[EEXIT] leaf function restores XCR0 to the value it held at the time of the most recent enclave entry.

42.7.8 Processor Extended States and ENCLU[EREPORT]

The ENCLU[EREPORT] leaf function creates the MAC-protected REPORT structure that reports on the enclave's identity. ENCLU[EREPORT] includes in the report the values of SECS.ATTRIBUTES.XFRM and SECS.MISCSELECT.

42.7.9 Processor Extended States and ENCLU[EGETKEY]

The ENCLU[EGETKEY] leaf function returns a cryptographic key based on the information provided by the KEYREQUEST structure. Intel SGX provides the means for isolation between different operating conditions by allowing an enclave developer to select which bits out of XFRM and MISCSELECT need to be included in the derivation of the keys.

...

42.8.1 Availability of Intel® SGX instructions in SMM

Enclave instructions are not available in SMM, and any attempt to execute ENCLS or ENCLU instructions inside SMM results in an invalid-opcode exception (#UD).

42.8.2 SMI while Inside an Enclave

If the logical processor executing inside an enclave receives an SMI, the logical processor exits the enclave asynchronously. The response to an SMI received while executing inside an enclave depends on whether the dual-monitor treatment is enabled. For detailed discussion of transfer to SMM, see Chapter 34, "System Management Mode" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is not enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM handler. In addition to saving the synthetic architectural state to the SMRAM State Save Map (SSM), the logical processor also sets the "Enclave Interruption" bit in the SMRAM SSM (bit position 1 in SMRAM field at offset 7EE0H).

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM monitor via SMM VM exit. The SMM VM exit sets the "Enclave Interruption" bit in the Exit Reason (see Table 42-1) and in the Guest Interruptibility State field (see Table 42-2) of the SMM VMCS.

...

42.9 INTERACTIONS OF INIT, SIPI, AND WAIT-FOR-SIPI WITH INTEL® SGX

INIT received inside an enclave, while the logical processor is not in VMX operation, causes the logical processor to exit the enclave asynchronously. After the AEX, the processor's architectural state is initialized to "Power-on" state (Table 9.1 in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*). If the logical processor is BSP, then it proceeds to execute the BIOS initialization code. If the logical processor is an AP, it enters wait-for-SIPI state.

INIT received inside an enclave, while the logical processor (LP) is in VMX root operation, follows regular Intel Architecture behavior and is blocked.

INIT received inside an enclave, while the logical processor is in VMX non-root operation, causes an AEX. Subsequent to the AEX, the INIT causes a VM exit with the Enclave Interruption bit in the exit-reason field in the VMCS.

A processor cannot be inside an enclave in the wait-for-SIPI state. Consequently, a SIPI received while inside an enclave is lost.

Intel SGX does not change the behavior of the processor in the wait-for-SIPI state.

The SGX-related processor states after INIT-SIPI-SIPI is as follows:

- EPC Settings: Unchanged
- EPCM: Unchanged
- CPUID.LEAF_12H.*: Unchanged
- ENCLAVE_MODE: 0 (LP exits enclave asynchronously)
- MEE state: Unchanged

Software should be aware that following INIT-SIPI-SIPI, the EPC might contain valid pages and should take appropriate measures such as initialize the EPC with the EREMOVE leaf function.

42.10 INTERACTIONS WITH DMA

DMA is not allowed to access any Processor Reserved Memory.

42.11 INTERACTIONS WITH TXT

42.11.1 Enclaves Created Prior to Execution of GETSEC

Enclaves which have been created before the GETSEC[SENTER] leaf function are available for execution after the successful completion of GETSEC[SENTER] and the corresponding SINIT ACM. Actions that a TXT Launched Environment performs in preparation to execute code in the Launched Environment, also applies to enclave code that would run after GETSEC[SENTER].

42.11.2 Interaction of GETSEC with Intel® SGX

All leaf functions of the GETSEC instruction are illegal inside an enclave, and results in an invalid-opcode exception (#UD).

Responding Logical Processors (RLP) which are executing inside an enclave at the time a GETSEC[SENTER] event occurs perform an AEX from the enclave and then enter the Wait-for-SIPI state.

RLP executing inside an enclave at the time of GETSEC[SEXIT], behave as defined for GETSEC[SEXIT]-that is, the RLPs pause during execution of SEXIT and resume after the completion of SEXIT.

The execution of a TXT launch does not affect Intel SGX configuration or security parameters.

42.11.3 Interactions with Authenticated Code Modules (ACMs)

Intel SGX only allows launching ACMs with an Intel SGX SVN that is at the same level or higher than the expected Intel SGX SVN. The expected Intel SGX SVN is specified by BIOS and locked down by the processor on the first successful execution of an Intel SGX instruction that doesn't return an error code. Intel SGX provides interfaces for system software to discover whether a non faulting Intel SGX instruction has been executed, and evaluate the suitability of the Intel SGX SVN value of any ACM that is expected to be launched by the OS or the VMM.

These interfaces are provided through a read-only MSR called the IA32_SGX_SVN_STATUS MSR (MSR address 500h). The IA32_SGX_SVN_STATUS MSR has the format shown in Table 42-2.

Table 42-2 Layout of the IA32_SGX_SVN_STATUS MSR

Bit Position	Name	ACM Module ID	Value
0	Lock	N.A.	▪ If 1, indicates that a non-faulting Intel SGX instruction has been executed, consequently, launching a properly signed ACM but with Intel SGX SVN value less than the BIOS specified Intel SGX SVN threshold would lead to an TXT shutdown. ▪ If 0, indicates that the processor will allow a properly signed ACM to launch irrespective of the Intel SGX SVN value of the ACM.
15:1	RSVD	N.A.	0
23:16	SGX_SVN_SINIT	SINIT ACM	▪ If CPUID.01H:ECX.SMX =1, this field reflects the expected threshold of Intel SGX SVN for the SINIT ACM. ▪ If CPUID.01H:ECX.SMX =0, this field is reserved (0).
63:24	RSVD	N.A.	0

OS/VMM that wishes to launch an architectural ACM such as SINIT is expected to read the IA32_SGX_SVN_STATUS MSR to determine whether the ACM can be launched or a new ACM is needed:

- If either the Intel SGX SVN of the ACM is greater than the value reported by IA32_SGX_SVN_STATUS, or the lock bit in the IA32_SGX_SVN_STATUS is not set, then the OS/VMM can safely launch the ACM.
- If the Intel SGX SVN value reported in the corresponding component of the IA32_SGX_SVN_STATUS is greater than the Intel SGX SVN value in the ACM's header, and if bit 0 of IA32_SGX_SVN_STATUS is 1, then the OS/VMM should not launch that version of the ACM. It should obtain an updated version of the ACM either from the BIOS or from an external resource.

However, OSVs/VMMs are strongly advised to update their version of the ACM any time they detect that the Intel SGX SVN of the ACM carried by the OS/VMM is lower than that reported by IA32_SGX_SVN_STATUS MSR, irrespective of the setting of the lock bit.

...

42.13.1 HLE and RTM Debug

RTM debug will be suppressed on opt-out enclave entry. After opt-out entry, the logical processor will behave as if IA32_DEBUG_CTL[15]=0. Any #DB detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if either DR7[11] = 0 OR IA32_DEBUGCTL[15] = 0, any #DB or #BP detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if DR7[11] = 1 AND IA32_DEBUGCTL[15] = 1, any #DB or #BP detected inside an RTM translation will

- terminate speculative execution,
- set RIP to the address of the XBEGIN instruction, and
- be delivered as #DB (implying an Intel SGX AEX; any #BP is converted to #DB).

- DR6[16] will be cleared, indicating RTM debug (if the #DB causes a VM exit, DR6 is not modified but bit 16 of the pending debug exceptions field in the VMCS will be set).

42.14 INTEL® SGX INTERACTIONS WITH S STATES

Whenever an Intel SGX enabled processor enters S3-S5 state, enclaves are destroyed. This is due to the EPC being destroyed when power down occurs. It is the application runtime’s responsibility to re-instantiate an enclave after a power transition for which the enclaves were destroyed.

...

42.15.2 Machine Check Enables (IA32_MCi_CTL)

All supported IA32_MCi_CTL bits for all the machine check banks must be set for Intel SGX to be available (CPUID.SGX_Leaf.0:EAX[SGX1] == 1). Any act of clearing bits from '1 to '0 in any of the IA32_MCi_CTL register may disable Intel SGX (set CPUID.SGX_Leaf.0:EAX[SGX1] to 0) until the next reset.

...

42.17 INTEL SGX INTERACTION WITH PROTECTION KEYS

SGX interactions with PKRU are as follows:

- CPUID.(EAX=12H, ECX=1):ECX.PKRU indicates whether SECS.ATTRIBUTES.XFRM.PKRU can be set. If SECS.ATTRIBUTES.XFRM.PKRU is set, then PKRU is saved and cleared as part of AEX and is restored as part of ERESUME. If CR4.PKE is set, an enclave can execute RDPKRU and WRKRU independent of whether SECS.ATTRIBUTES.XFRM.PKRU is set.

SGX interactions with domain permission checks are as follows:

1) If CR4.PKE is not set, then legacy and SGX permission checks are not effected.

2) If CR4.PKE is set, then domain permission checks are applied to all non-enclave access and enclave accesses to user pages in addition to legacy and SGX permission checks at a higher priority than SGX permission checks.

3) Implicit accesses aren't subject to domain permission checks.

...

35. Updates to Chapter 43, Volume 3D

Change bars show changes to Chapter 43 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D:* System Programming Guide, Part 4.

...

43.1.1 Debug Enclave vs. Production Enclave

The SECS of each enclave provides a bit, SECS.ATTRIBUTES.DEBUG, indicating whether the enclave is a debug enclave (if set) or a production enclave (if 0). If this bit is set, software outside the enclave can use EDBGRD/

EDBGWR to access the EPC memory of the enclave. The value of DEBUG is not included in the measurement of the enclave and therefore doesn't require an alternate SIGSTRUCT to be generated to debug the enclave.

The ATTRIBUTES field in the SECS is reported in the enclave's attestation, and is included in the key derivation. Enclave secrets that were protected by the enclave using Intel SGX keys when it ran as a production enclave will not be accessible by the debug enclave. A debugger needs to be aware that special debug content might be required for a debug enclave to run in a meaningful way.

EPC memory belonging to a debug enclave can be accessed via the EDBGRD/EDBGWR leaf functions (see Section 41.4), while that belonging to a non-debug enclave cannot be accessed by these leaf functions.

43.1.2 Tool-Chain Opt-in

The TCS.FLAGS.DBGOPTIN bit controls interactions of certain debug and profiling features with enclaves, including code/data breakpoints, TF, RF, monitor trap flag, BTF, LBRs, BTM, BTS, Intel Processor Trace, and performance monitoring. This bit is forced to zero when EPC pages are added via EADD. A debugger can set this bit via EDBGWR to the TCS of a debug enclave.

An enclave entry through a TCS with the TCS.FLAGS.DBGOPTIN set to 0 is called an **opt-out entry**. Conversely, an enclave entry through a TCS with TCS.FLAGS.DBGOPTIN set to 1 is called an **opt-in entry**.

...

43.2.1 Single Stepping ENCLS Instruction Leafs

If the RFLAGS.TF bit is set at the beginning of ENCLS, then a single-step debug exception is pending as a trap-class exception on the instruction boundary immediately after the ENCLS instruction. Additionally, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending on the instruction boundary immediately after the instruction if the instruction does not fault.

43.2.2 Single Stepping ENCLU Instruction Leafs

The interactions of the unprivileged Intel SGX instruction ENCLU are leaf dependent.

An enclave entry via EENTER/ERESUME leaf functions of the ENCLU, in certain cases, may mask the RFLAGS.TF bit, and mask the setting of the “monitor trap flag” VM-execution control. In such situations, an exit from the enclave, either via the EEXIT leaf function or via an AEX unmasks the RFLAGS.TF bit and the “monitor trap flag” VM-execution control. The details of this masking/unmasking and the pending of single stepping events across EENTER/ERESUME/EEXIT/AEX are covered in detail in Section 43.2.3.

If the EFLAGS.TF bit is set at the beginning of EREPORT or EGETKEY leafs, and if the EFLAGS.TF is not masked by the preceding enclave entry, then a single-step debug exception is pending on the instruction boundary immediately after the ENCLU instruction. Additionally, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, and if the monitor trap flag is not masked by the preceding enclave entry, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

If the instruction under consideration results in a fault, then the control flow goes to the fault handler, and no single-step debug exception is asserted. In such a situation, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending after the delivery of the fault (or any nested exception). No MTF VM exit occurs if another VM exit occurs before reaching that boundary on which an MTF VM exit would be pending.

43.2.3 Single-Stepping Enclave Entry with Opt-out Entry

43.2.3.1 Single Stepping without AEX

Figure 40-1 shows the most common case for single-stepping after an opt-out entry.

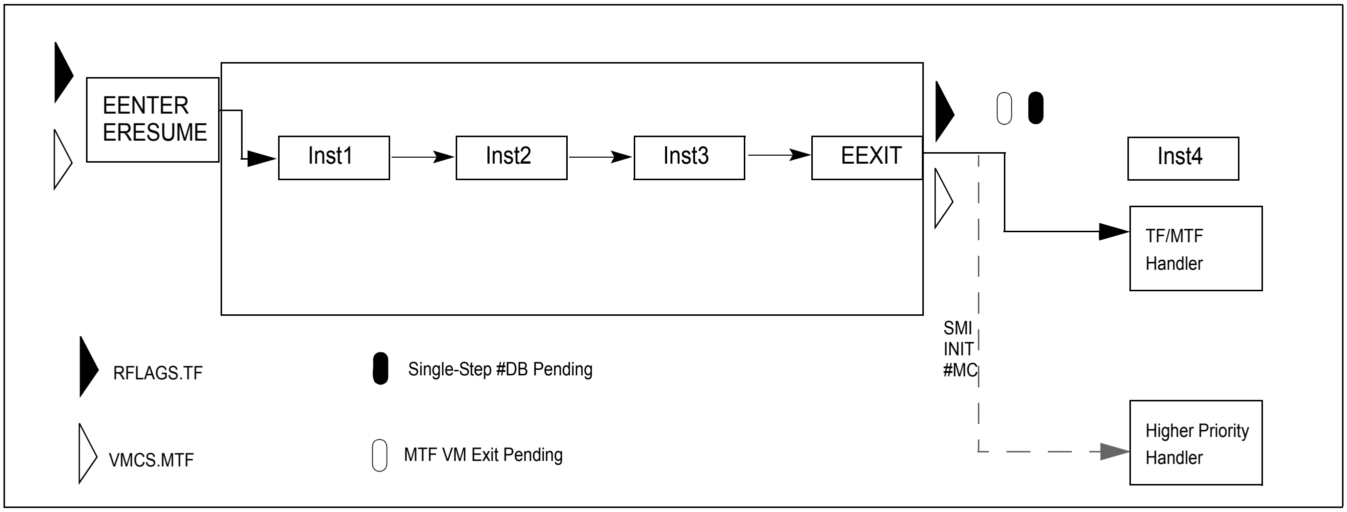


Figure 43-1 Single Stepping with Opt-out Entry - No AEX

In this scenario, if the RFLAGS.TF bit is set at the time of the enclave entry, then a single step debug exception is pending on the instruction boundary after EEXIT. Additionally, if the enclave is executing in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after EEXIT.

The value of the RFLAGS.TF bit at the end of EEXIT is the same as the value of RFLAGS.TF at the time of the enclave entry.

43.2.3.2 Single Step Preempted by AEX Due to Non-SMI Event

Figure 43-2 shows the interaction of single stepping with AEX due to a non-SMI event after an opt-out entry.

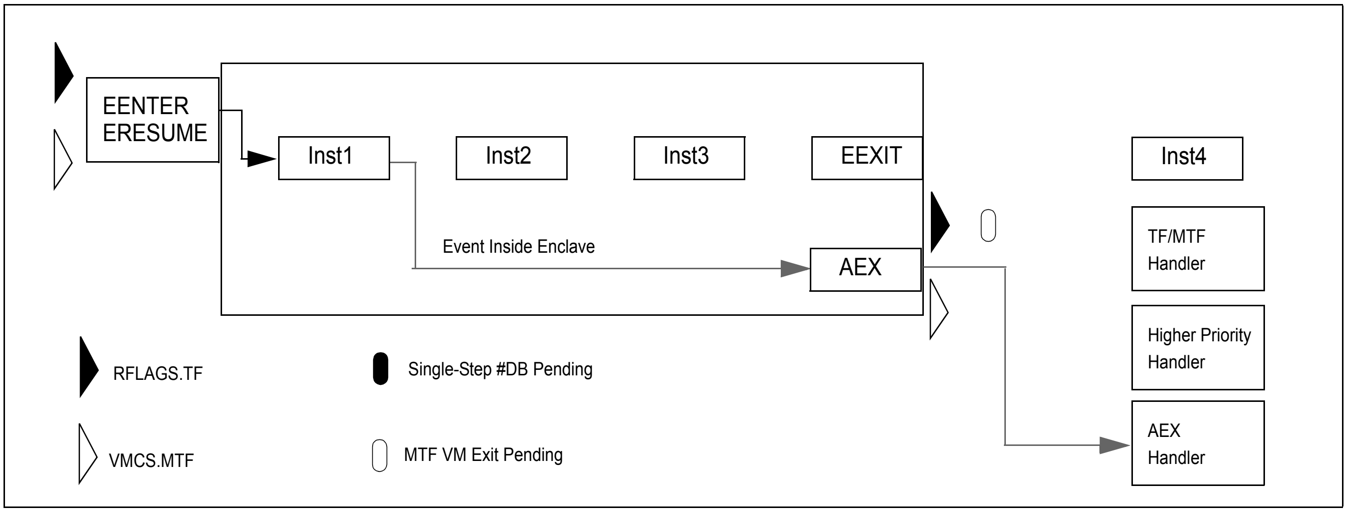


Figure 43-2 Single Stepping with Opt-out Entry -AEX Due to Non-SMI Event Before Single-Step Boundary

In this scenario, if the enclave is executing in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after the AEX. No MTF VM exit occurs if another VM exit happens before reaching that instruction boundary.

The value of the RFLAGS.TF bit at the end of AEX is the same as the value of RFLAGS.TF at the time of the enclave entry.

43.2.4 RFLAGS.TF Treatment on AEX

The value of EFLAGS.TF at the end of AEX from an opt-out enclave is same as the value of EFLAGS.TF at the time of the enclave entry. The value of EFLAGS.TF at the end of AEX from an opt-in enclave is unmodified. The EFLAGS.TF saved in GPR portion of the SSA on an AEX is 0. For more detail see EENTER and ERESUME in Chapter 5.

43.2.5 Restriction on Setting of TF after an Opt-Out Entry

Enclave entered through an opt-out entry is not allowed to set EFLAGS.TF. The POPF instruction forces RFLAGS.TF to 0 if the enclave was entered through opt-out entry.

...

43.3.1 Breakpoint Suppression

Following an opt-out entry:

- Instruction breakpoints are suppressed during execution in an enclave.
- Data breakpoints are not triggered on accesses to the address range defined by ELRANGE.
- Data breakpoints are triggered on accesses to addresses outside the ELRANGE

Following an opt-in entry instruction and data breakpoints are not suppressed.

The processor does not report any matches on debug breakpoints that are suppressed on enclave entry. However, the processor does not clear any bits in DR6 that were already set at the time of the enclave entry.

43.3.2 Reporting of Instruction Breakpoint on Next Instruction on a Debug Trap

A debug exception caused by the single-step execution mode or when a data breakpoint condition was met causes the processor to perform an AEX. Following such an AEX, the processor reports in the debug status register (DR6) matches of the new instruction pointer (the AEP address) in a breakpoint address register setup to detect instruction execution.

43.3.3 RF Treatment on AEX

RF flag value saved in SSA is the same as what would have been pushed on stack if the exception or event causing the AEX occurred when executing outside an enclave (see Section 17.3.1.1). Following an AEX, the RF flag is 0 in the synthetic state.

43.3.4 Breakpoint Matching in Intel® SGX Instruction Flows

Implicit accesses made by Intel SGX instructions to EPC regions do not trigger data breakpoints. Explicit accesses made by ENCLS[ECREATE], ENCLS[EADD], ENCLS[EEXTEND], ENCLS[EINIT], ENCLS[EREMOVE], ENCLS[ETRACK], ENCLS[EBLOCK], ENCLS[EPA], ENCLS[EWB], ENCLS[ELD], ENCLS[EDBGRD], ENCLS[EDBGWR], ENCLU[EENTER], and ENCLU[ERESUME] to the EPC operands do not trigger data breakpoints.

Explicit accesses made by the Intel SGX instructions (ENCLU[EGETKEY] and ENCLU[EREPORT]) executed by an enclave following an opt-in entry, trigger data breakpoints on accesses to their EPC operands. All Intel SGX instructions trigger data breakpoints on accesses to their non-EPC operands.

43.4 INT3 CONSIDERATION

43.4.1 Behavior of INT3 Inside an Enclave

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 RIP saved in the SSA on AEX is that of the INT3 instruction. The RIP saved on the stack (or in the TSS or VMCS) is that of the AEP.

If execution of INT3 in an enclave causes a VM exit, the event type in the VM-exit interruption information field indicates a hardware exception (type 3; not a software exception with type 6) and the VM-exit instruction length field is saved as zero.

43.4.2 Debugger Considerations

The INT3 is fault-like inside an enclave and the RIP saved in the SSA on AEX is that of the INT3 instruction. Consequently, the debugger must not decrement SSA.RIP for #BP coming from an enclave to re-execute the instruction at the RIP of the INT3 instruction on a subsequent enclave entry.

43.4.3 VMM Considerations

As described above, INT3 executed by enclave delivers #BP with “interruption type” of 3. A VMM that re-injects #BP into the guest can obtain the VM entry interruption information from appropriate VMCS fields (as recommended in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*).

VMMs that create the VM-entry interruption information based on the interruption vector should use event type of 3 (instead of 6) when they detect a VM exit incident to enclave mode that is due to an event with vector 3.

43.5 BRANCH TRACING

43.5.1 BTF Treatment

When software enables single-stepping on branches then:

- Following an opt-in entry using EENTER the processor generates a single step debug exception.
- Following an EEXIT the processor generates a single-step debug exception

Enclave entry using ERESUME (opt-in or opt-out) and an AEX from the enclave do not cause generation of the single-step debug exception.

43.5.2 LBR Treatment

43.5.2.1 LBR Stack on Opt-in Entry

Following an opt-in entry into an enclave, last branch recording facilities if enabled continued to store branch records in the LBR stack MSRs as follows:

- On enclave entry using EENTER/ERESUME, the processor push the address of EENTER/ERESUME instruction into MSR_LASTBRANCH_n_FROM_IP, and the destination address of the EENTER/ERESUME into MSR_LASTBRANCH_n_TO_IP.
- On EEXIT, the processor pushes the address of EEXIT instruction into MSR_LASTBRANCH_n_FROM_IP, and the address of EEXIT destination into MSR_LASTBRANCH_n_TO_IP.
- On AEX, the processor pushes RIP saved in the SSA into MSR_LASTBRANCH_n_FROM_IP, and the address of AEP into MSR_LASTBRANCH_n_TO_IP.
- For every branch inside the enclave, a branch record is pushed on the LBR stack.

Figure 43-3 shows an example of LBR stack manipulation after an opt-in entry. Every arrow in this picture indicates a branch record pushed on the LBR stack. The “From IP” of the branch record contains the linear address of the instruction located at the start of the arrow, while the “To IP” of the branch record contains the linear address of the instruction at the end of the arrow.

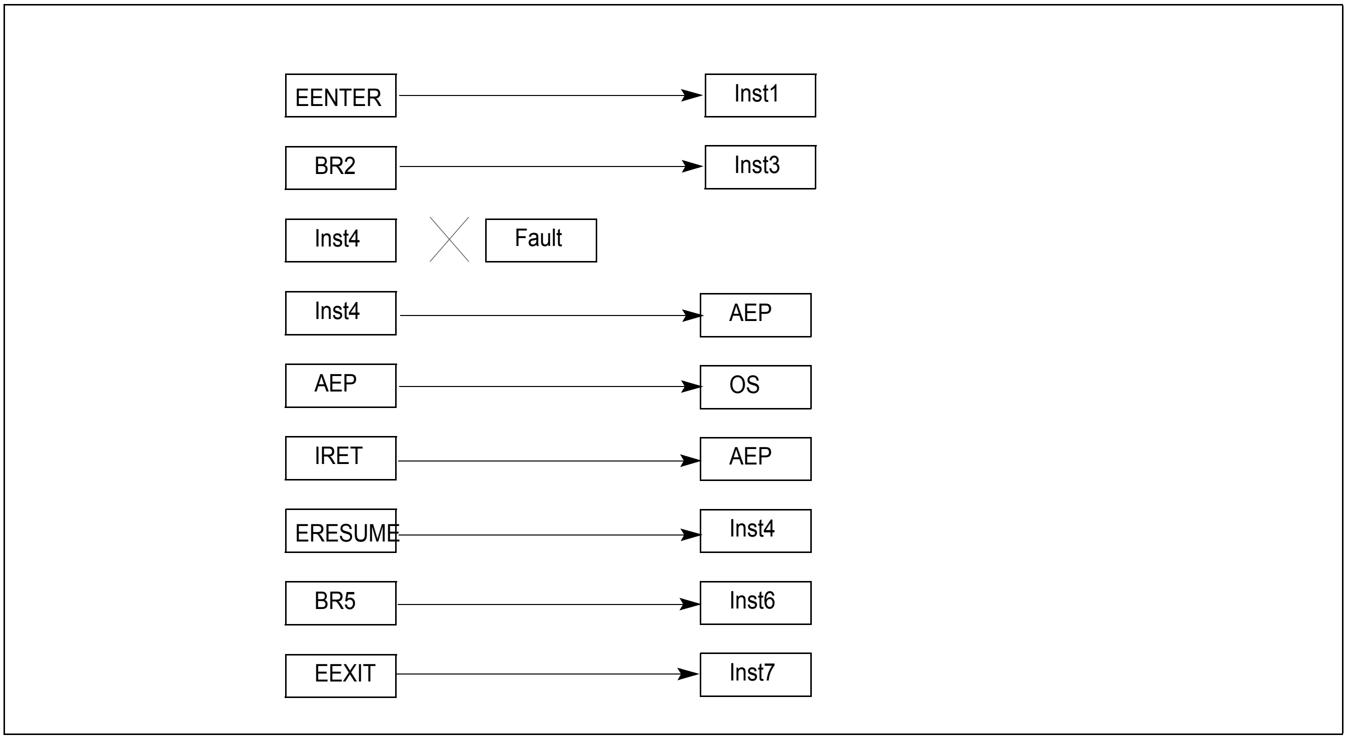


Figure 43-3 LBR Stack Interaction with Opt-in Entry

43.5.2.2 LBR Stack on Opt-out Entry

An opt-out entry into an enclave suppresses last branch recording facilities, and enclave exit after an opt-out entry un-suppresses last branch recording facilities.

Opt-out entry into an enclave does not push any record on LBR stack.

If last branch recording facilities were enabled at the time of enclave entry, then EEXIT following such an enclave entry pushes one record on LBR stack. The MSR_LASTBRANCH_n_FROM_IP of such record holds the linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while the MSR_LASTBRANCH_n_TO_IP of such record holds linear address of the destination of EEXIT.

Additionally, if last branch recording facilities were enabled at the time of enclave entry, then an AEX after such an entry pushes one record on LBR stack, before pushing record for the event causing the AEX if the event pushes a record on LBR stack. The MSR_LASTBRANCH_n_FROM_IP of the new record holds linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while MSR_LASTBRANCH_n_TO_IP of the new record holds linear address of the AEP. If the event causing AEX pushes a record on LBR stack, then the MSR_LASTBRANCH_n_FROM_IP for that record holds linear address of the AEP.

Figure 43-4 shows an example of LBR stack manipulation after an opt-out entry. Every arrow in this picture indicates a branch record pushed on the LBR stack. The “From IP” of the branch record contains the linear address of the instruction located at the start of the arrow, while the “To IP” of the branch record contains the linear address of the instruction at the end of the arrow.

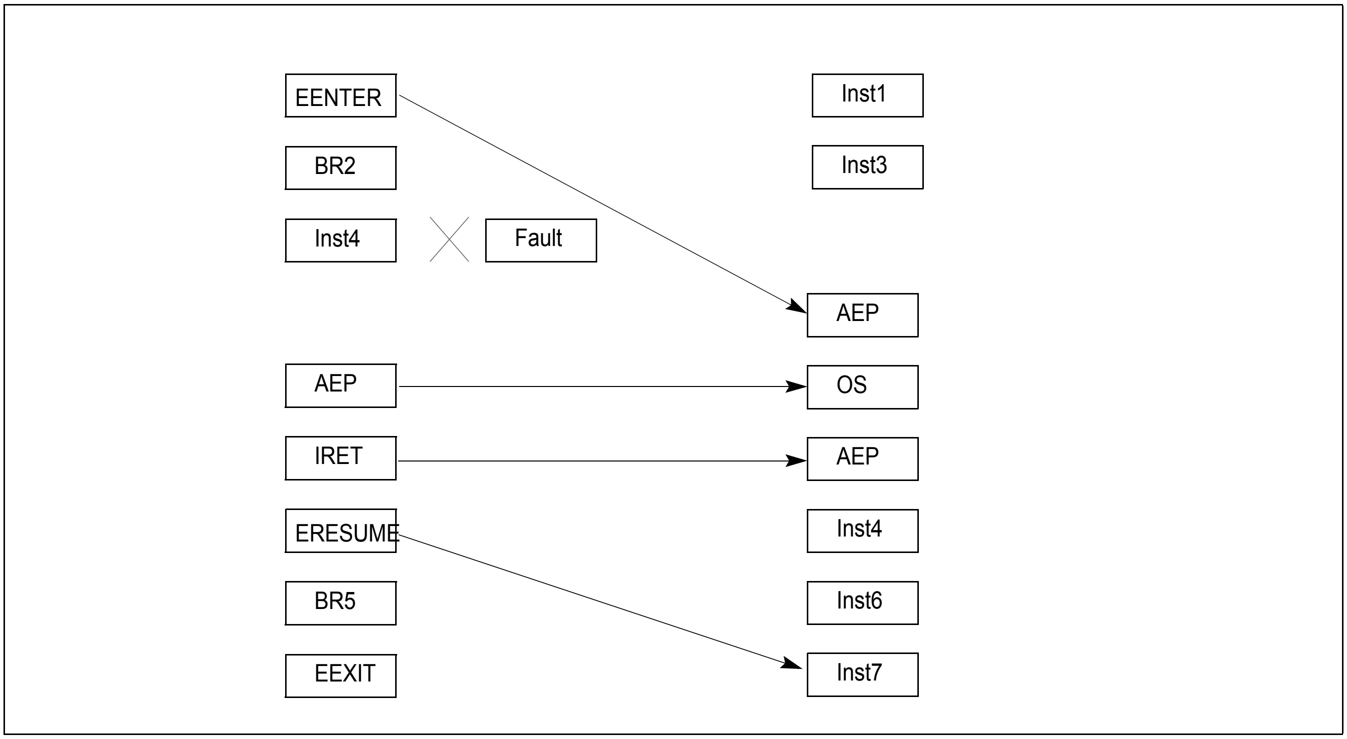


Figure 43-4 LBR Stack Interaction with Opt-out Entry

43.5.2.3 Mispredict Bit, Record Type, and Filtering

All branch records resulting from Intel SGX instructions/AEXs are reported as predicted branches, and consequently, bit 63 of MSR_LASTBRANCH_n_FROM_IP for such records is set. Branch records due to these Intel SGX operations are always non-HLE/non-RTM records.

EENTER, ERESUME, EEXIT, and AEX are considered to be far branches. Consequently, bit 8 in MSR_LBR_SELECT controls filtering of the new records introduced by Intel SGX.

43.6 INTERACTION WITH PERFORMANCE MONITORING

43.6.1 IA32_PERF_GLOBAL_STATUS Enhancement

On processors supporting Intel SGX, the IA32_PERF_GLOBAL_STATUS MSR provides a bit indicator, known as “Anti Side-channel Interference” (ASCI) at bit position 60. If this bit is 0, the performance monitoring data in various performance monitoring counters are accumulated normally as defined by relevant architectural/microarchitectural conditions. If the ASCI bit is set, the contents in various performance monitoring counters can be affected by the direct or indirect consequence of Intel SGX protection of enclave code executing in the processor.

43.6.2 Performance Monitoring with Opt-in Entry

An opt-in enclave entry allow performance monitoring logic to observe the contribution of enclave code executing in the processor. Thus the contents of performance monitoring counters does not distinguish between contribution originating from enclave code or otherwise. All counters, events, precise events, etc. continue to work as defined in the IA32/Intel 64 Software Developer Manual. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is not set.

43.6.3 Performance Monitoring with Opt-out Entry

In general, performance monitoring activities are suppressed when entering an opt-out enclave. This applies to all thread-specific, configured performance monitoring, except for the cycle-counting fixed counter, IA32_FIXED_CTR1 and IA32_FIXED_CTR2. Upon entering an opt-out enclave, IA32_FIXED_CTR0, IA32_PMCx will stop accumulating counts. Additionally, if PEBS is configured to capture PEBS record for this thread, PEBS record generation will also be suppressed. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is set.

Performance monitoring on the sibling thread may also be affected. Any one of IA32_FIXED_CTRx or IA32_PMCx on the sibling thread configured to monitor thread-specific eventing logic with AnyThread =1 is demoted to count only MyThread while an opt-out enclave is executing on the other thread.

...

43.6.5 PEBS Record Generation on Intel® SGX Instructions

All leaf functions of the ENCLS instruction report "Eventing RIP" of the ENCLS instruction if a PEBS record is generated at the end of the instruction execution. Additionally, the EGETKEY and EREPORT leaf functions of the ENCLU instruction report "Eventing RIP" of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution.

If the EENTER and ERESUME leaf functions are performing an opt-in entry report "Eventing RIP" of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution. On the other hand, if these leaf functions are performing an opt-out entry, then these leaf functions result in PEBS being suppressed, and no PEBS record is generated at the end of these instructions.

A PEBS record is generated if there is a PEBS event pending at the end of EEXIT (due to a counter overflowing during enclave execution or during EEXIT execution). This PEBS record contains the architectural state of the logical processor at the end of EEXIT. If the enclave was entered via an opt-in entry, then this record reports the "Eventing RIP" as the linear address of the ENCLU[EEXIT] instruction. If the enclave was entered via an opt-out entry, then the record reports the "Eventing RIP" as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A PEBS record is generated after the AEX if there is a PEBS event pending at the end of AEX (due to a counter overflowing during enclave execution or during AEX execution). This PEBS record contains the synthetic state of the logical processor that is established at the end of AEX. For opt-in entry, this record has the EVENTING_RIP set to the RIP saved in the SSA. For opt-out entry, the record has the EVENTING_RIP set to the linear address of EENTER/ERESUME used for the last enclave entry.

If the enclave was entered via an opt-in entry, then this record reports the "Eventing RIP" as the linear address in the SSA of the enclave (a.k.a., the "Eventing LIP" inside the enclave). If the enclave was entered via an opt-out entry, then the record reports the "Eventing RIP" as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A second PEBS event may be pended during the Enclave Exiting Event (EEE). If the PEBS event is taken at the end of delivery of the EEE then the "Eventing RIP" in this second PEBS record is the linear address of the AEP.

43.6.6 Exception-Handling on PEBS/BTS Loads/Stores after AEX

The operating system should allocate sections of the DS save area from a non-paged pool, and mark them as accessed and dirty. If the loads/stores to any section of the DS save area incur faults then such faults are reported to the OS/VMM immediately, and generation of the PEBS/BTS record is skipped and may leave the buffers in a state where they have a partial PEBS or BTS records.

However, any events that are detected during PEBS/BTS record generation at the end of AEX and before delivering the Enclave Exiting Event (EEE) cannot be reported immediately to the OS/VMM, as an event window is not open at the end of AEX. Consequently, fault-like events such as page faults, EPT faults, EPT mis-configuration, and accesses to APIC-access page detected on stores to the PEBS/BTS buffer are not reported, and generation of the PEBS and/or BTS record at the end of AEX is aborted (this may leave the buffers in a state where they have partial PEBS or BTS records). Trap-like events detected on stores to the PEBS/BTS buffer (such as debug traps) are pended until the next instruction boundary, where they are handled according to the architecturally defined priority. The processor continues the handling of the Enclave Exiting Event (SMI, NMI, interrupt, exception delivery, VM exit, etc.) after aborting the PEBS/BTS record generation.

43.6.6.1 Other Interactions with Performance Monitoring

For opt-in entry, EENTER, ERESUME, EEXIT, and AEX are all treated as predicted far branches, and any counters that are counting such branches are incremented by 1 as a part of retirement of these instructions. Retirement of these instructions is also counted in any counters configured to count instructions retired.

For opt-out entry, execution inside an enclave is treated as a single predicted branch, and all branch-counting performance monitoring counters are incremented accordingly. Additionally, such execution is also counted as a single instruction, and all performance monitoring counters counting instructions are incremented accordingly.

Enclave entry does not affect any performance monitoring counters shared between cores.

...

36. Updates to Appendix A, Volume 3D

Change bars show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

A.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:

- Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.
- If bit 5 is read as 1, VM exits store the value of IA32_EFER.LMA into the "IA-32e mode guest" VM-entry control; see Section 27.2 for more details. This bit is read as 1 on any logical processor that supports the 1-setting of the "unrestricted guest" VM-execution control.
- Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
  - — Bit 6 reports (if set) the support for activity state 1 (HLT).
  - — Bit 7 reports (if set) the support for activity state 2 (shutdown).
  - — Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

  If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. All implementations support VM entry to activity state 0 (active).
- If bit 14 is read as 1, Intel® Processor Trace (Intel PT) can be used in VMX operation. If the processor supports Intel PT but does not allow it to be used in VMX operation, execution of VMXON clears IA32_RTIT_CTL.TraceEn (see "VMXON—Enter VMX Operation" in Chapter 30); any attempt to set that bit while in VMX operation (including VMX root operation) using the WRMSR instruction causes a general-protection exception.

- If bit 15 is read as 1, the RDMSR instruction can be used in system-management mode (SMM) to read the IA32_SMBASE MSR (MSR address 9EH). See Section 34.15.6.4.
- Bits 24:16 indicate the number of CR3-target values supported by the processor. This number is a value between 0 and 256, inclusive (bit 24 is set if and only if bits 23:16 are clear).
- Bits 27:25 is used to compute the recommended maximum number of MSRs that should appear in the VM-exit MSR-store list, the VM-exit MSR-load list, or the VM-entry MSR-load list. Specifically, if the value bits 27:25 of IA32_VMX_MISC is N, then 512 * (N + 1) is the recommended maximum number of MSRs to be included in each list. If the limit is exceeded, undefined processor behavior may result (including a machine check during the VMX transition).
- If bit 28 is read as 1, bit 2 of the IA32_SMM_MONITOR_CTL can be set to 1. VMXOFF unblocks SMIs unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 34.14.4).
- If bit 29 is read as 1, software can use VMWRITE to write to any supported field in the VMCS; otherwise, VMWRITE cannot be used to modify VM-exit information fields.
- If bit 30 is read as 1, VM entry allows injection of a software interrupt, software exception, or privileged software exception with an instruction length of 0.
- Bits 63:32 report the 32-bit MSEG revision identifier used by the processor.
- Bits 13:9 and bit 31 are reserved and are read as 0.

...

37. Updates to Appendix B, Volume 3D

Change bars show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3D:* System Programming Guide, Part 4.

...

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
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)		00002009H

Table B-4   Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
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
TSC multiplier (full)[14]	000011001B	00002032H
TSC multiplier (high)[14]		00002033H

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 either the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support either 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 “use TSC scaling” 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

Table B-6 Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
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

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.

...

38. Updates to Appendix C, Volume 3D

Change bars show changes to Appendix C of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

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 26.7). 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. 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1. This case includes executions of BOUND that cause #BR, executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD2 (they cause #UD).
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 34.15.2).
6	**Other SMI.** An SMI arrived and caused an SMM VM exit (see Section 34.15.2) 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 34.15.2).
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.
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.

Table C-1 Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
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.
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 entry occurred due to the 1-setting of the “monitor trap flag” VM-execution control and injection of an MTF VM exit as part of VM entry. See Section 25.5.2.
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 26.8).
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 26.6.7).
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	**INVVPID.** Guest software attempted to execute INVVPID.
54	**WBINVD.** Guest software attempted to execute WBINVD 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 and “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.

...