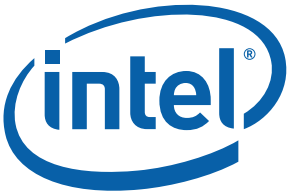


Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

September 2015

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: 252046-048

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

Revision	Description	Date
-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
-048	• Removed Documentation Changes 1-19 • Add Documentation Changes 1-33	September 2015

§

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 5, Volume 1
3	Updates to Chapter 8, Volume 1
4	Updates to Chapter 13, Volume 1
5	Updates to Chapter 15, Volume 1
6	Updates to Chapter 1, Volume 2A
7	Updates to Chapter 3, Volume 2A
8	Updates to Chapter 4, Volume 2B
9	Updates to Appendix B, Volume 2C
10	Updates to Chapter 1, Volume 3A
11	Updates to Chapter 2, Volume 3A
12	Updates to Chapter 4, Volume 3A
13	Updates to Chapter 6, Volume 3A
14	Updates to Chapter 8, Volume 3A
15	Updates to Chapter 11, Volume 3A
16	Updates to Chapter 15, Volume 3B
17	Updates to Chapter 18, Volume 3B
18	Updates to Chapter 19, Volume 3B
19	Updates to Chapter 24, Volume 3C
20	Updates to Chapter 25, Volume 3C
21	Updates to Chapter 26, Volume 3C
22	Updates to Chapter 27, Volume 3C
23	Updates to Chapter 29, Volume 3C
24	Updates to Chapter 35, Volume 3C
25	Updates to Chapter 36, Volume 3C
26	New Chapter 37, New Volume 3D
27	New Chapter 38, New Volume 3D

Documentation Changes(Sheet 2 of 2)

No.	DOCUMENTATION CHANGES
28	New Chapter 39, New Volume 3D
29	New Chapter 40, New Volume 3D
30	New Chapter 41, New Volume 3D
31	New Chapter 42, New Volume 3D
32	New Chapter 43, New Volume 3D
33	Updates to Appendix B, New 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 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™ processor family
- 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® Atom™ processor 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™ processor family is 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 32nm 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 and 5th generation Intel® Core™ processors 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.

...

2. Updates to Chapter 5, Volume 1

Change bars show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

This chapter provides an abridged overview of Intel 64 and IA-32 instructions. Instructions are divided into the following groups:

- General purpose
- x87 FPU
- x87 FPU and SIMD state management
- Intel® MMX technology

- SSE extensions
- SSE2 extensions
- SSE3 extensions
- SSSE3 extensions
- SSE4 extensions
- AESNI and PCLMULQDQ
- Intel® AVX extensions
- F16C, RDRAND, RDSEED, FS/GS base access
- FMA extensions
- Intel® AVX2 extensions
- Intel® Transactional Synchronization extensions
- System instructions
- IA-32e mode: 64-bit mode instructions
- VMX instructions
- SMX instructions
- ADCX and ADOX
- Intel® Memory Protection Extensions
- Intel® Security Guard Extensions

Table 5-1 lists the groups and IA-32 processors that support each group. More recent instruction set extensions are listed in Table 5-2. Within these groups, most instructions are collected into functional subgroups.

Table 5-1. Instruction Groups in Intel 64 and IA-32 Processors

Instruction Set Architecture	Intel 64 and IA-32 Processor Support
General Purpose	All Intel 64 and IA-32 processors
x87 FPU	Intel486, Pentium, Pentium with MMX Technology, Celeron, Pentium Pro, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors
x87 FPU and SIMD State Management	Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors
MMX Technology	Pentium with MMX Technology, Celeron, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors
SSE Extensions	Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors
SSE2 Extensions	Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors
SSE3 Extensions	Pentium 4 supporting HT Technology (built on 90nm process technology), Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Xeon processor 3xxxx, 5xxx, 7xxx Series, Intel Atom processors
SSSE3 Extensions	Intel Xeon processor 3xxx, 5100, 5200, 5300, 5400, 5500, 5600, 7300, 7400, 7500 series, Intel Core 2 Extreme processors QX6000 series, Intel Core 2 Duo, Intel Core 2 Quad processors, Intel Pentium Dual-Core processors, Intel Atom processors

Table 5-1. Instruction Groups in Intel 64 and IA-32 Processors (Contd.)

Instruction Set Architecture	Intel 64 and IA-32 Processor Support
IA-32e mode: 64-bit mode instructions	Intel 64 processors
System Instructions	Intel 64 and IA-32 processors
VMX Instructions	Intel 64 and IA-32 processors supporting Intel Virtualization Technology
SMX Instructions	Intel Core 2 Duo processor E6x50, E8xxx; Intel Core 2 Quad processor Q9xxx

Table 5-2. Recent Instruction Set Extensions Introduction in Intel 64 and IA-32 Processors

Instruction Set Architecture	Processor Generation Introduction
SSE4.1 Extensions	Intel Xeon processor 3100, 3300, 5200, 5400, 7400, 7500 series, Intel Core 2 Extreme processors QX9000 series, Intel Core 2 Quad processor Q9000 series, Intel Core 2 Duo processors 8000 series, T9000 series.
SSE4.2 Extensions, CRC32, POPCNT	Intel Core i7 965 processor, Intel Xeon processors X3400, X3500, X5500, X6500, X7500 series.
AESNI, PCLMULQDQ	IntelL Xeon processor E7 series, Intel Xeon processors X3600, X5600, Intel Core i7 980X processor; Use CPUID to verify presence of AESNI and PCLMULQDQ across Intel Core processor families.
Intel AVX	Intel Xeon processor E3 and E5 families; 2nd Generation Intel Core i7, i5, i3 processor 2xxx families.
F16C, RDRAND, FS/GS base access	3rd Generation Intel Core processors, Intel Xeon processor E3-1200 v2 product family, Next Generation Intel Xeon processors, Intel Xeon processor E5 v2 and E7 v2 families.
FMA, AVX2, BMI1, BMI2, INVPCID	Intel Xeon processor E3-1200 v3 product family; 4th Generation Intel Core processor family.
TSX	Intel Xeon processor E7 v3 product family
ADX, RDSEED, CLAC, STAC	Intel Core M processor family; 5th Generation Intel Core processor family.
CLFLUSHOPT, XSAVEC, XSAVES, MPX, SGX1	6th Generation Intel Core processor family.

The following sections list instructions in each major group and subgroup. Given for each instruction is its mnemonic and descriptive names. When two or more mnemonics are given (for example, CMOVA/CMOVNBE), they represent different mnemonics for the same instruction opcode. Assemblers support redundant mnemonics for some instructions to make it easier to read code listings. For instance, CMOVA (Conditional move if above) and CMOVNBE (Conditional move if not below or equal) represent the same condition. For detailed information about specific instructions, see the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B & 2C.*

...

5.1.13 Miscellaneous Instructions

The miscellaneous instructions provide such functions as loading an effective address, executing a "no-operation," and retrieving processor identification information.

LEA Load effective address

NOP	No operation
UD2	Undefined instruction
XLAT/XLATB	Table lookup translation
CPUID	Processor identification
MOVBE[1]	Move data after swapping data bytes
PREFETCHW	Prefetch data into cache in anticipation of write
PREFETCHWT1	Prefetch hint T1 with intent to write
CLFLUSH	Flushes and invalidates a memory operand and its associated cache line from all levels of the processor's cache hierarchy
CLFLUSHOPT	Flushes and invalidates a memory operand and its associated cache line from all levels of the processor's cache hierarchy with optimized memory system throughput.

5.1.14 User Mode Extended Sate Save/Restore Instructions

XSAVE	Save processor extended states to memory
XSAVEC	Save processor extended states with compaction to memory
XSAVEOPT	Save processor extended states to memory, optimized
XRSTOR	Restore processor extended states from memory
XGETBV	Reads the state of an extended control register

5.1.15 Random Number Generator Instructions

RDRAND	Retrieves a random number generated from hardware
RDSEED	Retrieves a random number generated from hardware

...

5.6.4 SSE2 Cacheability Control and Ordering Instructions

SSE2 cacheability control instructions provide additional operations for caching of non-temporal data when storing data from XMM registers to memory. LFENCE and MFENCE provide additional control of instruction ordering on store operations.

CLFLUSH	See Section 5.1.13
LFENCE	Serializes load operations
MFENCE	Serializes load and store operations
PAUSE	Improves the performance of "spin-wait loops"
MASKMOVDQU	Non-temporal store of selected bytes from an XMM register into memory
MOVNTPD	Non-temporal store of two packed double-precision floating-point values from an XMM register into memory
MOVNTDQ	Non-temporal store of double quadword from an XMM register into memory
MOVNTI	Non-temporal store of a doubleword from a general-purpose register into memory

...

1. Processor support of MOVBE is enumerated by CPUID.01:ECX.MOVBE[bit 22] = 1

5.18 SYSTEM INSTRUCTIONS

The following system instructions are used to control those functions of the processor that are provided to support for operating systems and executives.

CLAC	Clear AC Flag in EFLAGS register
STAC	Set AC Flag in EFLAGS register
LGDT	Load global descriptor table (GDT) register
SGDT	Store global descriptor table (GDT) register
LLDT	Load local descriptor table (LDT) register
SLDT	Store local descriptor table (LDT) register
LTR	Load task register
STR	Store task register
LIDT	Load interrupt descriptor table (IDT) register
SIDT	Store interrupt descriptor table (IDT) register
MOV	Load and store control registers
LMSW	Load machine status word
SMSW	Store machine status word
CLTS	Clear the task-switched flag
ARPL	Adjust requested privilege level
LAR	Load access rights
LSL	Load segment limit
VERR	Verify segment for reading
VERW	Verify segment for writing
MOV	Load and store debug registers
INVD	Invalidate cache, no writeback
WBINVD	Invalidate cache, with writeback
INVLPG	Invalidate TLB Entry
INVPCID	Invalidate Process-Context Identifier
LOCK (prefix)	Lock Bus
HLT	Halt processor
RSM	Return from system management mode (SMM)
RDMSR	Read model-specific register
WRMSR	Write model-specific register
RDPMC	Read performance monitoring counters
RDTSC	Read time stamp counter
RDTSCP	Read time stamp counter and processor ID
SYSENTER	Fast System Call, transfers to a flat protected mode kernel at CPL = 0
SYSEXIT	Fast System Call, transfers to a flat protected mode kernel at CPL = 3
XSAVE	Save processor extended states to memory
XSAVEC	Save processor extended states with compaction to memory
XSAVEOPT	Save processor extended states to memory, optimized
XSAVES	Save processor supervisor-mode extended states to memory
XRSTOR	Restore processor extended states from memory
XRSTORS	Restore processor supervisor-mode extended states from memory

XGETBV	Reads the state of an extended control register
XSETBV	Writes the state of an extended control register
RDFSBASE	Reads from FS base address at any privilege level
RDGSBASE	Reads from GS base address at any privilege level
WRFSBASE	Writes to FS base address at any privilege level

WRGSBASEWrites to GS base address at any privilege level

...

5.22 INTEL® MEMORY PROTECTION EXTENSIONS

Intel Memory Protection Extensions (MPX) provides a set of instructions to enable software to add robust bounds checking capability to memory references. Details of Intel MPX are described in Chapter 16, "Intel® MPX".

BNDMK	Create a LowerBound and a UpperBound in a register.
BNDCL	Check the address of a memory reference against a LowerBound.
BNDCU	Check the address of a memory reference against an UpperBound in 1's compliment form.
BNDCN	Check the address of a memory reference against an UpperBound not in 1's compliment form.
BNDMOV	Copy or load from memory of the LowerBound and UpperBound to a register.
BNDMOV	Store to memory of the LowerBound and UpperBound from a register.
BNDLDX	Load bounds using address translation.
BNDSTX	Store bounds using address translation.

5.23 INTEL® SECURITY GUARD EXTENSIONS

Intel Security Guard Extensions (SGX) provide two sets of instruction leaf functions to enable application software to instantiate a protected container, referred to as an enclave. The enclave instructions are organized as leaf functions under two instruction mnemonics: ENCLS (ring 0) and ENCLU (ring 3). Details of Intel SGX are described in CHAPTER 37 through CHAPTER 43 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D*.

The first implementation of Intel SGX is also referred to as SGX1, it is introduced with the 6th Generation Intel Core Processors. The leaf functions supported in SGX1 is shown in Table 5-3.

Table 5-3. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EADD]	Add a page	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 by debugger	ENCLU[EREPORT]	Create a cryptographic report
ENCLS[EDBGWR]	Write data 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 as blocked		

Table 5-3. 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 as unblocked		
ENCLS[EPA]	Add version array		
ENCLS[EREMOVE]	Remove a page from EPC		
ENCLS[ETRACK]	Activate EBLOCK checks		
ENCLS[EWB]	Write back/invalidate an EPC page		

...

3. Updates to Chapter 8, Volume 1

Change bars show changes to Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

8.1.8 x87 FPU Instruction and Data (Operand) Pointers

The x87 FPU stores pointers to the instruction and data (operand) for the last non-control instruction executed. These are the x87 FPU instruction pointer and x87 FPU data (operand) pointers; software can save these pointers to provide state information for exception handlers. The pointers are illustrated in Figure 8-1 (the figure illustrates the pointers as used outside 64-bit mode; see below).

Note that the value in the x87 FPU data pointer register is always a pointer to a memory operand. If the last non-control instruction that was executed did not have a memory operand, the value in the data pointer register is undefined (reserved). If CPUID.(EAX=07H,ECX=0H):EBX[bit 6] = 1, the data pointer register is updated only for x87 non-control instructions that incur unmasked x87 exceptions.

The contents of the x87 FPU instruction and data pointer registers remain unchanged when any of the following instructions are executed: FCLEX/FNCLEX, FLDCW, FSTCW/FNSTCW, FSTSW/FNSTSW, FSTENV/FNSTENV, FLDENV, and WAIT/FWAIT.

For all the x87 FPUs and NPXs except the 8087, the x87 FPU instruction pointer points to any prefixes that preceded the instruction. For the 8087, the x87 FPU instruction pointer points only to the actual opcode.

The x87 FPU instruction and data pointers each consists of an offset and a segment selector. On processors that support IA-32e mode, each offset comprises 64 bits; on other processors, each offset comprises 32 bits. Each segment selector comprises 16 bits.

The pointers are accessed by the FINIT/FNINIT, FLDENV, FRSTOR, FSAVE/FNSAVE, FSTENV/FNSTENV, FXRSTOR, FXSAVE, XRSTOR, XSAVE, and XSAVEOPT instructions as follows:

- FINIT/FNINIT. Each instruction clears each 64-bit offset and 16-bit segment selector.
- FLDENV, FRSTOR. These instructions use the memory formats given in Figures 8-9 through 8-12:
  - For each 64-bit offset, each instruction loads the lower 32 bits from memory and clears the upper 32 bits.
  - If CR0.PE = 1, each instruction loads each 16-bit segment selector from memory; otherwise, it clears each 16-bit segment selector.
- FSAVE/FNSAVE, FSTENV/FNSTENV. These instructions use the memory formats given in Figures 8-9 through 8-12.

— Each instruction saves the lower 32 bits of each 64-bit offset into memory. the upper 32 bits are not saved.

— If CR0.PE = 1, each instruction saves each 16-bit segment selector into memory. If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 1, the processor deprecates the segment selectors of the x87 FPU instruction and data pointers; it saves each segment selector as 0000H.

— After saving these data into memory, FSAVE/FNSAVE clears each 64-bit offset and 16-bit segment selector.

- FXRSTOR, XRSTOR. These instructions load data from a memory image whose format depend on operating mode and the REX prefix. The memory formats are given in Tables 3-52, 3-55, and 3-56 in Chapter 3, "Instruction Set Reference, A-M," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

  — Outside of 64-bit mode or if REX.W = 0, the instructions operate as follows:

    - For each 64-bit offset, each instruction loads the lower 32 bits from memory and clears the upper 32 bits.
    - Each instruction loads each 16-bit segment selector from memory.

  — In 64-bit mode with REX.W = 1, the instructions operate as follows:

    - Each instruction loads each 64-bit offset from memory.
    - Each instruction clears each 16-bit segment selector.

- FXSAVE, XSAVE, and XSAVEOPT. These instructions store data into a memory image whose format depend on operating mode and the REX prefix. The memory formats are given in Tables 3-52, 3-55, and 3-56 in Chapter 3, "Instruction Set Reference, A-M," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

  — Outside of 64-bit mode or if REX.W = 0, the instructions operate as follows:

    - Each instruction saves the lower 32 bits of each 64-bit offset into memory. The upper 32 bits are not saved.
    - Each instruction saves each 16-bit segment selector into memory. If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 1, the processor deprecates the segment selectors of the x87 FPU instruction and data pointers; it saves each segment selector as 0000H.

  — In 64-bit mode with REX.W = 1, each instruction saves each 64-bit offset into memory. The 16-bit segment selectors are not saved.

8.1.9 Last Instruction Opcode

The x87 FPU stores in the 11-bit x87 FPU opcode register (FOP) the opcode of the last x87 non-control instruction executed that incurred an unmasked x87 exception. (This information provides state information for exception handlers.) Only the first and second opcode bytes (after all prefixes) are stored in the x87 FPU opcode register. Figure 8-8 shows the encoding of these two bytes. Since the upper 5 bits of the first opcode byte are the same for all floating-point opcodes (11011B), only the lower 3 bits of this byte are stored in the opcode register.

8.1.9.1 Fopcode Compatibility Sub-mode

Some Pentium 4 and Intel Xeon processors provide program control over the value stored into FOP. Here, bit 2 of the IA32_MISC_ENABLE MSR enables (set) or disables (clear) the fopcode compatibility mode.

If fopcode compatibility mode is enabled, FOP is defined as it had been in previous IA-32 implementations, as the opcode of the last x87 non-control instruction executed (even if that instruction did not incur an unmasked x87 exception).

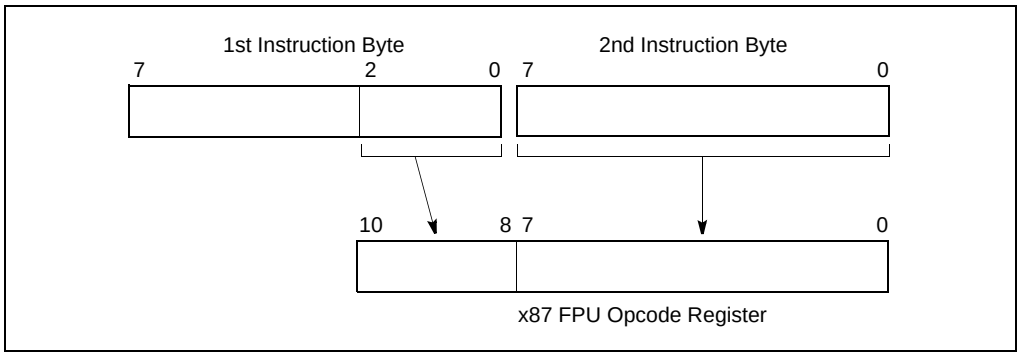


Figure 8-1. Contents of x87 FPU Opcode Registers

The fopcode compatibility mode should be enabled only when x87 FPU floating-point exception handlers are designed to use the fopcode to analyze program performance or restart a program after an exception has been handled.

More recent Intel 64 processors do not support fopcode compatibility mode and do not allow software to set bit 2 of the IA32_MISC_ENABLE MSR.

...

4. Updates to Chapter 13, Volume 1

Change bars show changes to Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

13.3 ENABLING THE XSAVE FEATURE SET AND XSAVE-ENABLED FEATURES

Software enables the XSAVE feature set by setting CR4.OSXSAVE[bit 18] to 1 (e.g., with the MOV to CR4 instruction). If this bit is 0, execution of any of XGETBV, XRSTOR, XRSTORS, XSAVE, XSAVEC, XSAVEOPT, XSAVES, and XSETBV causes an invalid-opcode exception (#UD).

When CR4.OSXSAVE = 1 and CPL = 0, executing the XSETBV instruction with ECX = 0 writes the 64-bit value in EDX:EAX to XCR0 (EAX is written to XCR0[31:0] and EDX to XCR0[63:32]). (Execution of the XSETBV instruction causes a general-protection fault — #GP — if CPL > 0.) The following items provide details regarding individual bits in XCR0:

- XCR0[0] is associated with x87 state (see Section 13.5.1). XCR0[0] is always 1. It has that value coming out of RESET. Executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and EAX[0] is 0.
- XCR0[1] is associated with SSE state (see Section 13.5.2). Software can use the XSAVE feature set to manage SSE state only if XCR0[1] = 1. The value of XCR0[1] in no way determines whether software can execute SSE instructions (these instructions can be executed even if XCR0[1] = 0).

  XCR0[1] is 0 coming out of RESET. As noted in Section 13.2, every processor that supports the XSAVE feature set allows software to set XCR0[1].
- XCR0[2] is associated with AVX state (see Section 13.5.3). Software can use the XSAVE feature set to manage AVX state only if XCR0[2] = 1. In addition, software can execute AVX instructions only if CR4.OSXSAVE = XCR0[2] = 1. Otherwise, any execution of an AVX instruction causes an invalid-opcode exception (#UD).

XCR0[2] is 0 coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[2] if and only if CPUID.(EAX=0DH,ECX=0):EAX[2] = 1. In addition, executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and EAX[2:1] has the value 10b; that is, software cannot enable the XSAVE feature set for AVX state but not for SSE state.

As noted in Section 13.1, the processor will preserve AVX state unmodified if software clears XCR0[2]. However, clearing XCR0[2] while AVX state is not in its initial configuration may cause SSE instructions to incur a power and performance penalty. See Section 13.5.3, "Enable the Use Of XSAVE Feature Set And XSAVE State Components" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for how system software can avoid this penalty.

- XCR0[4:3] are associated with MPX state (see Section 13.5.4). Software can use the XSAVE feature set to manage MPX state only if XCR0[4:3] = 11b. In addition, software can execute MPX instructions only if CR4.OSXSAVE = 1 and XCR0[4:3] = 11b. Otherwise, any execution of an MPX instruction causes an invalid-opcode exception (#UD).[1]

  XCR0[4:3] have value 00b coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[4:3] to 11b if and only if CPUID.(EAX=0DH,ECX=0):EAX[4:3] = 11b. In addition, executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0, EAX[4:3] is neither 00b nor 11b; that is, software can enable the XSAVE feature set for MPX state only if it does so for both state components.

  As noted in Section 13.1, the processor will preserve MPX state unmodified if software clears XCR0[4:3].

- XCR0[7:5] are associated with AVX-512 state (see Section 13.5.5). Software can use the XSAVE feature set to manage AVX-512 state only if XCR0[7:5] = 111b. In addition, software can execute AVX-512 instructions only if CR4.OSXSAVE = 1 and XCR0[7:5] = 111b. Otherwise, any execution of an AVX-512 instruction causes an invalid-opcode exception (#UD).

  XCR0[7:5] have value 000b coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[7:5] to 111b if and only if CPUID.(EAX=0DH,ECX=0):EAX[7:5] = 111b. In addition, executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0, EAX[7:5] is not 000b, and any bit is clear in EAX[2:1] or EAX[7:5]; that is, software can enable the XSAVE feature set for AVX-512 state only if it does so for all three state components, and only if it also does so for AVX state and SSE state. This implies that the value of XCR0[7:5] is always either 000b or 111b.

  As noted in Section 13.1, the processor will preserve AVX-512 state unmodified if software clears XCR0[7:5]. However, clearing XCR0[7:5] while AVX-512 state is not in its initial configuration may cause SSE and AVX instructions to incur a power and performance penalty. See Section 13.5.3, "Enable the Use Of XSAVE Feature Set And XSAVE State Components" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for how system software can avoid this penalty.

- XCR0[9] is associated with PKRU state (see Section 13.5.7). Software can use the XSAVE feature set to manage PKRU state only if XCR0[9] = 1. The value of XCR0[9] in no way determines whether software can use protection keys or execute other instructions that access PKRU state (these instructions can be executed even if XCR0[9] = 0).

  XCR0[9] is 0 coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[9] if and only if CPUID.(EAX=0DH,ECX=0):EAX[9] = 1.

- XCR0[63:10] and XCR0[8] are reserved.[2] Executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and any corresponding bit in EDX:EAX is not 0. These bits in XCR0 are all 0 coming out of RESET.

Software operating with CPL > 0 may need to determine whether the XSAVE feature set and certain XSAVE-enabled features have been enabled. If CPL > 0, execution of the MOV from CR4 instruction causes a general-protection fault (#GP). The following alternative mechanisms allow software to discover the enabling of the XSAVE feature set regardless of CPL:

1. If XCR0[3] = 0, executions of CALL, RET, JMP, and Jcc do not initialize the bounds registers.
2. Bit 8 corresponds to a supervisor state component. Since bits can be set in XCR0 only for user state components, that bit of XCR0 must be 0.

- The value of CR4.OSXSAVE is returned in CPUID.1:ECX.OSXSAVE[bit 27]. If software determines that CPUID.1:ECX.OSXSAVE = 1, the processor supports the XSAVE feature set and the feature set has been enabled in CR4.
- Executing the XGETBV instruction with ECX = 0 returns the value of XCR0 in EDX:EAX. XGETBV can be executed if CR4.OSXSAVE = 1 (if CPUID.1:ECX.OSXSAVE = 1), regardless of CPL.

Thus, software can use the following algorithm to determine the support and enabling for the XSAVE feature set:

1. Use CPUID to discover the value of CPUID.1:ECX.OSXSAVE.
   — If the bit is 0, either the XSAVE feature set is not supported by the processor or has not been enabled by software. Either way, the XSAVE feature set is not available, nor are XSAVE-enabled features such as AVX.
   — If the bit is 1, the processor supports the XSAVE feature set — including the XGETBV instruction — and it has been enabled by software. The XSAVE feature set can be used to manage x87 state (because XCR0[0] is always 1). Software requiring more detailed information can go on to the next step.
2. Execute XGETBV with ECX = 0 to discover the value of XCR0. If XCR0[1] = 1, the XSAVE feature set can be used to manage SSE state. If XCR0[2] = 1, the XSAVE feature set can be used to manage AVX state and software can execute AVX instructions. If XCR0[4:3] is 11b, the XSAVE feature set can be used to manage MPX state and software can execute MPX instructions. If XCR0[7:5] is 111b, the XSAVE feature set can be used to manage AVX-512 state and software can execute AVX-512 instructions. If XCR0[9] = 1, the XSAVE feature set can be used to manage PKRU state.

The IA32_XSS MSR (with MSR index DA0H) is zero coming out of RESET. If CR4.OSXSAVE = 1, CPUID.(EAX=0DH,ECX=1):EAX[3] = 1, and CPL = 0, executing the WRMSR instruction with ECX = DA0H writes the 64-bit value in EDX:EAX to the IA32_XSS MSR (EAX is written to IA32_XSS[31:0] and EDX to IA32_XSS[63:32]). The following items provide details regarding individual bits in the IA32_XSS MSR:

- IA32_XSS[8] is associated with PT state (see Section 13.5.6). Software can use XSAVES and XRSTORS to manage PT state only if IA32_XSS[8] = 1. The value of IA32_XSS[8] does not determine whether software can use Intel Processor Trace (the feature can be used even if IA32_XSS[8] = 0).
- IA32_XSS[63:9] and IA32_XSS[7:0] are reserved.[1] Executing the WRMSR instruction causes a general-protection fault (#GP) if ECX = DA0H and any corresponding bit in EDX:EAX is not 0. These bits in XCR0 are all 0 coming out of RESET.

The IA32_XSS MSR is 0 coming out of RESET.

There is no mechanism by which software operating with CPL > 0 can discover the value of the IA32_XSS MSR.

...

5. Updates to Chapter 15, Volume 1

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

15.3.8.1 Instruction Based Considerations

Programmers can use any instruction safely inside a transactional region. Further, programmers can use the Intel TSX instructions and prefixes at any privilege level. However, some instructions will always abort the transactional execution and cause execution to seamlessly and safely transition to a non-transactional path.

1. Bit 9 and bits 7:0 correspond to user state components. Since bits can be set in the IA32_XSS MSR only for supervisor state components, those bits of the MSR must be 0.

Intel TSX allows for most common instructions to be used inside transactional regions without causing aborts. The following operations inside a transactional region do not typically cause an abort.

- Operations on the instruction pointer register, general purpose registers (GPRs) and the status flags (CF, OF, SF, PF, AF, and ZF).
- Operations on XMM and YMM registers and the MXCSR register

However, programmers must be careful when intermixing SSE and AVX operations inside a transactional region. Intermixing SSE instructions accessing XMM registers and AVX instructions accessing YMM registers may cause transactional regions to abort.

CLD and STD instructions when used inside transactional regions may cause aborts if they change the value of the DF flag. However, if DF is 1, the STD instruction will not cause an abort. Similarly, if DF is 0, the CLD instruction will not cause an abort.

Instructions not enumerated here as causing abort when used inside a transactional region will typically not cause the execution to abort (examples include but are not limited to MFENCE, LFENCE, SFENCE, RDTSC, RDTSCP, etc.).

The following instructions will abort transactional execution on any implementation:

- XABORT
- CPUID
- PAUSE

In addition, in some implementations, the following instructions may always cause transactional aborts. These instructions are not expected to be commonly used inside typical transactional regions. However, programmers must not rely on these instructions to force a transactional abort, since whether they cause transactional aborts is implementation dependent.

- Operations on X87 and MMX architecture state. This includes all MMX and X87 instructions, including the FXRSTOR and FXSAVE instructions.
- Update to non-status portion of EFLAGS: CLI, STI, POPFD, POPFQ.
- Instructions that update segment registers, debug registers and/or control registers: MOV to DS/ES/FS/GS/SS, POP DS/ES/FS/GS/SS, LDS, LES, LFS, LGS, LSS, SWAPGS, WRFSBASE, WRGSBASE, LGDT, SGDT, LIDT, SIDT, LLDT, SLDT, LTR, STR, Far CALL, Far JMP, Far RET, IRET, MOV to DRx, MOV to CR0/CR2/CR3/CR4/CR8, CLTS and LMSW.
- Ring transitions: SYSENTER, SYSCALL, SYSEXIT, and SYSRET.
- TLB and Cacheability control: CLFLUSH, CLFLUSHOPT, INVD, WBINVD, INVLPG, INVPCID, and memory instructions with a non-temporal hint (V/MOVNTDQA, V/MOVNTDQ, V/MOVNTI, V/MOVNTPD, V/MOVNTPS, V/MOVNTQ, V/MASKMOVQ, and V/MASKMOVDQU).
- Processor state save: XSAVE, XSAVEOPT, and XRSTOR.
- Interrupts: INTn, INTO.
- IO: IN, INS, REP INS, OUT, OUTS, REP OUTS and their variants.
- VMX: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON, INVEPT, INVVPID, and VMFUNC.
- SMX: GETSEC.
- UD2, RSM, RDMSR, WRMSR, HLT, MONITOR, MWAIT, XSETBV, VZEROUPPER, MASKMOVQ, and V/MASKMOVDQU.

15.3.8.2 Runtime Considerations

In addition to the instruction-based considerations, runtime events may cause transactional execution to abort. These may be due to data access patterns or micro-architectural implementation causes. Keep in mind that the following list is not a comprehensive discussion of all abort causes.

Any fault or trap in a transactional region that must be exposed to software will be suppressed. Transactional execution will abort and execution will transition to a non-transactional execution, as if the fault or trap had never occurred. If any exception is not masked, that will result in a transactional abort and it will be as if the exception had never occurred.

When executed in VMX non-root operation, certain instructions may result in a VM exit. When such instructions are executed inside a transactional region, then instead of causing a VM exit, they will cause a transactional abort and the execution will appear as if instruction that would have caused a VM exit never executed.

Synchronous exception events (#DE, #OF, #NP, #SS, #GP, #BR, #UD, #AC, #XM, #PF, #NM, #TS, #MF, #DB, #BP/INT3) that occur during transactional execution may cause an execution not to commit transactionally, and require a non-transactional execution. These events are suppressed as if they had never occurred. With HLE, since the non-transactional code path is identical to the transactional code path, these events will typically re-appear when the instruction that caused the exception is re-executed non-transactionally, causing the associated synchronous events to be delivered appropriately in the non-transactional execution. The same behavior also applies to synchronous events (EPT violations, EPT misconfigurations, and accesses to the APIC-access page) that occur in VMX non-root operation.

Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution. The asynchronous events will be pended and handled after the transactional abort is processed. The same behavior also applies to asynchronous events (VMX-preemption timer expiry, virtual-interrupt delivery, and interrupt-window exiting) that occur in VMX non-root operation.

Transactional execution only supports write-back cacheable memory type operations. A transactional region may always abort if it includes operations on any other memory type. This includes instruction fetches to UC memory type.

Memory accesses within a transactional region may require the processor to set the Accessed and Dirty flags of the referenced page table entry. The behavior of how the processor handles this is implementation specific. Some implementations may allow the updates to these flags to become externally visible even if the transactional region subsequently aborts. Some Intel TSX implementations may choose to abort the transactional execution if these flags need to be updated. Further, a processor's page-table walk may generate accesses to its own transactionally written but uncommitted state. Some Intel TSX implementations may choose to abort the execution of a transactional region in such situations. Regardless, the architecture ensures that, if the transactional region aborts, then the transactionally written state will not be made architecturally visible through the behavior of structures such as TLBs.

Executing self-modifying code transactionally may also cause transactional aborts. Programmers must continue to follow the Intel recommended guidelines for writing self-modifying and cross-modifying code even when employing Intel TSX.

While an Intel TSX implementation will typically provide sufficient resources for executing common transactional regions, implementation constraints and excessive sizes for transactional regions may cause a transactional execution to abort and transition to a non-transactional execution. The architecture provides no guarantee of the amount of resources available to do transactional execution and does not guarantee that a transactional execution will ever succeed.

Conflicting requests to a cache line accessed within a transactional region may prevent the transactional region from executing successfully. For example, if logical processor P0 reads line A in a transactional region and another logical processor P1 writes A (either inside or outside a transactional region) then logical processor P0 may abort if logical processor P1’s write interferes with processor P0's ability to execute transactionally. Similarly, if P0 writes line A in a transactional region and P1reads or writes A (either inside or outside a transactional region), then P0 may abort if P1's access to A interferes with P0's ability to execute transactionally. In addition, other coherence traffic may at times appear as conflicting requests and may cause aborts. While these false conflicts may happen, they are expected to be uncommon. The conflict resolution policy to determine whether P0 or P1 aborts in the above scenarios is implementation specific.

...

6. 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:

- 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™ processor family
- 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® Atom™ processor 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™ processor family is 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 32nm 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 and 5th generation Intel® Core™ processors are based on the Intel® micro-architecture 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.

...

7. 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.

...

CLFLUSH—Flush Cache Line

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
0F AE /7	CLFLUSH *m8*	M	Valid	Valid	Flushes cache line containing *m8*.

Instruction Operand Encoding

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

Description

Invalidates from every level of the cache hierarchy in the cache coherence domain the cache line that contains the linear address specified with the memory operand. If that cache line contains modified data at any level of the cache hierarchy, that data is written back to memory. The source operand is a byte memory location.

The availability of CLFLUSH is indicated by the presence of the CPUID feature flag CLFSH (CPUID.01H:EDX[bit 19]). The aligned cache line size affected is also indicated with the CPUID instruction (bits 8 through 15 of the EBX register when the initial value in the EAX register is 1).

The memory attribute of the page containing the affected line has no effect on the behavior of this instruction. It should be noted that processors are free to speculatively fetch and cache data from system memory regions assigned a memory-type allowing for speculative reads (such as, the WB, WC, and WT memory types). PREFETCH*h* instructions can be used to provide the processor with hints for this speculative behavior. Because this speculative fetching can occur at any time and is not tied to instruction execution, the CLFLUSH instruction is not ordered with respect to PREFETCH*h* instructions or any of the speculative fetching mechanisms (that is, data can be speculatively loaded into a cache line just before, during, or after the execution of a CLFLUSH instruction that references the cache line).

Executions of the CLFLUSH instruction are ordered with respect to each other and with respect to writes, locked read-modify-write instructions, fence instructions, and executions of CLFLUSHOPT to the same cache line.[1] They are not ordered with respect to executions of CLFLUSHOPT to different cache lines.

The CLFLUSH instruction can be used at all privilege levels and is subject to all permission checking and faults associated with a byte load (and in addition, a CLFLUSH instruction is allowed to flush a linear address in an execute-only segment). Like a load, the CLFLUSH instruction sets the A bit but not the D bit in the page tables.

In some implementations, the CLFLUSH instruction may always cause transactional abort with Transactional Synchronization Extensions (TSX). The CLFLUSH instruction is not expected to be commonly used inside typical transactional regions. However, programmers must not rely on CLFLUSH instruction to force a transactional abort, since whether they cause transactional abort is implementation dependent.

The CLFLUSH instruction was introduced with the SSE2 extensions; however, because it has its own CPUID feature flag, it can be implemented in IA-32 processors that do not include the SSE2 extensions. Also, detecting the presence of the SSE2 extensions with the CPUID instruction does not guarantee that the CLFLUSH instruction is implemented in the processor.

CLFLUSH operation is the same in non-64-bit modes and 64-bit mode.

Operation

```
Flush_Cache_Line(SRC);
```

Intel C/C++ Compiler Intrinsic Equivalents

CLFLUSH: void _mm_clflush(void const *p)

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#SS(0)	For an illegal address in the SS segment.

1. Earlier versions of this manual specified that executions of the CLFLUSH instruction were ordered only by the MFENCE instruction. All processors implementing the CLFLUSH instruction also order it relative to the other operations enumerated above.

#PF(fault-code)	For a page fault.
#UD	If CPUID.01H:EDX.CLFSH[bit 19] = 0.
	If the LOCK prefix is used.
	If an instruction prefix F2H or F3H is used.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#UD	If CPUID.01H:EDX.CLFSH[bit 19] = 0.
	If the LOCK prefix is used.
	If an instruction prefix F2H or F3H is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	For a page fault.
#UD	If CPUID.01H:EDX.CLFSH[bit 19] = 0.
	If the LOCK prefix is used.
	If an instruction prefix F2H or F3H is used.

...

CLFLUSHOPT—Flush Cache Line Optimized

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F AE /7	CLFLUSHOPT *m8*	M	Valid	Valid	Flushes cache line containing *m8*.

Instruction Operand Encoding

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

Description

Invalidates from every level of the cache hierarchy in the cache coherence domain the cache line that contains the linear address specified with the memory operand. If that cache line contains modified data at any level of the cache hierarchy, that data is written back to memory. The source operand is a byte memory location.

The availability of CLFLUSHOPT is indicated by the presence of the CPUID feature flag CLFLUSHOPT (CPUID.(EAX=7,ECX=0):EBX[bit 23]). The aligned cache line size affected is also indicated with the CPUID instruction (bits 8 through 15 of the EBX register when the initial value in the EAX register is 1).

The memory attribute of the page containing the affected line has no effect on the behavior of this instruction. It should be noted that processors are free to speculatively fetch and cache data from system memory regions assigned a memory-type allowing for speculative reads (such as, the WB, WC, and WT memory types). PREFETCH*h* instructions can be used to provide the processor with hints for this speculative behavior. Because this speculative fetching can occur at any time and is not tied to instruction execution, the CLFLUSH instruction is not ordered with respect to PREFETCH*h* instructions or any of the speculative fetching mechanisms (that is, data can be speculatively loaded into a cache line just before, during, or after the execution of a CLFLUSH instruction that references the cache line).

Executions of the CLFLUSHOPT instruction are ordered with respect to fence instructions and to locked read-modify-write instructions; they are also ordered with respect to the following accesses to the cache line being invalidated: writes, executions of CLFLUSH, and executions of CLFLUSHOPT. They are not ordered with respect to writes, executions of CLFLUSH, or executions of CLFLUSHOPT that access other cache lines; to enforce ordering with such an operation, software can insert an SFENCE instruction between CFLUSHOPT and that operation.

The CLFLUSHOPT instruction can be used at all privilege levels and is subject to all permission checking and faults associated with a byte load (and in addition, a CLFLUSHOPT instruction is allowed to flush a linear address in an execute-only segment). Like a load, the CLFLUSHOPT instruction sets the A bit but not the D bit in the page tables.

In some implementations, the CLFLUSHOPT instruction may always cause transactional abort with Transactional Synchronization Extensions (TSX). The CLFLUSHOPT instruction is not expected to be commonly used inside typical transactional regions. However, programmers must not rely on CLFLUSHOPT instruction to force a transactional abort, since whether they cause transactional abort is implementation dependent.

CLFLUSHOPT operation is the same in non-64-bit modes and 64-bit mode.

Operation

```
Flush_Cache_Line_Optimized(SRC);
```

Intel C/C++ Compiler Intrinsic Equivalents

CLFLUSHOPT:void _mm_clflushopt(void const *p)

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#UD	If CPUID.(EAX=7,ECX=0):EBX.CLFLUSHOPT[bit 23] = 0.
	If the LOCK prefix is used.
	If an instruction prefix F2H or F3H is used.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#UD	If CPUID.(EAX=7,ECX=0):EBX.CLFLUSHOPT[bit 23] = 0.
	If the LOCK prefix is used.
	If an instruction prefix F2H or F3H is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	For a page fault.
#UD	If CPUID.(EAX=7,ECX=0):EBX.CLFLUSHOPT[bit 23] = 0.
	If the LOCK prefix is used.
	If an instruction prefix F2H or F3H is used.

...

CMPSS—Compare Scalar Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
F3 0F C2 /r *ib* CMPSS *xmm1, xmm2/m32, imm8*	RMI	V/V	SSE	Compare low single-precision floating-point value in *xmm2/m32* and *xmm1* using *imm8* as comparison predicate.
VEX.NDS.LIG.F3.0F.WIG C2 /r ib VCMPSS xmm1, xmm2, xmm3/m32, imm8	RVMI	V/V	AVX	Compare low single precision floating-point value in xmm3/m32 and xmm2 using bits 4:0 of imm8 as comparison predicate.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RMI	ModRM:reg (r, w)	ModRM:r/m (r)	imm8	NA
RVMI	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	imm8

Description

Compares the low single-precision floating-point values in the source operand (second operand) and the destination operand (first operand) and returns the results of the comparison to the destination operand. The comparison predicate operand (third operand) specifies the type of comparison performed. The comparison result is a doubleword mask of all 1s (comparison true) or all 0s (comparison false). The sign of zero is ignored for comparisons, so that −0.0 is equal to +0.0.

128-bit Legacy SSE version: The first source and destination operand (first operand) is an XMM register. The second source operand (second operand) can be an XMM register or 32-bit memory location. The comparison predicate operand is an 8-bit immediate, bits 2:0 of the immediate define the type of comparison to be performed (see Table 3-7). Bits 7:3 of the immediate is reserved. Bits (VLMAX-1:32) of the corresponding YMM destination register remain unchanged.

The unordered relationship is true when at least one of the two source operands being compared is a NaN; the ordered relationship is true when neither source operand is a NaN

A subsequent computational instruction that uses the mask result in the destination operand as an input operand will not generate a fault, since a mask of all 0s corresponds to a floating-point value of +0.0 and a mask of all 1s corresponds to a QNaN.

Note that processors with "CPUID.1H:ECX.AVX =0" do not implement the "greater-than", "greater-than-or-equal", "not-greater than", and "not-greater-than-or-equal relations" predicates. These comparisons can be made either by using the inverse relationship (that is, use the "not-less-than-or-equal" to make a "greater-than" comparison) or by using software emulation. When using software emulation, the program must swap the operands (copying registers when necessary to protect the data that will now be in the destination operand), and then perform the compare using a different predicate. The predicate to be used for these emulations is listed in Table 3-7 under the heading Emulation.

Compilers and assemblers may implement the following two-operand pseudo-ops in addition to the three-operand CMPSS instruction, for processors with "CPUID.1H:ECX.AVX =0". See Table 3-15. Compiler should treat reserved Imm8 values as illegal syntax.

...

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 3-17 shows information returned, depending on the initial value loaded into the EAX register. Table 3-18 shows the maximum CPUID input value recognized for each family of IA-32 processors on which CPUID is implemented.

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. *)
```

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

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. For example, using the Intel Core i7 processor, the following is true:

CPUID.EAX = 07H (*Returns EAX=EBX=ECX=EDX=0. *)

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 3-17. Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX EBX ECX EDX	Maximum Input Value for Basic CPUID Information (see Table 3-18) "Genu" "ntel" "inel"
01H	EAX EBX ECX EDX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-6) 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 Feature Information (see Figure 3-7 and Table 3-19) Feature Information (see Figure 3-8 and Table 3-20) **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 EBX ECX EDX	Cache and TLB Information (see Table 3-21) Cache and TLB Information Cache and TLB Information Cache and TLB Information
03H	EAX EBX ECX EDX	Reserved. Reserved. Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.) Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature. CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLE.BOOT_NT4[bit 22] = 0 (default).
	Deterministic Cache Parameters Leaf
04H	**NOTES:** Leaf 04H output depends on the initial value in ECX.* See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level on page 3-200. 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 0: 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 1: Cache Inclusiveness 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 2: 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

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		NOTES: * If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EAX[4:0] as 0. ** Add one to the return value to get the result. ***The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache **** The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. ***** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.
	MONITOR/MWAIT Leaf	
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	

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
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
	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.

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EBX	Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: Reserved Bit 03: BMI1 Bit 04: HLE Bit 05: AVX2 Bit 06: Reserved 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 Platform Quality of Service Monitoring (PQM) capability if 1. Bit 13: Deprecates FPU CS and FPU DS values if 1. Bit 14: MPX. Supports Intel® Memory Protection Extensions if 1. Bit 15: Supports Platform Quality of Service Enforcement (PQE) capability if 1. 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 31:26: Reserved
	ECX	Bit 00: PREFETCHWT1 Bits 02:01: Reserved 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 31:05: Reserved
	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

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	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.
	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

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
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 Bit 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 XCR0 | IA32_XSS.
	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)		

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
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 0 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 1 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.
	Platform QoS 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 31-0: Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.
	EDX	Bit 00: Reserved. Bit 01: Supports L3 Cache QoS Monitoring if 1. Bits 31:02: Reserved
	L3 Cache QoS Monitoring Capability Enumeration Sub-leaf (EAX = 0FH, ECX = 1)	
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX.
	EAX	Reserved.
	EBX	Bits 31-0: 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. Bits 31:01: Reserved
	Platform QoS Enforcement Enumeration Sub-leaf (EAX = 10H, ECX = 0)	

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
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 QoS Enforcement if 1. Bits 31:02: Reserved
	ECX	Reserved.
	EDX	Reserved.
L3 Cache QoS Enforcement Enumeration Sub-leaf (EAX = 10H, ECX = ResID =1)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 4:0: Length of the capacity bit mask for the corresponding ResID. Bits 31:05: Reserved
	EBX	Bits 31-0: 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 15:0: Highest COS number supported for this ResID. Bits 31:16: Reserved
Intel Processor Trace Enumeration Main Leaf (EAX = 14H, ECX = 0)		
14H		**NOTES:** Leaf 14H main leaf (ECX = 0).
	EAX	Bits 31:0: Reports the maximum number sub-leaves that are 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. Bits 01: If 1, Indicates support of Configurable PSB and Cycle-Accurate Mode. Bits 02: If 1, Indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset. Bits 03: If 1, Indicates support of MTC timing packet and suppression of COFI-based packets. Bits 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. Bits 02: If 1, Indicates support of Single-Range Output scheme. Bits 03: If 1, Indicates support of output to Trace Transport subsystem. Bit 30:04: Reserved Bit 31: If 1, Generated packets which contain IP payloads have LIP values, which include the CS base component.
	EDX	Bits 31- 00: Reserved

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	Intel Processor Trace Enumeration Sub-leaf (EAX = 14H, ECX = 1)	
14H	EAX	Bits 2:0: Number of configurable Address Ranges for filtering. Bits 15-03: Reserved Bit 31:16: Bitmap of supported MTC period encodings
	EBX	Bits 15-0: Bitmap of supported Cycle Threshold value encodings Bit 31:16: Bitmap of supported Configurable PSB frequency encodings
	ECX	Bits 31-00: Reserved
	EDX	Bits 31- 00: Reserved
	Time Stamp Counter/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 31:0: An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31-0: An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.
	ECX	Bits 31:0: Reserved = 0.
	EDX	Bits 31:0: Reserved = 0.
	Processor Frequency Information Leaf	
16H	EAX	Bits 15:0: Processor Base Frequency (in MHz). Bits 31:16: Reserved =0
	EBX	Bits 15:0: Maximum Frequency (in MHz). Bits 31:16: Reserved = 0
	ECX	Bits 15:0: 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.
	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	

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 3-18).
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode Bits 04-01 Reserved Bit 05: LZCNT Bits 07-06 Reserved Bit 08: PREFETCHW Bits 31-09 Reserved
	EDX	Bits 10-00: Reserved Bit 11: SYSCALL/SYSRET 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 Bits 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

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	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 **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 15-8: #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 = 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.

...

IRET/IRETD—Interrupt Return

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
CF	IRET	NP	Valid	Valid	Interrupt return (16-bit operand size).
CF	IRETD	NP	Valid	Valid	Interrupt return (32-bit operand size).
REX.W + CF	IRETQ	NP	Valid	N.E.	Interrupt return (64-bit operand size).

Instruction Operand Encoding

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

Description

Returns program control from an exception or interrupt handler to a program or procedure that was interrupted by an exception, an external interrupt, or a software-generated interrupt. These instructions are also used to perform a return from a nested task. (A nested task is created when a CALL instruction is used to initiate a task switch or when an interrupt or exception causes a task switch to an interrupt or exception handler.) See the section titled "Task Linking" in Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

IRET and IRETD are mnemonics for the same opcode. The IRETD mnemonic (interrupt return double) is intended for use when returning from an interrupt when using the 32-bit operand size; however, most assemblers use the IRET mnemonic interchangeably for both operand sizes.

In Real-Address Mode, the IRET instruction preforms a far return to the interrupted program or procedure. During this operation, the processor pops the return instruction pointer, return code segment selector, and EFLAGS image from the stack to the EIP, CS, and EFLAGS registers, respectively, and then resumes execution of the interrupted program or procedure.

In Protected Mode, the action of the IRET instruction depends on the settings of the NT (nested task) and VM flags in the EFLAGS register and the VM flag in the EFLAGS image stored on the current stack. Depending on the setting of these flags, the processor performs the following types of interrupt returns:

- Return from virtual-8086 mode.
- Return to virtual-8086 mode.
- Intra-privilege level return.
- Inter-privilege level return.
- Return from nested task (task switch).

If the NT flag (EFLAGS register) is cleared, the IRET instruction performs a far return from the interrupt procedure, without a task switch. The code segment being returned to must be equally or less privileged than the interrupt handler routine (as indicated by the RPL field of the code segment selector popped from the stack).

As with a real-address mode interrupt return, the IRET instruction pops the return instruction pointer, return code segment selector, and EFLAGS image from the stack to the EIP, CS, and EFLAGS registers, respectively, and then resumes execution of the interrupted program or procedure. If the return is to another privilege level, the IRET instruction also pops the stack pointer and SS from the stack, before resuming program execution. If the return is to virtual-8086 mode, the processor also pops the data segment registers from the stack.

If the NT flag is set, the IRET instruction performs a task switch (return) from a nested task (a task called with a CALL instruction, an interrupt, or an exception) back to the calling or interrupted task. The updated state of the task executing the IRET instruction is saved in its TSS. If the task is re-entered later, the code that follows the IRET instruction is executed.

If the NT flag is set and the processor is in IA-32e mode, the IRET instruction causes a general protection exception.

If nonmaskable interrupts (NMIs) are blocked (see Section 6.7.1, "Handling Multiple NMIs" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*), execution of the IRET instruction unblocks NMIs. This unblocking occurs even if the instruction causes a fault. In such a case, NMIs are unmasked before the exception handler is invoked.

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

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 PE = 0
    THEN GOTO REAL-ADDRESS-MODE;
ELSIF (IA32_EFER.LMA = 0)
    THEN
        IF (EFLAGS.VM = 1)
            THEN GOTO RETURN-FROM-VIRTUAL-8086-MODE;
            ELSE GOTO PROTECTED-MODE;
        FI;
    ELSE GOTO IA-32e-MODE;
FI;

REAL-ADDRESS-MODE;
    IF OperandSize = 32
        THEN
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            tempEFLAGS ← Pop();
            EFLAGS ← (tempEFLAGS AND 257FD5H) OR (EFLAGS AND 1A0000H);
        ELSE (* OperandSize = 16 *)
            EIP ← Pop(); (* 16-bit pop; clear upper 16 bits *)
            CS ← Pop(); (* 16-bit pop *)
            EFLAGS[15:0] ← Pop();
    FI;
    END;

RETURN-FROM-VIRTUAL-8086-MODE:
(* Processor is in virtual-8086 mode when IRET is executed and stays in virtual-8086 mode *)
    IF IOPL = 3 (* Virtual mode: PE = 1, VM = 1, IOPL = 3 *)
        THEN IF OperandSize = 32
            THEN
                EIP ← Pop();
                CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
                EFLAGS ← Pop();
                (* VM, IOPL,VIP and VIF EFLAG bits not modified by pop *)
                IF EIP not within CS limit
                    THEN #GP(0); FI;
```

```
            ELSE (* OperandSize = 16 *)
                EIP ← Pop(); (* 16-bit pop; clear upper 16 bits *)
                CS ← Pop(); (* 16-bit pop *)
                EFLAGS[15:0] ← Pop(); (* IOPL in EFLAGS not modified by pop *)
                IF EIP not within CS limit
                    THEN #GP(0); FI;
            FI;
        ELSE
            #GP(0); (* Trap to virtual-8086 monitor: PE = 1, VM = 1, IOPL < 3 *)
    FI;
END;

PROTECTED-MODE:
    IF NT = 1
        THEN GOTO TASK-RETURN; (* PE = 1, VM = 0, NT = 1 *)
    FI;
    IF OperandSize = 32
        THEN
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            tempEFLAGS ← Pop();
        ELSE (* OperandSize = 16 *)
            EIP ← Pop(); (* 16-bit pop; clear upper bits *)
            CS ← Pop(); (* 16-bit pop *)
            tempEFLAGS ← Pop(); (* 16-bit pop; clear upper bits *)
    FI;
    IF tempEFLAGS(VM) = 1 and CPL = 0
        THEN GOTO RETURN-TO-VIRTUAL-8086-MODE;
        ELSE GOTO PROTECTED-MODE-RETURN;
    FI;

TASK-RETURN: (* PE = 1, VM = 0, NT = 1 *)
    SWITCH-TASKS (without nesting) to TSS specified in link field of current TSS;
    Mark the task just abandoned as NOT BUSY;
    IF EIP is not within CS limit
        THEN #GP(0); FI;
END;

RETURN-TO-VIRTUAL-8086-MODE:
    (* Interrupted procedure was in virtual-8086 mode: PE = 1, CPL=0, VM = 1 in flag image *)
    IF EIP not within CS limit
        THEN #GP(0); FI;
    EFLAGS ← tempEFLAGS;
    ESP ← Pop();
    SS ← Pop(); (* Pop 2 words; throw away high-order word *)
    ES ← Pop(); (* Pop 2 words; throw away high-order word *)
    DS ← Pop(); (* Pop 2 words; throw away high-order word *)
    FS ← Pop(); (* Pop 2 words; throw away high-order word *)
    GS ← Pop(); (* Pop 2 words; throw away high-order word *)
    CPL ← 3;
```

```
    (* Resume execution in Virtual-8086 mode *)
END;

PROTECTED-MODE-RETURN: (* PE = 1 *)
    IF CS(RPL) > CPL
        THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
        ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
END;

RETURN-TO-OUTER-PRIVILEGE-LEVEL:
    IF OperandSize = 32
        THEN
            ESP ← Pop();
            SS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
    ELSE IF OperandSize = 16
        THEN
            ESP ← Pop(); (* 16-bit pop; clear upper bits *)
            SS ← Pop(); (* 16-bit pop *)
        ELSE (* OperandSize = 64 *)
            RSP ← Pop();
            SS ← Pop(); (* 64-bit pop, high-order 48 bits discarded *)
    FI;
    IF new mode ≠ 64-Bit Mode
        THEN
            IF EIP is not within CS limit
                THEN #GP(0); FI;
        ELSE (* new mode = 64-bit mode *)
            IF RIP is non-canonical
                    THEN #GP(0); FI;
    FI;
    EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
    IF OperandSize = 32
        THEN EFLAGS(RF, AC, ID) ← tempEFLAGS; FI;
    IF CPL ≤ IOPL
        THEN EFLAGS(IF) ← tempEFLAGS; FI;
    IF CPL = 0
        THEN
            EFLAGS(IOPL) ← tempEFLAGS;
            IF OperandSize = 32
                THEN EFLAGS(VM, VIF, VIP) ← tempEFLAGS; FI;
            IF OperandSize = 64
                THEN EFLAGS(VIF, VIP) ← tempEFLAGS; FI;
    FI;
    CPL ← CS(RPL);
    FOR each SegReg in (ES, FS, GS, and DS)
        DO
            tempDesc ← descriptor cache for SegReg (* hidden part of segment register *)
            IF tempDesc(DPL) < CPL AND tempDesc(Type) is data or non-conforming code
                THEN (* Segment register invalid *)
                    SegReg ← NULL;
```

```
                FI;
            OD;
END;

RETURN-TO-SAME-PRIVILEGE-LEVEL: (* PE = 1, RPL = CPL *)
    IF new mode ≠ 64-Bit Mode
        THEN
            IF EIP is not within CS limit
                THEN #GP(0); FI;
        ELSE (* new mode = 64-bit mode *)
            IF RIP is non-canonical
                    THEN #GP(0); FI;
    FI;
    EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
    IF OperandSize = 32 or OperandSize = 64
        THEN EFLAGS(RF, AC, ID) ← tempEFLAGS; FI;
    IF CPL ≤ IOPL
        THEN EFLAGS(IF) ← tempEFLAGS; FI;
    IF CPL = 0
        THEN (* VM = 0 in flags image *)
            EFLAGS(IOPL) ← tempEFLAGS;
            IF OperandSize = 32 or OperandSize = 64
                THEN EFLAGS(VIF, VIP) ← tempEFLAGS; FI;
    FI;
END;

IA-32e-MODE:
    IF NT = 1
        THEN #GP(0);
    ELSE IF OperandSize = 32
        THEN
            EIP ← Pop();
            CS ← Pop();
            tempEFLAGS ← Pop();
        ELSE IF OperandSize = 16
            THEN
                EIP ← Pop(); (* 16-bit pop; clear upper bits *)
                CS ← Pop(); (* 16-bit pop *)
                tempEFLAGS ← Pop(); (* 16-bit pop; clear upper bits *)
            FI;
        ELSE (* OperandSize = 64 *)
            THEN
                    RIP ← Pop();
                    CS ← Pop(); (* 64-bit pop, high-order 48 bits discarded *)
                    tempRFLAGS ← Pop();
    FI;
    IF tempCS.RPL > CPL
        THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
        ELSE
            IF instruction began in 64-Bit Mode
```

```
                THEN
                    IF OperandSize = 32
                        THEN
                            ESP ← Pop();
                            SS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
                    ELSE IF OperandSize = 16
                        THEN
                            ESP ← Pop(); (* 16-bit pop; clear upper bits *)
                            SS ← Pop(); (* 16-bit pop *)
                        ELSE (* OperandSize = 64 *)
                            RSP ← Pop();
                            SS ← Pop(); (* 64-bit pop, high-order 48 bits discarded *)
                    FI;
            FI;
            GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
END;
```

Flags Affected

All the flags and fields in the EFLAGS register are potentially modified, depending on the mode of operation of the processor. If performing a return from a nested task to a previous task, the EFLAGS register will be modified according to the EFLAGS image stored in the previous task's TSS.

Protected Mode Exceptions

#GP(0)	If the return code or stack segment selector is NULL.
	If the return instruction pointer is not within the return code segment limit.
#GP(selector)	If a segment selector index is outside its descriptor table limits.
	If the return code segment selector RPL is less than the CPL.
	If the DPL of a conforming-code segment is greater than the return code segment selector RPL.
	If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector.
	If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector.
	If the stack segment is not a writable data segment.
	If the stack segment selector RPL is not equal to the RPL of the return code segment selector.
	If the segment descriptor for a code segment does not indicate it is a code segment.
	If the segment selector for a TSS has its local/global bit set for local.
	If a TSS segment descriptor specifies that the TSS is not busy.
	If a TSS segment descriptor specifies that the TSS is not available.
#SS(0)	If the top bytes of stack are not within stack limits.
#NP(selector)	If the return code or stack segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If the return instruction pointer is not within the return code segment limit.
#SS	If the top bytes of stack are not within stack limits.

Virtual-8086 Mode Exceptions

#GP(0)	If the return instruction pointer is not within the return code segment limit.
	IF IOPL not equal to 3.
#PF(fault-code)	If a page fault occurs.
#SS(0)	If the top bytes of stack are not within stack limits.
#AC(0)	If an unaligned memory reference occurs and alignment checking is enabled.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

#GP(0)	If EFLAGS.NT[bit 14] = 1.

Other exceptions same as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If EFLAGS.NT[bit 14] = 1.
	If the return code segment selector is NULL.
	If the stack segment selector is NULL going back to compatibility mode.
	If the stack segment selector is NULL going back to CPL3 64-bit mode.
	If a NULL stack segment selector RPL is not equal to CPL going back to non-CPL3 64-bit mode.
	If the return instruction pointer is not within the return code segment limit.
	If the return instruction pointer is non-canonical.
#GP(Selector)	If a segment selector index is outside its descriptor table limits.
	If a segment descriptor memory address is non-canonical.
	If the segment descriptor for a code segment does not indicate it is a code segment.
	If the proposed new code segment descriptor has both the D-bit and L-bit set.
	If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector.
	If CPL is greater than the RPL of the code segment selector.
	If the DPL of a conforming-code segment is greater than the return code segment selector RPL.
	If the stack segment is not a writable data segment.
	If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector.
	If the stack segment selector RPL is not equal to the RPL of the return code segment selector.
#SS(0)	If an attempt to pop a value off the stack violates the SS limit.
	If an attempt to pop a value off the stack causes a non-canonical address to be referenced.
#NP(selector)	If the return code or stack segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.

#UD If the LOCK prefix is used.

...

Table 3-21. Encoding of CPUID Leaf 2 Descriptors

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

Table 3-21. Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
4BH	Cache	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size
4CH	Cache	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries
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: 1 GByte pages, 4-way set associative, 4 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, 126 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

Table 3-21. Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size
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.
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

Table 3-21. Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
F1H	Prefetch	128-Byte prefetching
FFH	General	CPUID leaf 2 does not report cache descriptor information, use CPUID leaf 4 to query cache parameters

…

8. 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.

…

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.
  - — Pentium III processor—1st- or 2nd-level cache.
  - — Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
  - — Pentium III processor—2nd-level cache.
  - — Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
  - — Pentium III processor—2nd-level cache.
  - — Pentium 4 and Intel Xeon processors—2nd-level cache.

- 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.
  - Pentium III processor—1st-level cache
  - Pentium 4 and Intel Xeon processors—2nd-level cache

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.

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 all 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 any other cached copy 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.

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.

...

PREFETCHWT1—Prefetch Vector Data Into Caches with Intent to Write and T1 Hint

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 0D /2 PREFETCHWT1 m8	M	V/V	PREFETCHWT1	Move data from m8 closer to the processor using T1 hint with intent to 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 line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by an intent to write hint (so that data is brought into 'Exclusive' state via a request for ownership) and a locality hint:

• T1 (temporal data with respect to first level cache)—prefetch data into the second level cache.

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. Use of any ModR/M value other than the specified ones will lead to unpredictable behavior.)

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 PREFETCHh 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.

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 PREFETCHh 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 PREFETCHh instruction is not ordered with respect to the fence instructions (MFENCE, SFENCE, and LFENCE) or locked memory references. A PREFETCHh instruction is also unordered with respect to CLFLUSH and CLFLUSHOPT instructions, other PREFETCHh 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

PREFETCH(mem, Level, State) Prefetches a byte memory location pointed by 'mem' into the cache level specified by 'Level'; a request for exclusive/ownership is done if 'State' is 1. Note that the memory location ignore cache line splits. This operation is considered a hint for the processor and may be skipped depending on implementation.

```
Prefetch (m8, Level = 1, EXCLUSIVE=1);
```

Flags Affected

All flags are affected

C/C++ Compiler Intrinsic Equivalent

```
void _mm_prefetch( char const *, int hint= _MM_HINT_ET1);
```

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.

...

RET—Return from Procedure

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
C3	RET	NP	Valid	Valid	Near return to calling procedure.
CB	RET	NP	Valid	Valid	Far return to calling procedure.
C2 *iw*	RET *imm16*	I	Valid	Valid	Near return to calling procedure and pop *imm16* bytes from stack.
CA *iw*	RET *imm16*	I	Valid	Valid	Far return to calling procedure and pop *imm16* bytes from stack.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA
I	imm16	NA	NA	NA

Description

Transfers program control to a return address located on the top of the stack. The address is usually placed on the stack by a CALL instruction, and the return is made to the instruction that follows the CALL instruction.

The optional source operand specifies the number of stack bytes to be released after the return address is popped; the default is none. This operand can be used to release parameters from the stack that were passed to the called procedure and are no longer needed. It must be used when the CALL instruction used to switch to a new procedure uses a call gate with a non-zero word count to access the new procedure. Here, the source operand for the RET instruction must specify the same number of bytes as is specified in the word count field of the call gate.

The RET instruction can be used to execute three different types of returns:

- **Near return** — A return to a calling procedure within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment return.
- **Far return** — A return to a calling procedure located in a different segment than the current code segment, sometimes referred to as an intersegment return.
- **Inter-privilege-level far return** — A far return to a different privilege level than that of the currently executing program or procedure.

The inter-privilege-level return type can only be executed in protected mode. See the section titled "Calling Procedures Using Call and RET" in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for detailed information on near, far, and inter-privilege-level returns.

When executing a near return, the processor pops the return instruction pointer (offset) from the top of the stack into the EIP register and begins program execution at the new instruction pointer. The CS register is unchanged.

When executing a far return, the processor pops the return instruction pointer from the top of the stack into the EIP register, then pops the segment selector from the top of the stack into the CS register. The processor then begins program execution in the new code segment at the new instruction pointer.

The mechanics of an inter-privilege-level far return are similar to an intersegment return, except that the processor examines the privilege levels and access rights of the code and stack segments being returned to determine if the control transfer is allowed to be made. The DS, ES, FS, and GS segment registers are cleared by the RET instruction during an inter-privilege-level return if they refer to segments that are not allowed to be accessed at the new privilege level. Since a stack switch also occurs on an inter-privilege level return, the ESP and SS registers are loaded from the stack.

If parameters are passed to the called procedure during an inter-privilege level call, the optional source operand must be used with the RET instruction to release the parameters on the return. Here, the parameters are released both from the called procedure's stack and the calling procedure's stack (that is, the stack being returned to).

In 64-bit mode, the default operation size of this instruction is the stack-address size, i.e. 64 bits. This applies to near returns, not far returns; the default operation size of far returns is 32 bits.

Operation

```
(* Near return *)
IF instruction = near return
    THEN;
        IF OperandSize = 32
            THEN
                IF top 4 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                EIP ← Pop();
            ELSE
                IF OperandSize = 64
                    THEN
                        IF top 8 bytes of stack not within stack limits
                            THEN #SS(0); FI;
                        RIP ← Pop();
                    ELSE (* OperandSize = 16 *)
                        IF top 2 bytes of stack not within stack limits
                            THEN #SS(0); FI;
                        tempEIP ← Pop();
                        tempEIP ← tempEIP AND 0000FFFFH;
                        IF tempEIP not within code segment limits
                            THEN #GP(0); FI;
                        EIP ← tempEIP;
                FI;
        FI;

    IF instruction has immediate operand
        THEN (* Release parameters from stack *)
            IF StackAddressSize = 32
                THEN
                    ESP ← ESP + SRC;
                ELSE
                    IF StackAddressSize = 64
                        THEN
                            RSP ← RSP + SRC;
                        ELSE (* StackAddressSize = 16 *)
                            SP ← SP + SRC;
                    FI;
            FI;
    FI;
FI;

(* Real-address mode or virtual-8086 mode *)
IF ((PE = 0) or (PE = 1 AND VM = 1)) and instruction = far return
```

```
    THEN
        IF OperandSize = 32
            THEN
                IF top 8 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                EIP ← Pop();
                CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            ELSE (* OperandSize = 16 *)
                IF top 4 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                tempEIP ← Pop();
                tempEIP ← tempEIP AND 0000FFFFH;
                IF tempEIP not within code segment limits
                    THEN #GP(0); FI;
                EIP ← tempEIP;
                CS ← Pop(); (* 16-bit pop *)
        FI;
    IF instruction has immediate operand
        THEN (* Release parameters from stack *)
            SP ← SP + (SRC AND FFFFH);
    FI;
FI;

(* Protected mode, not virtual-8086 mode *)
IF (PE = 1 and VM = 0 and IA32_EFER.LMA = 0) and instruction = far return
    THEN
        IF OperandSize = 32
            THEN
                IF second doubleword on stack is not within stack limits
                    THEN #SS(0); FI;
            ELSE (* OperandSize = 16 *)
                IF second word on stack is not within stack limits
                    THEN #SS(0); FI;
        FI;
    IF return code segment selector is NULL
        THEN #GP(0); FI;
    IF return code segment selector addresses descriptor beyond descriptor table limit
        THEN #GP(selector); FI;
    Obtain descriptor to which return code segment selector points from descriptor table;
    IF return code segment descriptor is not a code segment
        THEN #GP(selector); FI;
    IF return code segment selector RPL < CPL
        THEN #GP(selector); FI;
    IF return code segment descriptor is conforming
    and return code segment DPL > return code segment selector RPL
        THEN #GP(selector); FI;
    IF return code segment descriptor is non-conforming and return code
    segment DPL ≠ return code segment selector RPL
        THEN #GP(selector); FI;
    IF return code segment descriptor is not present
```

```
            THEN #NP(selector); FI:
        IF return code segment selector RPL > CPL
            THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
            ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL;
        FI;
    FI;

    RETURN-SAME-PRIVILEGE-LEVEL:
        IF the return instruction pointer is not within the return code segment limit
            THEN #GP(0); FI;
        IF OperandSize = 32
            THEN
                EIP ← Pop();
                CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            ELSE (* OperandSize = 16 *)
                EIP ← Pop();
                EIP ← EIP AND 0000FFFFH;
                CS ← Pop(); (* 16-bit pop *)
        FI;
        IF instruction has immediate operand
            THEN (* Release parameters from stack *)
                IF StackAddressSize = 32
                    THEN
                        ESP ← ESP + SRC;
                    ELSE (* StackAddressSize = 16 *)
                        SP ← SP + SRC;
                FI;
        FI;

    RETURN-TO-OUTER-PRIVILEGE-LEVEL:
        IF top (16 + SRC) bytes of stack are not within stack limits (OperandSize = 32)
        or top (8 + SRC) bytes of stack are not within stack limits (OperandSize = 16)
                THEN #SS(0); FI;
        Read return segment selector;
        IF stack segment selector is NULL
            THEN #GP(0); FI;
        IF return stack segment selector index is not within its descriptor table limits
            THEN #GP(selector); FI;
        Read segment descriptor pointed to by return segment selector;
        IF stack segment selector RPL ≠ RPL of the return code segment selector
        or stack segment is not a writable data segment
        or stack segment descriptor DPL ≠ RPL of the return code segment selector
                THEN #GP(selector); FI;
        IF stack segment not present
            THEN #SS(StackSegmentSelector); FI;
        IF the return instruction pointer is not within the return code segment limit
            THEN #GP(0); FI;
        CPL ← ReturnCodeSegmentSelector(RPL);
        IF OperandSize = 32
            THEN
```

```
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded; segment descriptor loaded *)
            CS(RPL) ← CPL;
            IF instruction has immediate operand
                THEN (* Release parameters from called procedure's stack *)
                    IF StackAddressSize = 32
                        THEN
                            ESP ← ESP + SRC;
                        ELSE (* StackAddressSize = 16 *)
                            SP ← SP + SRC;
                    FI;
            FI;
            tempESP ← Pop();
            tempSS ← Pop(); (* 32-bit pop, high-order 16 bits discarded; seg. descriptor loaded *)
            ESP ← tempESP;
            SS ← tempSS;
        ELSE (* OperandSize = 16 *)
            EIP ← Pop();
            EIP ← EIP AND 0000FFFFH;
            CS ← Pop(); (* 16-bit pop; segment descriptor loaded *)
            CS(RPL) ← CPL;
            IF instruction has immediate operand
                THEN (* Release parameters from called procedure's stack *)
                    IF StackAddressSize = 32
                        THEN
                            ESP ← ESP + SRC;
                        ELSE (* StackAddressSize = 16 *)
                            SP ← SP + SRC;
                    FI;
            FI;
            tempESP ← Pop();
            tempSS ← Pop(); (* 16-bit pop; segment descriptor loaded *)
            ESP ← tempESP;
            SS ← tempSS;
FI;

FOR each of segment register (ES, FS, GS, and DS)
    DO
        IF segment register points to data or non-conforming code segment
        and CPL > segment descriptor DPL (* DPL in hidden part of segment register *)
            THEN SegmentSelector ← 0; (* Segment selector invalid *)
        FI;
    OD;

IF instruction has immediate operand
    THEN (* Release parameters from calling procedure's stack *)
        IF StackAddressSize = 32
            THEN
                ESP ← ESP + SRC;
            ELSE (* StackAddressSize = 16 *)
```

```
                    SP ← SP + SRC;
                FI;
        FI;

(* IA-32e Mode *)
        IF (PE = 1 and VM = 0 and IA32_EFER.LMA = 1) and instruction = far return
            THEN
                IF OperandSize = 32
                    THEN
                        IF second doubleword on stack is not within stack limits
                            THEN #SS(0); FI;
                        IF first or second doubleword on stack is not in canonical space
                            THEN #SS(0); FI;
                    ELSE
                        IF OperandSize = 16
                            THEN
                                IF second word on stack is not within stack limits
                                    THEN #SS(0); FI;
                                IF first or second word on stack is not in canonical space
                                    THEN #SS(0); FI;
                            ELSE (* OperandSize = 64 *)
                                IF first or second quadword on stack is not in canonical space
                                    THEN #SS(0); FI;
                        FI
                FI;
            IF return code segment selector is NULL
                THEN GP(0); FI;
            IF return code segment selector addresses descriptor beyond descriptor table limit
                THEN GP(selector); FI;
            IF return code segment selector addresses descriptor in non-canonical space
                THEN GP(selector); FI;
            Obtain descriptor to which return code segment selector points from descriptor table;
            IF return code segment descriptor is not a code segment
                THEN #GP(selector); FI;
            IF return code segment descriptor has L-bit = 1 and D-bit = 1
                THEN #GP(selector); FI;
            IF return code segment selector RPL < CPL
                THEN #GP(selector); FI;
            IF return code segment descriptor is conforming
            and return code segment DPL > return code segment selector RPL
                THEN #GP(selector); FI;
            IF return code segment descriptor is non-conforming
            and return code segment DPL ≠ return code segment selector RPL
                THEN #GP(selector); FI;
            IF return code segment descriptor is not present
                THEN #NP(selector); FI:
            IF return code segment selector RPL > CPL
                THEN GOTO IA-32E-MODE-RETURN-TO-OUTER-PRIVILEGE-LEVEL;
                ELSE GOTO IA-32E-MODE-RETURN-SAME-PRIVILEGE-LEVEL;
            FI;
```

```
    FI;

IA-32E-MODE-RETURN-SAME-PRIVILEGE-LEVEL:
IF the return instruction pointer is not within the return code segment limit
    THEN #GP(0); FI;
IF the return instruction pointer is not within canonical address space
    THEN #GP(0); FI;
IF OperandSize = 32
    THEN
        EIP ← Pop();
        CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
    ELSE
        IF OperandSize = 16
            THEN
                EIP ← Pop();
                EIP ← EIP AND 0000FFFFH;
                CS ← Pop(); (* 16-bit pop *)
            ELSE (* OperandSize = 64 *)
                RIP ← Pop();
                CS ← Pop(); (* 64-bit pop, high-order 48 bits discarded *)
        FI;
FI;
IF instruction has immediate operand
    THEN (* Release parameters from stack *)
        IF StackAddressSize = 32
            THEN
                ESP ← ESP + SRC;
            ELSE
                IF StackAddressSize = 16
                    THEN
                        SP ← SP + SRC;
                    ELSE (* StackAddressSize = 64 *)
                        RSP ← RSP + SRC;
                FI;
        FI;
FI;

IA-32E-MODE-RETURN-TO-OUTER-PRIVILEGE-LEVEL:
IF top (16 + SRC) bytes of stack are not within stack limits (OperandSize = 32)
or top (8 + SRC) bytes of stack are not within stack limits (OperandSize = 16)
    THEN #SS(0); FI;
IF top (16 + SRC) bytes of stack are not in canonical address space (OperandSize = 32)
or top (8 + SRC) bytes of stack are not in canonical address space (OperandSize = 16)
or top (32 + SRC) bytes of stack are not in canonical address space (OperandSize = 64)
    THEN #SS(0); FI;
Read return stack segment selector;
IF stack segment selector is NULL
    THEN
        IF new CS descriptor L-bit = 0
            THEN #GP(selector);
```

```
        IF stack segment selector RPL = 3
            THEN #GP(selector);
FI;
IF return stack segment descriptor is not within descriptor table limits
        THEN #GP(selector); FI;
IF return stack segment descriptor is in non-canonical address space
        THEN #GP(selector); FI;
Read segment descriptor pointed to by return segment selector;
IF stack segment selector RPL ≠ RPL of the return code segment selector
or stack segment is not a writable data segment
or stack segment descriptor DPL ≠ RPL of the return code segment selector
    THEN #GP(selector); FI;
IF stack segment not present
    THEN #SS(StackSegmentSelector); FI;
IF the return instruction pointer is not within the return code segment limit
    THEN #GP(0); FI:
IF the return instruction pointer is not within canonical address space
    THEN #GP(0); FI;
CPL ← ReturnCodeSegmentSelector(RPL);
IF OperandSize = 32
    THEN
        EIP ← Pop();
        CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded, segment descriptor loaded *)
        CS(RPL) ← CPL;
        IF instruction has immediate operand
            THEN (* Release parameters from called procedure's stack *)
                IF StackAddressSize = 32
                    THEN
                        ESP ← ESP + SRC;
                    ELSE
                        IF StackAddressSize = 16
                            THEN
                                SP ← SP + SRC;
                            ELSE (* StackAddressSize = 64 *)
                                RSP ← RSP + SRC;
                        FI;
                FI;
        FI;
        tempESP ← Pop();
        tempSS ← Pop(); (* 32-bit pop, high-order 16 bits discarded, segment descriptor loaded *)
        ESP ← tempESP;
        SS ← tempSS;
    ELSE
        IF OperandSize = 16
            THEN
                EIP ← Pop();
                EIP ← EIP AND 0000FFFFH;
                CS ← Pop(); (* 16-bit pop; segment descriptor loaded *)
                CS(RPL) ← CPL;
                IF instruction has immediate operand
```

```
                    THEN (* Release parameters from called procedure's stack *)
                        IF StackAddressSize = 32
                            THEN
                                ESP ← ESP + SRC;
                            ELSE
                                IF StackAddressSize = 16
                                    THEN
                                        SP ← SP + SRC;
                                    ELSE (* StackAddressSize = 64 *)
                                        RSP ← RSP + SRC;
                                FI;
                        FI;
                FI;
                tempESP ← Pop();
                tempSS ← Pop(); (* 16-bit pop; segment descriptor loaded *)
                ESP ← tempESP;
                SS ← tempSS;
            ELSE (* OperandSize = 64 *)
                RIP ← Pop();
                CS ← Pop(); (* 64-bit pop; high-order 48 bits discarded; seg. descriptor loaded *)
                CS(RPL) ← CPL;
                IF instruction has immediate operand
                    THEN (* Release parameters from called procedure's stack *)
                        RSP ← RSP + SRC;
                FI;
                tempESP ← Pop();
                tempSS ← Pop(); (* 64-bit pop; high-order 48 bits discarded; seg. desc. loaded *)
                ESP ← tempESP;
                SS ← tempSS;
        FI;
FI;

FOR each of segment register (ES, FS, GS, and DS)
    DO
        IF segment register points to data or non-conforming code segment
        and CPL > segment descriptor DPL; (* DPL in hidden part of segment register *)
            THEN SegmentSelector ← 0; (* SegmentSelector invalid *)
        FI;
    OD;

IF instruction has immediate operand
    THEN (* Release parameters from calling procedure's stack *)
        IF StackAddressSize = 32
            THEN
                ESP ← ESP + SRC;
            ELSE
                IF StackAddressSize = 16
                    THEN
                        SP ← SP + SRC;
                    ELSE (* StackAddressSize = 64 *)
```

```
                    RSP ← RSP + SRC;
                FI;
        FI;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the return code or stack segment selector NULL.
	If the return instruction pointer is not within the return code segment limit
#GP(selector)	If the RPL of the return code segment selector is less then the CPL.
	If the return code or stack segment selector index is not within its descriptor table limits.
	If the return code segment descriptor does not indicate a code segment.
	If the return code segment is non-conforming and the segment selector's DPL is not equal to the RPL of the code segment's segment selector
	If the return code segment is conforming and the segment selector's DPL greater than the RPL of the code segment's segment selector
	If the stack segment is not a writable data segment.
	If the stack segment selector RPL is not equal to the RPL of the return code segment selector.
	If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector.
#SS(0)	If the top bytes of stack are not within stack limits.
	If the return stack segment is not present.
#NP(selector)	If the return code segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory access occurs when the CPL is 3 and alignment checking is enabled.

Real-Address Mode Exceptions

#GP	If the return instruction pointer is not within the return code segment limit
#SS	If the top bytes of stack are not within stack limits.

Virtual-8086 Mode Exceptions

#GP(0)	If the return instruction pointer is not within the return code segment limit
#SS(0)	If the top bytes of stack are not within stack limits.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory access occurs when alignment checking is enabled.

Compatibility Mode Exceptions

Same as 64-bit mode exceptions.

64-Bit Mode Exceptions

#GP(0)	If the return instruction pointer is non-canonical.
	If the return instruction pointer is not within the return code segment limit.
	If the stack segment selector is NULL going back to compatibility mode.

	If the stack segment selector is NULL going back to CPL3 64-bit mode.
	If a NULL stack segment selector RPL is not equal to CPL going back to non-CPL3 64-bit mode.
	If the return code segment selector is NULL.
#GP(selector)	If the proposed segment descriptor for a code segment does not indicate it is a code segment.
	If the proposed new code segment descriptor has both the D-bit and L-bit set.
	If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector.
	If CPL is greater than the RPL of the code segment selector.
	If the DPL of a conforming-code segment is greater than the return code segment selector RPL.
	If a segment selector index is outside its descriptor table limits.
	If a segment descriptor memory address is non-canonical.
	If the stack segment is not a writable data segment.
	If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector.
	If the stack segment selector RPL is not equal to the RPL of the return code segment selector.
#SS(0)	If an attempt to pop a value off the stack violates the SS limit.
	If an attempt to pop a value off the stack causes a non-canonical address to be referenced.
#NP(selector)	If the return code or stack segment is 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.

...

VFMADD132SD/VFMADD213SD/VFMADD231SD — Fused Multiply-Add of Scalar Double-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W1 99 /r VFMADD132SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm2/mem*, add to *xmm1* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 A9 /r VFMADD213SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm1*, add to *xmm2/mem* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 B9 /r VFMADD231SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm1* and *xmm2/mem*, add to *xmm0* and put result in *xmm0*.

Instruction Operand Encoding

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

Description

Performs a SIMD multiply-add computation on the low packed double-precision floating-point values using three source operands and writes the multiply-add result in the destination operand. The destination operand is also the first source operand. The second operand must be a SIMD register. The third source operand can be a SIMD register or a memory location.

VFMADD132SD: Multiplies the low packed double-precision floating-point value from the first source operand to the low packed double-precision floating-point value in the third source operand, adds the infinite precision intermediate result to the low packed double-precision floating-point values in the second source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFMADD213SD: Multiplies the low packed double-precision floating-point value from the second source operand to the low packed double-precision floating-point value in the first source operand, adds the infinite precision intermediate result to the low packed double-precision floating-point value in the third source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFMADD231SD: Multiplies the low packed double-precision floating-point value from the second source to the low packed double-precision floating-point value in the third source operand, adds the infinite precision intermediate result to the low packed double-precision floating-point value in the first source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 64-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFMADD132SD DEST, SRC2, SRC3
DEST[63:0] ← RoundFPControl_MXCSR(DEST[63:0]*SRC3[63:0] + SRC2[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFMADD213SD DEST, SRC2, SRC3
DEST[63:0] ← RoundFPControl_MXCSR(SRC2[63:0]*DEST[63:0] + SRC3[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFMADD231SD DEST, SRC2, SRC3
DEST[63:0] ← RoundFPControl_MXCSR(SRC2[63:0]*SRC3[63:0] + DEST[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFMADD132SD: __m128d _mm_fmadd_sd (__m128d a, __m128d b, __m128d c);

VFMADD213SD: __m128d _mm_fmadd_sd (__m128d a, __m128d b, __m128d c);

VFMADD231SD: __m128d _mm_fmadd_sd (__m128d a, __m128d b, __m128d c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VFMADD132SS/VFMADD213SS/VFMADD231SS — Fused Multiply-Add of Scalar Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W0 99 /r VFMADD132SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm0* and *xmm2/mem*, add to *xmm1* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W0 A9 /r VFMADD213SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm0* and *xmm1*, add to *xmm2/mem* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W0 B9 /r VFMADD231SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm1* and *xmm2/mem*, add to *xmm0* and put result in *xmm0*.

Instruction Operand Encoding

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

Description

Performs a SIMD multiply-add computation on packed single-precision floating-point values using three source operands and writes the multiply-add results in the destination operand. The destination operand is also the first source operand. The second operand must be a SIMD register. The third source operand can be a SIMD register or a memory location.

VFMADD132SS: Multiplies the low packed single-precision floating-point value from the first source operand to the low packed single-precision floating-point value in the third source operand, adds the infinite precision intermediate result to the low packed single-precision floating-point value in the second source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFMADD213SS: Multiplies the low packed single-precision floating-point value from the second source operand to the low packed single-precision floating-point value in the first source operand, adds the infinite precision intermediate result to the low packed single-precision floating-point value in the third source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFMADD231SS: Multiplies the low packed single-precision floating-point value from the second source operand to the low packed single-precision floating-point value in the third source operand, adds the infinite precision intermediate result to the low packed single-precision floating-point value in the first source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 32-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFMADD132SS DEST, SRC2, SRC3

DEST[31:0] ← RoundFPControl_MXCSR(DEST[31:0]*SRC3[31:0] + SRC2[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

VFMADD213SS DEST, SRC2, SRC3

DEST[31:0] ← RoundFPControl_MXCSR(SRC2[31:0]*DEST[31:0] + SRC3[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

VFMADD231SS DEST, SRC2, SRC3

DEST[31:0] ← RoundFPControl_MXCSR(SRC2[31:0]*SRC3[63:0] + DEST[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFMADD132SS: __m128 _mm_fmadd_ss (__m128 a, __m128 b, __m128 c);

VFMADD213SS: __m128 _mm_fmadd_ss (__m128 a, __m128 b, __m128 c);

VFMADD231SS: __m128 _mm_fmadd_ss (__m128 a, __m128 b, __m128 c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VFMSUB132SD/VFMSUB213SD/VFMSUB231SD — Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W1 9B /r VFMSUB132SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm2/mem*, subtract *xmm1* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 AB /r VFMSUB213SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm1*, subtract *xmm2/mem* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 BB /r VFMSUB231SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm1* and *xmm2/mem*, subtract *xmm0* and put result in *xmm0*.

Instruction Operand Encoding

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

Description

Performs a SIMD multiply-subtract computation on the low packed double-precision floating-point values using three source operands and writes the multiply-add result in the destination operand. The destination operand is also the first source operand. The second operand must be a SIMD register. The third source operand can be a SIMD register or a memory location.

VFMSUB132SD: Multiplies the low packed double-precision floating-point value from the first source operand to the low packed double-precision floating-point value in the third source operand. From the infinite precision intermediate result, subtracts the low packed double-precision floating-point values in the second source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFMSUB213SD: Multiplies the low packed double-precision floating-point value from the second source operand to the low packed double-precision floating-point value in the first source operand. From the infinite precision intermediate result, subtracts the low packed double-precision floating-point value in the third source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFMSUB231SD: Multiplies the low packed double-precision floating-point value from the second source to the low packed double-precision floating-point value in the third source operand. From the infinite precision intermediate result, subtracts the low packed double-precision floating-point value in the first source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 64-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFMSUB132SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(DEST[63:0]*SRC3[63:0] - SRC2[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFMSUB213SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(SRC2[63:0]*DEST[63:0] - SRC3[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFMSUB231SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(SRC2[63:0]*SRC3[63:0] - DEST[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFMSUB132SD: __m128d _mm_fmsub_sd (__m128d a, __m128d b, __m128d c);

VFMSUB213SD: __m128d _mm_fmsub_sd (__m128d a, __m128d b, __m128d c);

VFMSUB231SD: __m128d _mm_fmsub_sd (__m128d a, __m128d b, __m128d c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VFMSUB132SS/VFMSUB213SS/VFMSUB231SS — Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W0 9B /r VFMSUB132SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm0* and *xmm2/mem*, subtract *xmm1* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W0 AB /r VFMSUB213SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm0* and *xmm1*, subtract *xmm2/mem* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W0 BB /r VFMSUB231SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm1* and *xmm2/mem*, subtract *xmm0* and put result in *xmm0*.

Instruction Operand Encoding

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

Description

Performs a SIMD multiply-subtract computation on the low packed single-precision floating-point values using three source operands and writes the multiply-add result in the destination operand. The destination operand is also the first source operand. The second operand must be a SIMD register. The third source operand can be a SIMD register or a memory location.

VFMSUB132SS: Multiplies the low packed single-precision floating-point value from the first source operand to the low packed single-precision floating-point value in the third source operand. From the infinite precision intermediate result, subtracts the low packed single-precision floating-point values in the second source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFMSUB213SS: Multiplies the low packed single-precision floating-point value from the second source operand to the low packed single-precision floating-point value in the first source operand. From the infinite precision intermediate result, subtracts the low packed single-precision floating-point value in the third source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFMSUB231SS: Multiplies the low packed single-precision floating-point value from the second source to the low packed single-precision floating-point value in the third source operand. From the infinite precision intermediate result, subtracts the low packed single-precision floating-point value in the first source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 32-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFMSUB132SS DEST, SRC2, SRC3

DEST[31:0] ← RoundFPControl_MXCSR(DEST[31:0]*SRC3[31:0] - SRC2[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

VFMSUB213SS DEST, SRC2, SRC3

DEST[31:0] ← RoundFPControl_MXCSR(SRC2[31:0]*DEST[31:0] - SRC3[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

VFMSUB231SS DEST, SRC2, SRC3

DEST[31:0] ← RoundFPControl_MXCSR(SRC2[31:0]*SRC3[63:0] - DEST[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFMSUB132SS: __m128 _mm_fmsub_ss (__m128 a, __m128 b, __m128 c);

VFMSUB213SS: __m128 _mm_fmsub_ss (__m128 a, __m128 b, __m128 c);

VFMSUB231SS: __m128 _mm_fmsub_ss (__m128 a, __m128 b, __m128 c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VFNMADD132SD/VFNMADD213SD/VFNMADD231SD — Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W1 9D /r VFNMADD132SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm2/mem*, negate the multiplication result and add to *xmm1* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 AD /r VFNMADD213SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm1*, negate the multiplication result and add to *xmm2/mem* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 BD /r VFNMADD231SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm1* and *xmm2/mem*, negate the multiplication result and add to *xmm0* and put result in *xmm0*.

Instruction Operand Encoding

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

Description

VFNMADD132SD: Multiplies the low packed double-precision floating-point value from the first source operand to the low packed double-precision floating-point value in the third source operand, adds the negated infinite precision intermediate result to the low packed double-precision floating-point values in the second source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFNMADD213SD: Multiplies the low packed double-precision floating-point value from the second source operand to the low packed double-precision floating-point value in the first source operand, adds the negated infinite precision intermediate result to the low packed double-precision floating-point value in the third source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFNMADD231SD: Multiplies the low packed double-precision floating-point value from the second source to the low packed double-precision floating-point value in the third source operand, adds the negated infinite precision intermediate result to the low packed double-precision floating-point value in the first source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 64-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFNMADD132SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(- (DEST[63:0]*SRC3[63:0]) + SRC2[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFNMADD213SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(- (SRC2[63:0]*DEST[63:0]) + SRC3[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFNMADD231SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(- (SRC2[63:0]*SRC3[63:0]) + DEST[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFNMADD132SD: __m128d _mm_fnmadd_sd (__m128d a, __m128d b, __m128d c);

VFNMADD213SD: __m128d _mm_fnmadd_sd (__m128d a, __m128d b, __m128d c);

VFNMADD231SD: __m128d _mm_fnmadd_sd (__m128d a, __m128d b, __m128d c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VFNMADD132SS/VFNMADD213SS/VFNMADD231SS — Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W0 9D /r VFNMADD132SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm0* and *xmm2/mem*, negate the multiplication result and add to *xmm1* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W0 AD /r VFNMADD213SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm0* and *xmm1*, negate the multiplication result and add to *xmm2/mem* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W0 BD /r VFNMADD231SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from *xmm1* and *xmm2/mem*, negate the multiplication result and add to *xmm0* and put result in *xmm0*.

Instruction Operand Encoding

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

Description

VFNMADD132SS: Multiplies the low packed single-precision floating-point value from the first source operand to the low packed single-precision floating-point value in the third source operand, adds the negated infinite precision intermediate result to the low packed single-precision floating-point value in the second source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFNMADD213SS: Multiplies the low packed single-precision floating-point value from the second source operand to the low packed single-precision floating-point value in the first source operand, adds the negated infinite precision intermediate result to the low packed single-precision floating-point value in the third source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFNMADD231SS: Multiplies the low packed single-precision floating-point value from the second source operand to the low packed single-precision floating-point value in the third source operand, adds the negated infinite precision intermediate result to the low packed single-precision floating-point value in the first source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 32-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFNMADD132SS DEST, SRC2, SRC3
DEST[31:0] ← RoundFPControl_MXCSR(- (DEST[31:0]*SRC3[31:0]) + SRC2[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

VFNMADD213SS DEST, SRC2, SRC3
DEST[31:0] ← RoundFPControl_MXCSR(- (SRC2[31:0]*DEST[31:0]) + SRC3[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

VFNMADD231SS DEST, SRC2, SRC3
DEST[31:0] ← RoundFPControl_MXCSR(- (SRC2[31:0]*SRC3[63:0]) + DEST[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFNMADD132SS: __m128 _mm_fnmadd_ss (__m128 a, __m128 b, __m128 c);

VFNMADD213SS: __m128 _mm_fnmadd_ss (__m128 a, __m128 b, __m128 c);

VFNMADD231SS: __m128 _mm_fnmadd_ss (__m128 a, __m128 b, __m128 c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VFNMSUB132SD/VFNMSUB213SD/VFNMSUB231SD — Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W1 9F /r VFNMSUB132SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm2/mem*, negate the multiplication result and subtract *xmm1* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 AF /r VFNMSUB213SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm0* and *xmm1*, negate the multiplication result and subtract *xmm2/mem* and put result in *xmm0*.
VEX.DDS.LIG.66.0F38.W1 BF /r VFNMSUB231SD *xmm0, xmm1, xmm2/m64*	A	V/V	FMA	Multiply scalar double-precision floating-point value from *xmm1* and *xmm2/mem*, negate the multiplication result and subtract *xmm0* and put result in *xmm0*.

Instruction Operand Encoding

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

Description

VFNMSUB132SD: Multiplies the low packed double-precision floating-point value from the first source operand to the low packed double-precision floating-point value in the third source operand. From negated infinite precision intermediate result, subtracts the low double-precision floating-point value in the second source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFNMSUB213SD: Multiplies the low packed double-precision floating-point value from the second source operand to the low packed double-precision floating-point value in the first source operand. From negated infinite precision intermediate result, subtracts the low double-precision floating-point value in the third source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VFNMSUB231SD: Multiplies the low packed double-precision floating-point value from the second source to the low packed double-precision floating-point value in the third source operand. From negated infinite precision intermediate result, subtracts the low double-precision floating-point value in the first source operand, performs rounding and stores the resulting packed double-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 64-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFNMSUB132SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(- (DEST[63:0]*SRC3[63:0]) - SRC2[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFNMSUB213SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(- (SRC2[63:0]*DEST[63:0]) - SRC3[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

VFNMSUB231SD DEST, SRC2, SRC3

DEST[63:0] ← RoundFPControl_MXCSR(- (SRC2[63:0]*SRC3[63:0]) - DEST[63:0])
DEST[127:64] ← DEST[127:64]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFNMSUB132SD: __m128d _mm_fnmsub_sd (__m128d a, __m128d b, __m128d c);

VFNMSUB213SD: __m128d _mm_fnmsub_sd (__m128d a, __m128d b, __m128d c);

VFNMSUB231SD: __m128d _mm_fnmsub_sd (__m128d a, __m128d b, __m128d c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VFNMSUB132SS/VFNMSUB213SS/VFNMSUB231SS — Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.LIG.66.0F38.W0 9F /r VFNMSUB132SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from xmm0 and xmm2/mem, negate the multiplication result and subtract xmm1 and put result in xmm0.
VEX.DDS.LIG.66.0F38.W0 AF /r VFNMSUB213SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from xmm0 and xmm1, negate the multiplication result and subtract xmm2/mem and put result in xmm0.
VEX.DDS.LIG.66.0F38.W0 BF /r VFNMSUB231SS *xmm0, xmm1, xmm2/m32*	A	V/V	FMA	Multiply scalar single-precision floating-point value from xmm1 and xmm2/mem, negate the multiplication result and subtract xmm0 and put result in xmm0.

Instruction Operand Encoding

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

Description

VFNMSUB132SS: Multiplies the low packed single-precision floating-point value from the first source operand to the low packed single-precision floating-point value in the third source operand. From negated infinite precision intermediate result, the low single-precision floating-point value in the second source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFNMSUB213SS: Multiplies the low packed single-precision floating-point value from the second source operand to the low packed single-precision floating-point value in the first source operand. From negated infinite precision intermediate result, the low single-precision floating-point value in the third source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VFNMSUB231SS: Multiplies the low packed single-precision floating-point value from the second source to the low packed single-precision floating-point value in the third source operand. From negated infinite precision intermediate result, the low single-precision floating-point value in the first source operand, performs rounding and stores the resulting packed single-precision floating-point value to the destination operand (first source operand).

VEX.128 encoded version: The destination operand (also first source operand) is a XMM register and encoded in reg_field. The second source operand is a XMM register and encoded in VEX.vvvv. The third source operand is a XMM register or a 32-bit memory location and encoded in rm_field. The upper bits ([VLMAX-1:128]) of the YMM destination register are zeroed.

Compiler tools may optionally support a complementary mnemonic for each instruction mnemonic listed in the opcode/instruction column of the summary table. The behavior of the complementary mnemonic in situations involving NANs are governed by the definition of the instruction mnemonic defined in the opcode/instruction column. See also Section 14.5.1, "FMA Instruction Operand Order and Arithmetic Behavior" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operation

In the operations below, "+", "-", and "*" symbols represent addition, subtraction, and multiplication operations with infinite precision inputs and outputs (no rounding).

VFNMSUB132SS DEST, SRC2, SRC3
DEST[31:0] ← RoundFPControl_MXCSR(- (DEST[31:0]*SRC3[31:0]) - SRC2[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0
VFNMSUB213SS DEST, SRC2, SRC3
DEST[31:0] ← RoundFPControl_MXCSR(- (SRC2[31:0]*DEST[31:0]) - SRC3[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

VFNMSUB231SS DEST, SRC2, SRC3
DEST[31:0] ← RoundFPControl_MXCSR(- (SRC2[31:0]*SRC3[63:0]) - DEST[31:0])
DEST[127:32] ← DEST[127:32]
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VFNMSUB132SS: __m128 _mm_fnmsub_ss (__m128 a, __m128 b, __m128 c);

VFNMSUB213SS: __m128 _mm_fnmsub_ss (__m128 a, __m128 b, __m128 c);

VFNMSUB231SS: __m128 _mm_fnmsub_ss (__m128 a, __m128 b, __m128 c);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Other Exceptions

See Exceptions Type 3

...

VINSERTF128 — Insert Packed Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
VEX.NDS.256.66.0F3A.W0 18 /r ib VINSERTF128 *ymm1, ymm2, xmm3/m128, imm8*	RVM	V/V	AVX	Insert 128-bits of floating point data selected by *imm8* from *xmm3/m128* and the remaining values from *ymm2* into *ymm1.*

Instruction Operand Encoding

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

Description

Performs an insertion of 128-bits of packed floating-point values from the second source operand (third operand) into an the destination operand (first operand) at an 128-bit offset from imm8[0]. The remaining portions of the destination are written by the corresponding fields of the first source operand (second operand). The second source operand can be either an XMM register or a 128-bit memory location.

The high 7 bits of the immediate are ignored.

Operation

```
TEMP[255:0] ← SRC1[255:0]
CASE (imm8[0]) OF
    0: TEMP[127:0] ← SRC2[127:0]
    1: TEMP[255:128] ← SRC2[127:0]
ESAC
DEST ←TEMP
```

Intel C/C++ Compiler Intrinsic Equivalent

VINSERTF128: __m256 _mm256_insertf128_ps (__m256 a, __m128 b, int offset);

VINSERTF128: __m256d _mm256_insertf128_pd (__m256d a, __m128d b, int offset);

VINSERTF128: __m256i _mm256_insertf128_si256 (__m256i a, __m128i b, int offset);

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 6; additionally

#UD	If VEX.W = 1.

...

XSAVES—Save Processor Extended States Supervisor

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F C7 /5	XSAVES *mem*	M	Valid	Valid	Save state components specified by EDX:EAX to *mem* with compaction, optimizing if possible.
REX.W+ 0F C7 /5	XSAVES64 *mem*	M	Valid	N.E.	Save state components specified by EDX:EAX to *mem* with compaction, optimizing if possible.

Instruction Operand Encoding

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

Description

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

The format of the XSAVE area is detailed in Section 13.4, "XSAVE Area," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Section 13.11, "Operation of XSAVES," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* provides a detailed description of the operation of the XSAVES instruction. The following items provide a high-level outline:

- Execution of XSAVES is similar to that of XSAVEC. XSAVES differs from XSAVEC in that it can save state components corresponding to bits set in the IA32_XSS MSR and that it may use the modified optimization.
- XSAVES saves state component *i* only if RFBM[*i*] = 1 and XINUSE[*i*] = 1.[1] (XINUSE is a bitmap by which the processor tracks the status of various state components. See Section 13.6, "Processor Tracking of XSAVE-Managed State.") Even if both bits are 1, XSAVES may optimize and not save state component *i* if (1) state component *i* has not been modified since the last execution of XRTOR or XRSTORS; and (2) this execution of XSAVES correspond to that last execution of XRTOR or XRSTORS as determined by XRSTOR_INFO (see the Operation section below).
- XSAVES does not modify bytes 511:464 of the legacy region of the XSAVE area (see Section 13.4.1, "Legacy Region of an XSAVE Area").
- XSAVES writes the logical AND of RFBM and XINUSE to the XSTATE_BV field of the XSAVE header.[2] (See Section 13.4.2, "XSAVE Header.") XSAVES sets bit 63 of the XCOMP_BV field and sets bits 62:0 of that field to RFBM[62:0]. XSAVES does not write to any parts of the XSAVE header other than the XSTATE_BV and XCOMP_BV fields.
- XSAVES always uses the compacted format of the extended region of the XSAVE area (see Section 13.4.3, "Extended Region of an XSAVE Area").

1. There is an exception for state component 1 (SSE). MXCSR is part of SSE state, but XINUSE[1] may be 0 even if MXCSR does not have its initial value of 1F80H. In this case, the init optimization does not apply and XSAVEC will save SSE state as long as RFBM[1] = 1 and the modified optimization is not being applied.
2. There is an exception for state component 1 (SSE). MXCSR is part of SSE state, but XINUSE[1] may be 0 even if MXCSR does not have its initial value of 1F80H. In this case, XSAVES sets XSTATE_BV[1] to 1 as long as RFBM[1] = 1.

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

Operation

```
RFBM ← (XCR0 OR IA32_XSS) AND EDX:EAX;          /* bitwise logical OR and AND */
IF in VMX non-root operation
    THEN VMXNR ← 1;
    ELSE VMXNR ← 0;
FI;
LAXA ← linear address of XSAVE area;
COMPMASK ← RFBM OR 80000000_00000000H;
IF XRSTOR_INFO = ⟨CPL,VMXNR,LAXA,COMPMASK⟩
    THEN MODOPT ← 1;
    ELSE MODOPT ← 0;
FI;

IF RFBM[0] = 1 and XINUSE[0] = 1
    THEN store x87 state into legacy region of XSAVE area;
    /* might avoid saving if x87 state is not modified and MODOPT = 1 */
FI;
IF RFBM[1] = 1 and (XINUSE[1] = 1 or MXCSR ≠ 1F80H)
    THEN store SSE state into legacy region of XSAVE area;
    /* might avoid saving if SSE state is not modified and MODOPT = 1 */
FI;
IF RFBM[2] = 1 AND XINUSE[2] = 1
    THEN store AVX state into extended region of XSAVE area;
    /* might avoid saving if AVX state is not modified and MODOPT = 1 */
FI;

XSTATE_BV field in XSAVE header ← XINUSE AND RFBM;[1]
XCOMP_BV field in XSAVE header ← COMPMASK;
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XSAVES: void _xsaves(void * , unsigned __int64);

XSAVES64: void _xsaves64(void * , unsigned __int64);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.

1. If MXCSR does not have its initial value of 1F80H, XSAVES sets XSTATE_BV[1] to 1 as long as RFBM[1] = 1, regardless of the value of XINUSE[1].

#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSS[bit 3] = 0. If CR4.OSXSAVE[bit 18] = 0. If any of the LOCK, 66H, F3H or F2H prefixes is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 64-byte boundary, regardless of segment. If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSS[bit 3] = 0. If CR4.OSXSAVE[bit 18] = 0. If any of the LOCK, 66H, F3H or F2H prefixes is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form. If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSS[bit 3] = 0. If CR4.OSXSAVE[bit 18] = 0. If any of the LOCK, 66H, F3H or F2H prefixes is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte misalignments).

...

9. Updates to Appendix B, Volume 2C

Change bars show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C:* Appendices.

...

B.13 SSE4.2 FORMATS AND ENCODING TABLE

The tables in this section provide SSE4.2 formats and encodings. Some SSE4.2 instructions require a mandatory prefix (66H, F2H, F3H) as part of the three-byte opcode. These prefixes are included in the tables. In 64-bit mode, some instructions requires REX.W, the byte sequence of REX.W prefix in the opcode sequence is shown.

Table B-36. Encodings of SSE4.2 instructions

Instruction and Format	Encoding
CRC32 — Accumulate CRC32	
reg2 to reg1	1111 0010:0000 1111:0011 1000: 1111 000w :11 reg1 reg2
mem to reg	1111 0010:0000 1111:0011 1000: 1111 000w : mod reg r/m
bytereg2 to reg1	1111 0010:0100 WR0B:0000 1111:0011 1000: 1111 0000 :11 reg1 bytereg2
m8 to reg	1111 0010:0100 WR0B:0000 1111:0011 1000: 1111 0000 : mod reg r/m
qwreg2 to qwreg1	1111 0010:0100 1R0B:0000 1111:0011 1000: 1111 0001 :11 qwreg1 qwreg2
mem64 to qwreg	1111 0010:0100 1R0B:0000 1111:0011 1000: 1111 0001 : mod qwreg r/m
PCMPESTRI— Packed Compare Explicit-Length Strings To Index	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0001: mod xmmreg r/m
PCMPESTRM— Packed Compare Explicit-Length Strings To Mask	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0000: mod xmmreg r/m
PCMPISTRI— Packed Compare Implicit-Length String To Index	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0011:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0011: mod xmmreg r/m

Table B-36. Encodings of SSE4.2 instructions

Instruction and Format	Encoding
PCMPISTRM— Packed Compare Implicit-Length Strings To Mask	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0010: mod xmmreg r/m
PCMPGTQ— Packed Compare Greater Than	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0111: mod xmmreg r/m
POPCNT— Return Number of Bits Set to 1	
reg2 to reg1	1111 0011:0000 1111:1011 1000:11 reg1 reg2
mem to reg1	1111 0011:0000 1111:1011 1000:mod reg1 r/m
qwreg2 to qwreg1	1111 0011:0100 1R0B:0000 1111:1011 1000:11 reg1 reg2
mem64 to qwreg1	1111 0011:0100 1R0B:0000 1111:1011 1000:mod reg1 r/m

...

10. 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™ processor family
- 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® Atom™ processor 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™ processor family is 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 32nm 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 and 5th generation Intel® Core™ processors 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.

1.2 OVERVIEW OF THE SYSTEM PROGRAMMING GUIDE

A description of this manual's content follows:

Chapter 1 — About This Manual. Gives an overview of all seven volumes of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*. It also describes the notational conventions in these manuals and lists related Intel manuals and documentation of interest to programmers and hardware designers.

Chapter 2 — System Architecture Overview. Describes the modes of operation used by Intel 64 and IA-32 processors and the mechanisms provided by the architectures to support operating systems and executives, including the system-oriented registers and data structures and the system-oriented instructions. The steps necessary for switching between real-address and protected modes are also identified.

Chapter 3 — Protected-Mode Memory Management. Describes the data structures, registers, and instructions that support segmentation and paging. The chapter explains how they can be used to implement a "flat" (unsegmented) memory model or a segmented memory model.

Chapter 4 — Paging. Describes the paging modes supported by Intel 64 and IA-32 processors.

Chapter 5 — Protection. Describes the support for page and segment protection provided in the Intel 64 and IA-32 architectures. This chapter also explains the implementation of privilege rules, stack switching, pointer validation, user and supervisor modes.

Chapter 6 — Interrupt and Exception Handling. Describes the basic interrupt mechanisms defined in the Intel 64 and IA-32 architectures, shows how interrupts and exceptions relate to protection, and describes how the architecture handles each exception type. Reference information for each exception is given in this chapter. Includes programming the LINT0 and LINT1 inputs and gives an example of how to program the LINT0 and LINT1 pins for specific interrupt vectors.

Chapter 7 — Task Management. Describes mechanisms the Intel 64 and IA-32 architectures provide to support multitasking and inter-task protection.

Chapter 8 — Multiple-Processor Management. Describes the instructions and flags that support multiple processors with shared memory, memory ordering, and Intel® Hyper-Threading Technology. Includes MP initialization for P6 family processors and gives an example of how to use of the MP protocol to boot P6 family processors in an MP system.

Chapter 9 — Processor Management and Initialization. Defines the state of an Intel 64 or IA-32 processor after reset initialization. This chapter also explains how to set up an Intel 64 or IA-32 processor for real-address mode operation and protected- mode operation, and how to switch between modes.

Chapter 10 — Advanced Programmable Interrupt Controller (APIC). Describes the programming interface to the local APIC and gives an overview of the interface between the local APIC and the I/O APIC. Includes APIC bus message formats and describes the message formats for messages transmitted on the APIC bus for P6 family and Pentium processors.

Chapter 11 — Memory Cache Control. Describes the general concept of caching and the caching mechanisms supported by the Intel 64 or IA-32 architectures. This chapter also describes the memory type range registers (MTRRs) and how they can be used to map memory types of physical memory. Information on using the new cache control and memory streaming instructions introduced with the Pentium III, Pentium 4, and Intel Xeon processors is also given.

Chapter 12 — Intel® MMX™ Technology System Programming. Describes those aspects of the Intel® MMX™ technology that must be handled and considered at the system programming level, including: task switching, exception handling, and compatibility with existing system environments.

Chapter 13 — System Programming For Instruction Set Extensions And Processor Extended States. Describes the operating system requirements to support SSE/SSE2/SSE3/SSSE3/SSE4 extensions, including task switching, exception handling, and compatibility with existing system environments. The latter part of this chapter describes the extensible framework of operating system requirements to support processor extended states. Processor extended state may be required by instruction set extensions beyond those of SSE/SSE2/SSE3/SSSE3/SSE4 extensions.

Chapter 14 — Power and Thermal Management. Describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

Chapter 15 — Machine-Check Architecture. Describes the machine-check architecture and machine-check exception mechanism found in the Pentium 4, Intel Xeon, and P6 family processors. Additionally, a signaling mechanism for software to respond to hardware corrected machine check error is covered.

Chapter 16 — Interpreting Machine-Check Error Codes. Gives an example of how to interpret the error codes for a machine-check error that occurred on a P6 family processor.

Chapter 17 — Debugging, Branch Profiles and Time-Stamp Counter. Describes the debugging registers and other debug mechanism provided in Intel 64 or IA-32 processors. This chapter also describes the time-stamp counter.

Chapter 18 — Performance Monitoring. Describes the Intel 64 and IA-32 architectures' facilities for monitoring performance.

Chapter 19 — Performance-Monitoring Events. Lists architectural performance events. Non-architectural performance events (i.e. model-specific events) are listed for each generation of microarchitecture.

Chapter 20 — 8086 Emulation. Describes the real-address and virtual-8086 modes of the IA-32 architecture.

Chapter 21 — Mixing 16-Bit and 32-Bit Code. Describes how to mix 16-bit and 32-bit code modules within the same program or task.

Chapter 22 — IA-32 Architecture Compatibility. Describes architectural compatibility among IA-32 processors.

Chapter 23 — Introduction to Virtual-Machine Extensions. Describes the basic elements of virtual machine architecture and the virtual-machine extensions for Intel 64 and IA-32 Architectures.

Chapter 24 — Virtual-Machine Control Structures. Describes components that manage VMX operation. These include the working-VMCS pointer and the controlling-VMCS pointer.

Chapter 25 — VMX Non-Root Operation. Describes the operation of a VMX non-root operation. Processor operation in VMX non-root mode can be restricted programmatically such that certain operations, events or conditions can cause the processor to transfer control from the guest (running in VMX non-root mode) to the monitor software (running in VMX root mode).

Chapter 26 — VM Entries. Describes VM entries. VM entry transitions the processor from the VMM running in VMX root-mode to a VM running in VMX non-root mode. VM-Entry is performed by the execution of VMLAUNCH or VMRESUME instructions.

Chapter 27 — VM Exits. Describes VM exits. Certain events, operations or situations while the processor is in VMX non-root operation may cause VM-exit transitions. In addition, VM exits can also occur on failed VM entries.

Chapter 28 — VMX Support for Address Translation. Describes virtual-machine extensions that support address translation and the virtualization of physical memory.

Chapter 29 — APIC Virtualization and Virtual Interrupts. Describes the VMCS including controls that enable the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC).

Chapter 30 — VMX Instruction Reference. Describes the virtual-machine extensions (VMX). VMX is intended for a system executive to support virtualization of processor hardware and a system software layer acting as a host to multiple guest software environments.

Chapter 31 — Virtual-Machine Monitoring Programming Considerations. Describes programming considerations for VMMs. VMMs manage virtual machines (VMs).

Chapter 32 — Virtualization of System Resources. Describes the virtualization of the system resources. These include: debugging facilities, address translation, physical memory, and microcode update facilities.

Chapter 33 — Handling Boundary Conditions in a Virtual Machine Monitor. Describes what a VMM must consider when handling exceptions, interrupts, error conditions, and transitions between activity states.

Chapter 34 — System Management Mode. Describes Intel 64 and IA-32 architectures' system management mode (SMM) facilities.

Chapter 35 — Model-Specific Registers (MSRs). Lists the MSRs available in the Pentium processors, the P6 family processors, the Pentium 4, Intel Xeon, Intel Core Solo, Intel Core Duo processors, and Intel Core 2 processor family and describes their functions.

Chapter 36 — Intel® Processor Trace. Describes details of Intel® Processor Trace.

Chapter 37 — Introduction to Intel® Software Guard Extensions. Provides an overview of the Intel® Software Guard Extensions (Intel® SGX) set of instructions.

Chapter 38 — Enclave Access Control and Data Structures. Describes Enclave Access Control procedures and defines various Intel SGX data structures.

Chapter 39 — Enclave Operation. Describes enclave creation and initialization, adding pages and measuring an enclave, and enclave entry and exit.

Chapter 40 — Enclave Exiting Events. Describes enclave-exiting events (EEE) and asynchronous enclave exit (AEX).

Chapter 41 — SGX Instruction References. Describes the supervisor and user level instructions provided by Intel SGX.

Chapter 42 — Intel® SGX Interactions with IA32 and Intel® 64 Architecture. Describes the Intel SGX collection of enclave instructions for creating protected execution environments on processors supporting IA32 and Intel 64 architectures.

Chapter 43 — Enclave Code Debug and Profiling. Describes enclave code debug processes and options.

Appendix A — VMX Capability Reporting Facility. Describes the VMX capability MSRs. Support for specific VMX features is determined by reading capability MSRs.

Appendix B — Field Encoding in VMCS. Enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.).

Appendix C — VM Basic Exit Reasons. Describes the 32-bit fields that encode reasons for a VM exit. Examples of exit reasons include, but are not limited to: software interrupts, processor exceptions, software traps, NMIs, external interrupts, and triple faults.

...

11.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:

- 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 are 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. 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.
- **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.

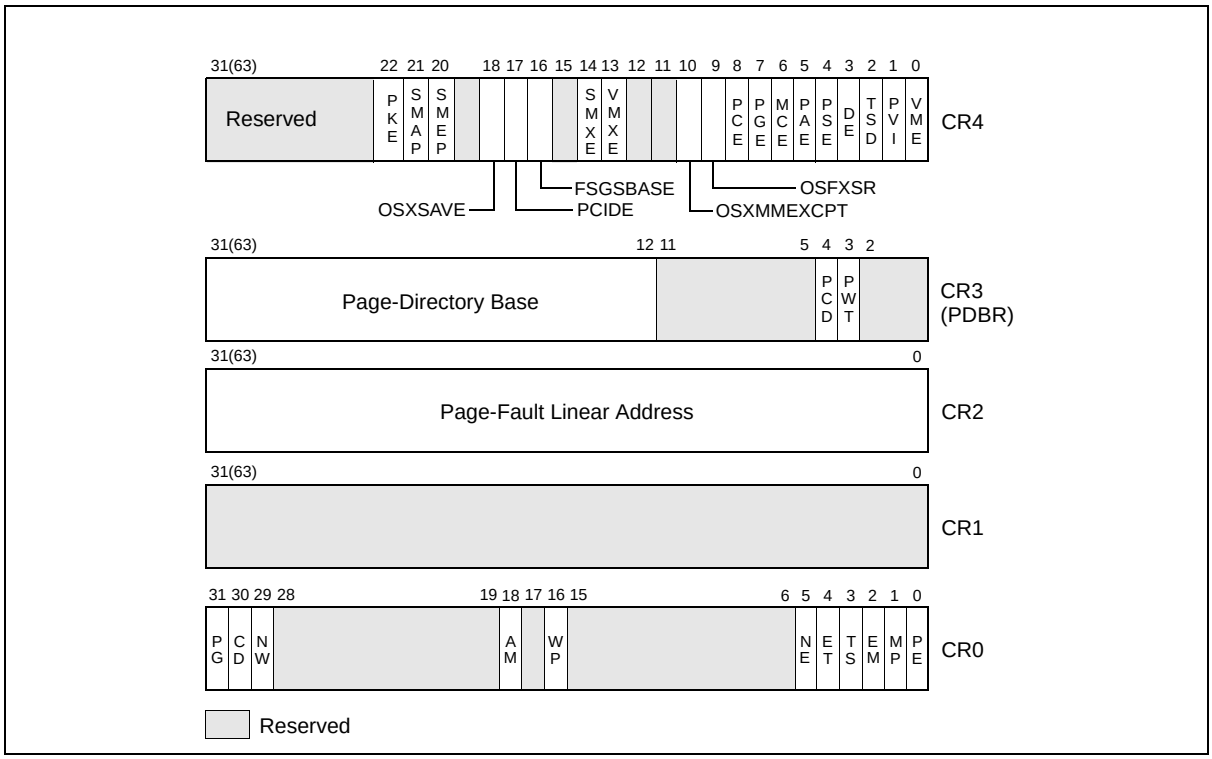


Figure 2-7. Control Registers

...

12. Updates to Chapter 4, Volume 3A

Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

4.7 PAGE-FAULT EXCEPTIONS

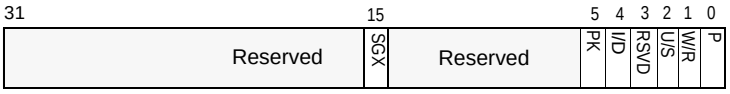
Accesses using linear addresses may cause **page-fault exceptions** (#PF; exception 14). An access to a linear address may cause page-fault exception for either of two reasons: (1) there is no translation for the linear address; or (2) there is a translation for the linear address, but its access rights do not permit the access.

As noted in Section 4.3, Section 4.4.2, and Section 4.5, there is no translation for a linear address if the translation process for that address would use a paging-structure entry in which the P flag (bit 0) is 0 or one that sets a reserved bit. If there is a translation for a linear address, its access rights are determined as specified in Section 4.6.

When Intel® Software Guard Extensions (Intel® SGX) are enabled, the processor may deliver exception 14 for reasons unrelated to paging. See Section 38.3, "Access-control Requirements" and Section 38.19, "Enclave Page Cache Map (EPCM)" in Chapter 38, "Enclave Access Control and Data Structures." Such an exception is called an

SGX-induced page fault. The processor uses the error code to distinguish SGX-induced page faults from ordinary page faults.

Figure 4-12 illustrates the error code that the processor provides on delivery of a page-fault exception. The following items explain how the bits in the error code describe the nature of the page-fault exception:



P
0 The fault was caused by a non-present page.
1 The fault was caused by a page-level protection violation.

W/R
0 The access causing the fault was a read.
1 The access causing the fault was a write.

U/S
0 A supervisor-mode access caused the fault.
1 A user-mode access caused the fault.

RSVD
0 The fault was not caused by reserved bit violation.
1 The fault was caused by a reserved bit set to 1 in some paging-structure entry.

I/D
0 The fault was not caused by an instruction fetch.
1 The fault was caused by an instruction fetch.

PK
0 The fault was not caused by protection keys.
1 There was a protection-key violation.

SGX
0 The fault is not related to SGX.
1 The fault resulted from violation of SGX-specific access-control requirements.

Figure 4-12. Page-Fault Error Code

- P flag (bit 0).
  This flag is 0 if there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.
- W/R (bit 1).
  If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- U/S (bit 2).
  If a user-mode access caused the page-fault exception, this flag is 1; it is 0 if a supervisor-mode access did so. This flag describes the access causing the page-fault exception, not the access rights specified by paging. User-mode and supervisor-mode accesses are defined in Section 4.6.
- RSVD flag (bit 3).
  This flag is 1 if there is no translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address. (Because reserved bits are not checked in a paging-structure entry whose P flag is 0, bit 3 of the error code can be set only if bit 0 is also set.[1])

1. Some past processors had errata for some page faults that occur when there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address. Due to these errata, some such page faults produced error codes that cleared bit 0 (P flag) and set bit 3 (RSVD flag).

Bits reserved in the paging-structure entries are reserved for future functionality. Software developers should be aware that such bits may be used in the future and that a paging-structure entry that causes a page-fault exception on one processor might not do so in the future.

- I/D flag (bit 4).
  This flag is 1 if (1) the access causing the page-fault exception was an instruction fetch; and (2) either (a) CR4.SMEP = 1; or (b) both (i) CR4.PAE = 1 (either PAE paging or IA-32e paging is in use); and (ii) IA32_EFER.NXE = 1. Otherwise, the flag is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- PK flag (bit 5).
  This flag is 1 if (1) IA32_EFER.LMA = CR4.PKE = 1; (2) the access causing the page-fault exception was a data access; (3) the linear address was a user-mode address with protection key *i*; and (5) the PKRU register (see Section 4.6.2) is such that either (a) AD*i* = 1; or (b) the following all hold: (i) WD*i* = 1; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode access.
- SGX flag (bit 15).
  This flag is 1 if the exception is unrelated to paging and resulted from violation of SGX-specific access-control requirements. Because such a violation can occur only if there is no ordinary page fault, this flag is set only if the P flag (bit 0) is 1 and the RSVD flag (bit 3) and the PK flag (bit 5) are both 0.

Page-fault exceptions occur only due to an attempt to use a linear address. Failures to load the PDPTE registers with PAE paging (see Section 4.4.1) cause general-protection exceptions (#GP(0)) and not page-fault exceptions.

...

13. 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 14—Page-Fault Exception (#PF)

Exception Class Fault.

Description

Indicates that, with paging enabled (the PG flag in the CR0 register is set), the processor detected one of the following conditions while using the page-translation mechanism to translate a linear address to a physical address:

- The P (present) flag in a page-directory or page-table entry needed for the address translation is clear, indicating that a page table or the page containing the operand is not present in physical memory.
- The procedure does not have sufficient privilege to access the indicated page (that is, a procedure running in user mode attempts to access a supervisor-mode page). If the SMAP flag is set in CR4, a page fault may also be triggered by code running in supervisor mode that tries to access data at a user-mode address. If the PKE flag is set in CR4, the PKRU register may cause page faults on data accesses to user-mode addresses with certain protection keys.
- Code running in user mode attempts to write to a read-only page. If the WP flag is set in CR0, the page fault will also be triggered by code running in supervisor mode that tries to write to a read-only page.
- An instruction fetch to a linear address that translates to a physical address in a memory page with the execute-disable bit set (for information about the execute-disable bit, see Chapter 4, "Paging"). If the SMEP

flag is set in CR4, a page fault will also be triggered by code running in supervisor mode that tries to fetch an instruction from a user-mode address.

- One or more reserved bits in paging-structure entry are set to 1. See description below of RSVD error code flag.
- An enclave access violates one of the specified access-control requirements. See Section 38.3, "Access-control Requirements" and Section 38.19, "Enclave Page Cache Map (EPCM)" in Chapter 38, "Enclave Access Control and Data Structures." In this case, the exception is called an **SGX-induced page fault**. The processor uses the error code (below) to distinguish SGX-induced page faults from ordinary page faults.

The exception handler can recover from page-not-present conditions and restart the program or task without any loss of program continuity. It can also restart the program or task after a privilege violation, but the problem that caused the privilege violation may be uncorrectable.

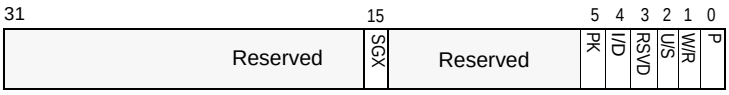
See also: Section 4.7, "Page-Fault Exceptions."

Exception Error Code

Yes (special format). The processor provides the page-fault handler with two items of information to aid in diagnosing the exception and recovering from it:

- An error code on the stack. The error code for a page fault has a format different from that for other exceptions (see Figure 6-9). The processor establishes the bits in the error code as follows:
  - — P flag (bit 0).
    This flag is 0 if there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.
  - — W/R (bit 1).
    If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
  - — U/S (bit 2).
    If a user-mode access caused the page-fault exception, this flag is 1; it is 0 if a supervisor-mode access did so. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
  - — RSVD flag (bit 3).
    This flag is 1 if there is no translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address.
  - — I/D flag (bit 4).
    This flag is 1 if the access causing the page-fault exception was an instruction fetch. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
  - — PK flag (bit 5).
    This flag is 1 if the access causing the page-fault exception was a data access to a user-mode address with protection key disallowed by the value of the PKRU register.
  - — SGX flag (bit 15).
    This flag is 1 if the exception is unrelated to paging and resulted from violation of SGX-specific access-control requirements. Because such a violation can occur only if there is no ordinary page fault, this flag is set only if the P flag (bit 0) is 1 and the RSVD flag (bit 3) and the PK flag (bit 5) are both 0.

  See Section 4.6, "Access Rights" and Section 4.7, "Page-Fault Exceptions" for more information about page-fault exceptions and the error codes that they produce.



P
0 The fault was caused by a non-present page.
1 The fault was caused by a page-level protection violation.

W/R
0 The access causing the fault was a read.
1 The access causing the fault was a write.

U/S
0 A supervisor-mode access caused the fault.
1 A user-mode access caused the fault.

RSVD
0 The fault was not caused by reserved bit violation.
1 The fault was caused by a reserved bit set to 1 in some paging-structure entry.

I/D
0 The fault was not caused by an instruction fetch.
1 The fault was caused by an instruction fetch.

PK
0 The fault was not caused by protection keys.
1 There was a protection-key violation.

SGX
0 The fault is not related to SGX.
1 The fault resulted from violation of SGX-specific access-control requirements.

Figure 6-9. Page-Fault Error Code

- The contents of the CR2 register. The processor loads the CR2 register with the 32-bit linear address that generated the exception. The page-fault handler can use this address to locate the corresponding page directory and page-table entries. Another page fault can potentially occur during execution of the page-fault handler; the handler should save the contents of the CR2 register before a second page fault can occur.[1] If a page fault is caused by a page-level protection violation, the access flag in the page-directory entry is set when the fault occurs. The behavior of IA-32 processors regarding the access flag in the corresponding page-table entry is model specific and not architecturally defined.

Saved Instruction Pointer

The saved contents of CS and EIP registers generally point to the instruction that generated the exception. If the page-fault exception occurred during a task switch, the CS and EIP registers may point to the first instruction of the new task (as described in the following “Program State Change” section).

Program State Change

A program-state change does not normally accompany a page-fault exception, because the instruction that causes the exception to be generated is not executed. After the page-fault exception handler has corrected the violation (for example, loaded the missing page into memory), execution of the program or task can be resumed.

When a page-fault exception is generated during a task switch, the program-state may change, as follows. During a task switch, a page-fault exception can occur during any of following operations:

1. Processors update CR2 whenever a page fault is detected. If a second page fault occurs while an earlier page fault is being delivered, the faulting linear address of the second fault will overwrite the contents of CR2 (replacing the previous address). These updates to CR2 occur even if the page fault results in a double fault or occurs during the delivery of a double fault.

- While writing the state of the original task into the TSS of that task.
- While reading the GDT to locate the TSS descriptor of the new task.
- While reading the TSS of the new task.
- While reading segment descriptors associated with segment selectors from the new task.
- While reading the LDT of the new task to verify the segment registers stored in the new TSS.

In the last two cases the exception occurs in the context of the new task. The instruction pointer refers to the first instruction of the new task, not to the instruction which caused the task switch (or the last instruction to be executed, in the case of an interrupt). If the design of the operating system permits page faults to occur during task-switches, the page-fault handler should be called through a task gate.

If a page fault occurs during a task switch, 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 page-fault 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.)

Additional Exception-Handling Information

Special care should be taken to ensure that an exception that occurs during an explicit stack switch does not cause the processor to use an invalid stack pointer (SS:ESP). Software written for 16-bit IA-32 processors often use a pair of instructions to change to a new stack, for example:

```
MOV SS, AX
MOV SP, StackTop
```

When executing this code on one of the 32-bit IA-32 processors, it is possible to get a page fault, general-protection fault (#GP), or alignment check fault (#AC) after the segment selector has been loaded into the SS register but before the ESP register has been loaded. At this point, the two parts of the stack pointer (SS and ESP) are inconsistent. The new stack segment is being used with the old stack pointer.

The processor does not use the inconsistent stack pointer if the exception handler switches to a well defined stack (that is, the handler is a task or a more privileged procedure). However, if the exception handler is called at the same privilege level and from the same task, the processor will attempt to use the inconsistent stack pointer.

In systems that handle page-fault, general-protection, or alignment check exceptions within the faulting task (with trap or interrupt gates), software executing at the same privilege level as the exception handler should initialize a new stack by using the LSS instruction rather than a pair of MOV instructions, as described earlier in this note. When the exception handler is running at privilege level 0 (the normal case), the problem is limited to procedures or tasks that run at privilege level 0, typically the kernel of the operating system.

...

14. Updates to Chapter 8, Volume 3A

Change bars show changes to Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

8.2.2 Memory Ordering in P6 and More Recent Processor Families

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

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

- Reads are not reordered with other reads.
- Writes are not reordered with older reads.
- Writes to memory are not reordered with other writes, with the following exceptions:
  - — streaming stores (writes) executed with the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD); and
  - — string operations (see Section 8.2.4.1).
- No write to memory may be reordered with an execution of the CLFLUSH instruction; a write may be reordered with an execution of the CLFLUSHOPT instruction that flushes a cache line other than the one being written.[1] Executions of the CLFLUSH instruction are not reordered with each other. Executions of CLFLUSHOPT that access different cache lines may be reordered with each other. An execution of CLFLUSHOPT may be reordered with an execution of CLFLUSH that accesses a different cache line.
- Reads may be reordered with older writes to different locations but not with older writes to the same location.
- Reads or writes cannot be reordered with I/O instructions, locked instructions, or serializing instructions.
- Reads cannot pass earlier LFENCE and MFENCE instructions.
- Writes and executions of CLFLUSH and CLFLUSHOPT cannot pass earlier LFENCE, SFENCE, and MFENCE instructions.
- LFENCE instructions cannot pass earlier reads.
- SFENCE instructions cannot pass earlier writes or executions of CLFLUSH and CLFLUSHOPT.
- MFENCE instructions cannot pass earlier reads, writes, or executions of CLFLUSH and CLFLUSHOPT.

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

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

See the example in Figure 8-1. Consider three processors in a system and each processor performs three writes, one to each of three defined locations (A, B, and C). Individually, the processors perform the writes in the same program order, but because of bus arbitration and other memory access mechanisms, the order that the three processors write the individual memory locations can differ each time the respective code sequences are executed on the processors. The final values in location A, B, and C would possibly vary on each execution of the write sequence.

...

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

8.7.13.1 Processor Caches

For processors supporting Intel Hyper-Threading Technology, the caches are shared. Any cache manipulation instruction that is executed on one logical processor has a global effect on the cache hierarchy of the physical processor. Note the following:

- **WBINVD instruction** — The entire cache hierarchy is invalidated after modified data is written back to memory. All logical processors are stopped from executing until after the write-back and invalidate operation is completed. A special bus cycle is sent to all caching agents. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.
- **INVD instruction** — The entire cache hierarchy is invalidated without writing back modified data to memory. All logical processors are stopped from executing until after the invalidate operation is completed. A special bus cycle is sent to all caching agents.
- **CLFLUSH and CLFLUSHOPT instructions** — The specified cache line is invalidated from the cache hierarchy after any modified data is written back to memory and a bus cycle is sent to all caching agents, regardless of which logical processor caused the cache line to be filled.
- **CD flag in control register CR0** — Each logical processor has its own CR0 control register, and thus its own CD flag in CR0. The CD flags for the two logical processors are ORed together, such that when any logical processor sets its CD flag, the entire cache is nominally disabled.

...

15. Updates to Chapter 11, Volume 3A

Change bars show changes to Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

11.1 INTERNAL CACHES, TLBS, AND BUFFERS

The Intel 64 and IA-32 architectures support cache, translation look aside buffers (TLBs), and a store buffer for temporary on-chip (and external) storage of instructions and data. (Figure 11-1 shows the arrangement of caches, TLBs, and the store buffer for the Pentium 4 and Intel Xeon processors.) Table 11-1 shows the characteristics of these caches and buffers for the Pentium 4, Intel Xeon, P6 family, and Pentium processors. **The sizes and characteristics of these units are machine specific and may change in future versions of the processor.** The CPUID instruction returns the sizes and characteristics of the caches and buffers for the processor on which the instruction is executed. See "CPUID—CPU Identification" in Chapter 3, "Instruction Set Reference, A-M," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

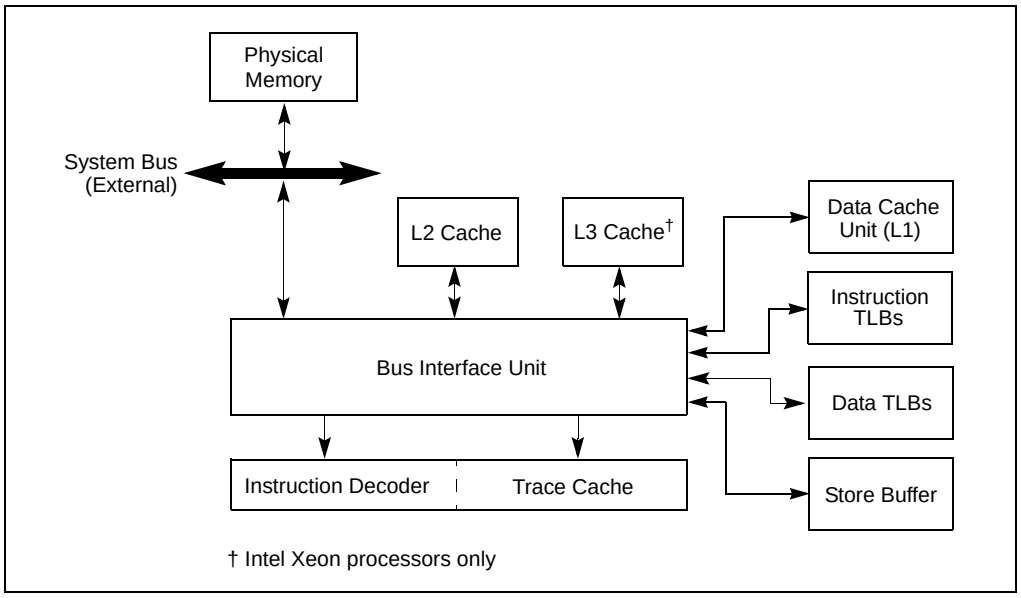


Figure 11-1. Cache Structure of the Pentium 4 and Intel Xeon Processors

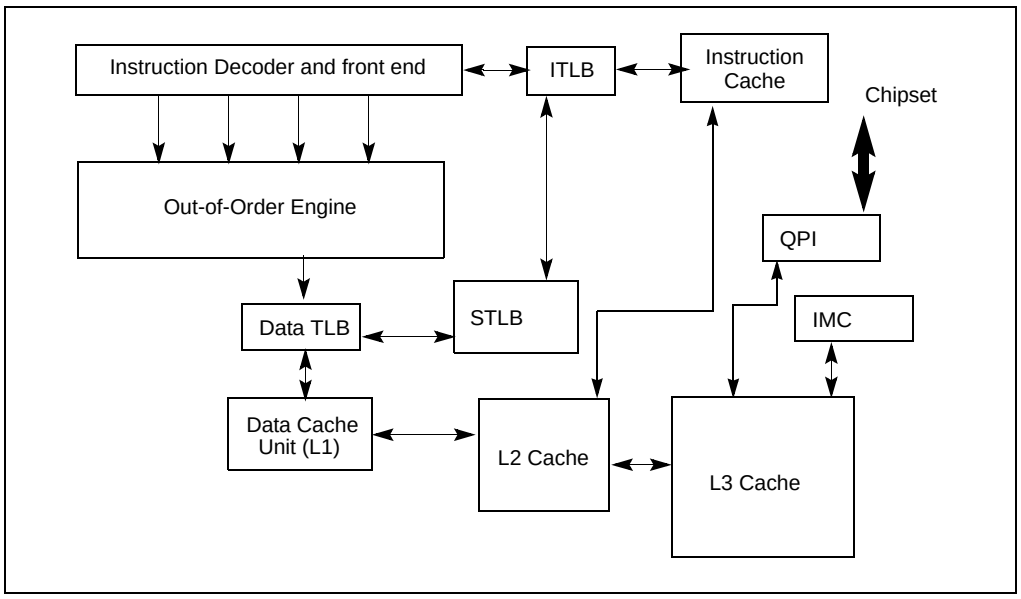


Figure 11-2. Cache Structure of the Intel Core i7 Processors

Figure 11-2 shows the cache arrangement of Intel Core i7 processor.

Table 11-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors

Cache or Buffer	Characteristics
Trace Cache[1]	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst® microarchitecture): 12 Kμops, 8-way set associative. ▪ Intel Core i7, Intel Core 2 Duo, Intel® Atom™, Intel Core Duo, Intel Core Solo, Pentium M processor: not implemented. ▪ P6 family and Pentium processors: not implemented.
L1 Instruction Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): not implemented. ▪ Intel Core i7 processor: 32-KByte, 4-way set associative. ▪ Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M processor: 32-KByte, 8-way set associative. ▪ P6 family and Pentium processors: 8- or 16-KByte, 4-way set associative, 32-byte cache line size; 2-way set associative for earlier Pentium processors.
L1 Data Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 8-KByte, 4-way set associative, 64-byte cache line size. ▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 16-KByte, 8-way set associative, 64-byte cache line size. ▪ Intel Atom processors: 24-KByte, 6-way set associative, 64-byte cache line size. ▪ Intel Core i7, Intel Core 2 Duo, Intel Core Duo, Intel Core Solo, Pentium M and Intel Xeon processors: 32-KByte, 8-way set associative, 64-byte cache line size. ▪ P6 family processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KBytes, 2-way set associative for earlier P6 family processors. ▪ Pentium processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KByte, 2-way set associative for earlier Pentium processors.
L2 Unified Cache	▪ Intel Core 2 Duo and Intel Xeon processors: up to 4-MByte (or 4MBx2 in quadcore processors), 16-way set associative, 64-byte cache line size. ▪ Intel Core 2 Duo and Intel Xeon processors: up to 6-MByte (or 6MBx2 in quadcore processors), 24-way set associative, 64-byte cache line size. ▪ Intel Core i7, i5, i3 processors: 256KBbyte, 8-way set associative, 64-byte cache line size. ▪ Intel Atom processors: 512-KByte, 8-way set associative, 64-byte cache line size. ▪ Intel Core Duo, Intel Core Solo processors: 2-MByte, 8-way set associative, 64-byte cache line size ▪ Pentium 4 and Intel Xeon processors: 256, 512, 1024, or 2048-KByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. ▪ Pentium M processor: 1 or 2-MByte, 8-way set associative, 64-byte cache line size. ▪ P6 family processors: 128-KByte, 256-KByte, 512-KByte, 1-MByte, or 2-MByte, 4-way set associative, 32-byte cache line size. ▪ Pentium processor (external optional): System specific, typically 256- or 512-KByte, 4-way set associative, 32-byte cache line size.
L3 Unified Cache	▪ Intel Xeon processors: 512-KByte, 1-MByte, 2-MByte, or 4-MByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. ▪ Intel Core i7 processor, Intel Xeon processor 5500: Up to 8MByte, 16-way set associative, 64-byte cache line size. ▪ Intel Xeon processor 5600: Up to 12MByte, 64-byte cache line size. ▪ Intel Xeon processor 7500: Up to 24MByte, 64-byte cache line size.
Instruction TLB (4-KByte Pages)	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 128 entries, 4-way set associative. ▪ Intel Atom processors: 32-entries, fully associative. ▪ Intel Core i7, i5, i3 processors: 64-entries per thread (128-entries per core), 4-way set associative. ▪ Intel Core 2 Duo, Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. ▪ P6 family processors: 32 entries, 4-way set associative. ▪ Pentium processor: 32 entries, 4-way set associative; fully set associative for Pentium processors with MMX technology.

Table 11-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors (Contd.)

Cache or Buffer	Characteristics
Data TLB (4-KByte Pages)	▪ Intel Core i7, i5, i3 processors, DTLB0: 64-entries, 4-way set associative. ▪ Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 256 entries, 4 ways. ▪ Intel Atom processors: 16-entry-per-thread micro-TLB, fully associative; 64-entry DTLB, 4-way set associative; 16-entry PDE cache, fully associative. ▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 64 entry, fully set associative, shared with large page DTLB. ▪ Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. ▪ Pentium and P6 family processors: 64 entries, 4-way set associative; fully set, associative for Pentium processors with MMX technology.
Instruction TLB (Large Pages)	▪ Intel Core i7, i5, i3 processors: 7-entries per thread, fully associative. ▪ Intel Core 2 Duo processors: 4 entries, 4 ways. ▪ Pentium 4 and Intel Xeon processors: large pages are fragmented. ▪ Intel Core Duo, Intel Core Solo, Pentium M processor: 2 entries, fully associative. ▪ P6 family processors: 2 entries, fully associative. ▪ Pentium processor: Uses same TLB as used for 4-KByte pages.
Data TLB (Large Pages)	▪ Intel Core i7, i5, i3 processors, DTLB0: 32-entries, 4-way set associative. ▪ Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 32 entries, 4 ways. ▪ Intel Atom processors: 8 entries, 4-way set associative. ▪ Pentium 4 and Intel Xeon processors: 64 entries, fully set associative; shared with small page data TLBs. ▪ Intel Core Duo, Intel Core Solo, Pentium M processor: 8 entries, fully associative. ▪ P6 family processors: 8 entries, 4-way set associative. ▪ Pentium processor: 8 entries, 4-way set associative; uses same TLB as used for 4-KByte pages in Pentium processors with MMX technology.
Second-level Unified TLB (4-KByte Pages)	▪ Intel Core i7, i5, i3 processor, STLB: 512-entries, 4-way set associative.
Store Buffer	▪ Intel Core i7, i5, i3 processors: 32entries. ▪ Intel Core 2 Duo processors: 20 entries. ▪ Intel Atom processors: 8 entries, used for both WC and store buffers. ▪ Pentium 4 and Intel Xeon processors: 24 entries. ▪ Pentium M processor: 16 entries. ▪ P6 family processors: 12 entries. ▪ Pentium processor: 2 buffers, 1 entry each (Pentium processors with MMX technology have 4 buffers for 4 entries).
Write Combining (WC) Buffer	▪ Intel Core 2 Duo processors: 8 entries. ▪ Intel Atom processors: 8 entries, used for both WC and store buffers. ▪ Pentium 4 and Intel Xeon processors: 6 or 8 entries. ▪ Intel Core Duo, Intel Core Solo, Pentium M processors: 6 entries. ▪ P6 family processors: 4 entries.

NOTES:

1 Introduced to the IA-32 architecture in the Pentium 4 and Intel Xeon processors.

Intel 64 and IA-32 processors may implement four types of caches: the trace cache, the level 1 (L1) cache, the level 2 (L2) cache, and the level 3 (L3) cache. See Figure 11-1. Cache availability is described below:

- **Intel Core i7, i5, i3 processor Family and Intel Xeon processor Family based on Intel® microarchitecture code name Nehalem and Intel® microarchitecture code name Westmere** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache. Each processor core has its own L1 and L2. The L3 cache is an inclusive, unified data and instruction cache, shared by all processor cores inside a physical package. No trace cache is implemented.

- **Intel® Core™ 2 processor family and Intel® Xeon® processor family based on Intel® Core™ microarchitecture** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip; it is shared between two processor cores in a dual-core processor implementation. Quad-core processors have two L2, each shared by two processor cores. No trace cache is implemented.
- **Intel® Atom™ processor** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache is located on the processor chip. No trace cache is implemented.
- **Intel® Core™ Solo and Intel® Core™ Duo processors** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip. It is shared between two processor cores in a dual-core processor implementation. No trace cache is implemented.
- **Pentium® 4 and Intel® Xeon® processors Based on Intel NetBurst® microarchitecture** — The trace cache caches decoded instructions (μops) from the instruction decoder and the L1 cache contains data. The L2 and L3 caches are unified data and instruction caches located on the processor chip. Dualcore processors have two L2, one in each processor core. Note that the L3 cache is only implemented on some Intel Xeon processors.
- **P6 family processors** — The L1 cache is divided into two sections: one dedicated to caching instructions (pre-decoded instructions) and the other to caching data. The L2 cache is a unified data and instruction cache located on the processor chip. P6 family processors do not implement a trace cache.
- **Pentium® processors** — The L1 cache has the same structure as on P6 family processors. There is no trace cache. The L2 cache is a unified data and instruction cache external to the processor chip on earlier Pentium processors and implemented on the processor chip in later Pentium processors. For Pentium processors where the L2 cache is external to the processor, access to the cache is through the system bus.

For Intel Core i7 processors and processors based on Intel Core, Intel Atom, and Intel NetBurst microarchitectures, Intel Core Duo, Intel Core Solo and Pentium M processors, the cache lines for the L1 and L2 caches (and L3 caches if supported) are 64 bytes wide. The processor always reads a cache line from system memory beginning on a 64-byte boundary. (A 64-byte aligned cache line begins at an address with its 6 least-significant bits clear.) A cache line can be filled from memory with a 8-transfer burst transaction. The caches do not support partially-filled cache lines, so caching even a single doubleword requires caching an entire line.

The L1 and L2 cache lines in the P6 family and Pentium processors are 32 bytes wide, with cache line reads from system memory beginning on a 32-byte boundary (5 least-significant bits of a memory address clear.) A cache line can be filled from memory with a 4-transfer burst transaction. Partially-filled cache lines are not supported.

The trace cache in processors based on Intel NetBurst microarchitecture is available in all execution modes: protected mode, system management mode (SMM), and real-address mode. The L1,L2, and L3 caches are also available in all execution modes; however, use of them must be handled carefully in SMM (see Section 34.4.2, "SMRAM Caching").

The TLBs store the most recently used page-directory and page-table entries. They speed up memory accesses when paging is enabled by reducing the number of memory accesses that are required to read the page tables stored in system memory. The TLBs are divided into four groups: instruction TLBs for 4-KByte pages, data TLBs for 4-KByte pages; instruction TLBs for large pages (2-MByte, 4-MByte or 1-GByte pages), and data TLBs for large pages. The TLBs are normally active only in protected mode with paging enabled. When paging is disabled or the processor is in real-address mode, the TLBs maintain their contents until explicitly or implicitly flushed (see Section 11.9, "Invalidating the Translation Lookaside Buffers (TLBs)").

Processors based on Intel Core microarchitectures implement one level of instruction TLB and two levels of data TLB. Intel Core i7 processor provides a second-level unified TLB.

The store buffer is associated with the processors instruction execution units. It allows writes to system memory and/or the internal caches to be saved and in some cases combined to optimize the processor's bus accesses. The store buffer is always enabled in all execution modes.

The processor's caches are for the most part transparent to software. When enabled, instructions and data flow through these caches without the need for explicit software control. However, knowledge of the behavior of these caches may be useful in optimizing software performance. For example, knowledge of cache dimensions and replacement algorithms gives an indication of how large of a data structure can be operated on at once without causing cache thrashing.

In multiprocessor systems, maintenance of cache consistency may, in rare circumstances, require intervention by system software. For these rare cases, the processor provides privileged cache control instructions for use in flushing caches and forcing memory ordering.

There are several instructions that software can use to improve the performance of the L1, L2, and L3 caches, including the PREFETCHh, CLFLUSH, and CLFLUSHOPT instructions and the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD). The use of these instructions are discussed in Section 11.5.5, "Cache Management Instructions."

...

11.5.5 Cache Management Instructions

The Intel 64 and IA-32 architectures provide several instructions for managing the L1, L2, and L3 caches. The INVD and WBINVD instructions are privileged instructions and operate on the L1, L2 and L3 caches as a whole. The PREFETCHh, CLFLUSH and CLFLUSHOPT instructions and the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD) offer more granular control over caching, and are available to all privileged levels.

The INVD and WBINVD instructions are used to invalidate the contents of the L1, L2, and L3 caches. The INVD instruction invalidates all internal cache entries, then generates a special-function bus cycle that indicates that external caches also should be invalidated. The INVD instruction should be used with care. It does not force a write-back of modified cache lines; therefore, data stored in the caches and not written back to system memory will be lost. Unless there is a specific requirement or benefit to invalidating the caches without writing back the modified lines (such as, during testing or fault recovery where cache coherency with main memory is not a concern), software should use the WBINVD instruction.

The WBINVD instruction first writes back any modified lines in all the internal caches, then invalidates the contents of both the L1, L2, and L3 caches. It ensures that cache coherency with main memory is maintained regardless of the write policy in effect (that is, write-through or write-back). Following this operation, the WBINVD instruction generates one (P6 family processors) or two (Pentium and Intel486 processors) special-function bus cycles to indicate to external cache controllers that write-back of modified data followed by invalidation of external caches should occur. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.

The PREFETCH*h* instructions allow a program to suggest to the processor that a cache line from a specified location in system memory be prefetched into the cache hierarchy (see Section 11.8, "Explicit Caching").

The CLFLUSH and CLFLUSHOPT instructions allow selected cache lines to be flushed from memory. These instructions give a program the ability to explicitly free up cache space, when it is known that cached section of system memory will not be accessed in the near future.

The non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD) allow data to be moved from the processor's registers directly into system memory without being also written into the L1, L2, and/or L3 caches. These instructions can be used to prevent cache pollution when operating on data that is going to be modified only once before being stored back into system memory. These instructions operate on data in the general-purpose, MMX, and XMM registers.

...

16. Updates to Chapter 15, Volume 3B

Change bars show changes to Chapter 15 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

15.3.2.1 IA32_MCi_CTL MSRs

The IA32_MC*i*_CTL MSR controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units). Each of the 64 flags (EE*j*) represents a potential error. Setting an EE*j* flag enables signaling #MC of the associated error and clearing it disables signaling of the error. Error logging happens regardless of the setting of these bits. The processor drops writes to bits that are not implemented. Figure 15-5 shows the bit fields of IA32_MC*i*_CTL.

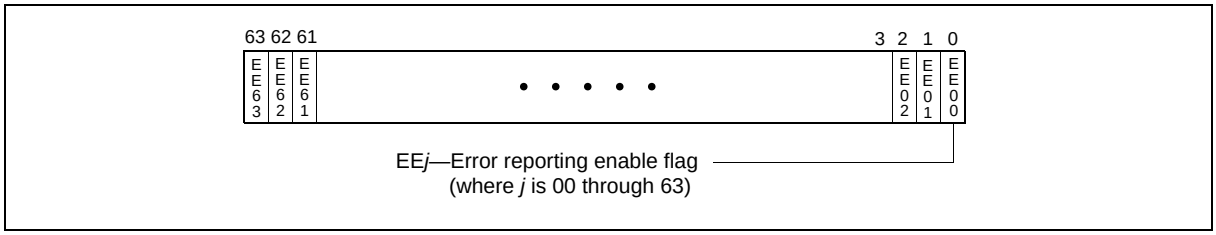


Figure 15-5. IA32_MC*i*_CTL Register

NOTE

For P6 family processors, processors based on Intel Core microarchitecture (excluding those on which on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH and onward): the operating system or executive software must not modify the contents of the IA32_MC0_CTL MSR. This MSR is internally aliased to the EBL_CR_POWERON MSR and controls platform-specific error handling features. System specific firmware (the BIOS) is responsible for the appropriate initialization of the IA32_MC0_CTL MSR. P6 family processors only allow the writing of all 1s or all 0s to the IA32_MC*i*_CTL MSR.

15.3.2.2 IA32_MCi_STATUS MSRS

Each IA32_MCi_STATUS MSR contains information related to a machine-check error if its VAL (valid) flag is set (see Figure 15-6). Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.

NOTE

Figure 15-6 depicts the IA32_MCi_STATUS MSR when IA32_MCG_CAP[24] = 1, IA32_MCG_CAP[11] = 1 and IA32_MCG_CAP[10] = 1. When IA32_MCG_CAP[24] = 0 and IA32_MCG_CAP[11] = 1, bits 56:55 is reserved and bits 54:53 for threshold-based error reporting. When IA32_MCG_CAP[11] = 0, bits 56:53 are part of the "Other Information" field. The use of bits 54:53 for threshold-based error reporting began with Intel Core Duo processors, and is currently used for cache memory. See Section 15.4, "Enhanced Cache Error reporting," for more information. When IA32_MCG_CAP[10] = 0, bits 52:38 are part of the "Other Information" field. The use of bits 52:38 for corrected MC error count is introduced with Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH.

Where:

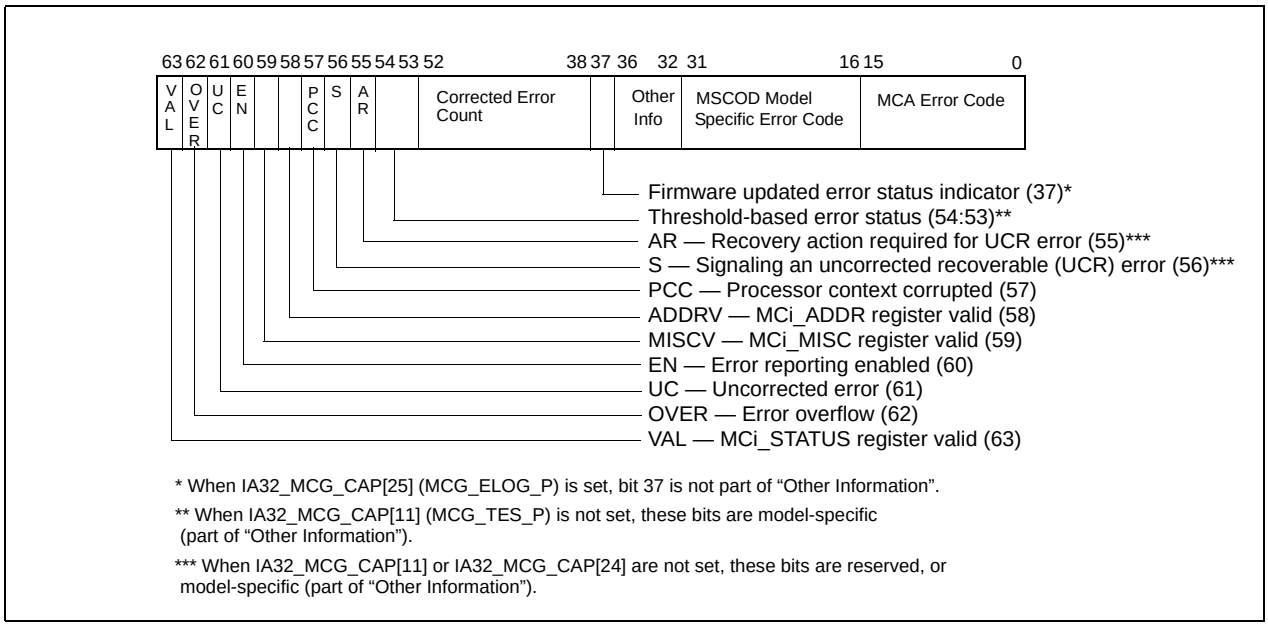


Figure 15-6. IA32_MC*i*_STATUS Register

- **MCA (machine-check architecture) error code field, bits 15:0** — Specifies the machine-check architecture-defined error code for the machine-check error condition detected. The machine-check architecture-defined error codes are guaranteed to be the same for all IA-32 processors that implement the machine-check architecture. See Section 15.9, "Interpreting the MCA Error Codes," and Chapter 16, "Interpreting Machine-Check Error Codes", for information on machine-check error codes.
- **Model-specific error code field, bits 31:16** — Specifies the model-specific error code that uniquely identifies the machine-check error condition detected. The model-specific error codes may differ among IA-32 processors for the same machine-check error condition. See Chapter 16, "Interpreting Machine-Check Error Codes" for information on model-specific error codes.
- **Reserved, Error Status, and Other Information fields, bits 56:32** —
    - If IA32_MCG_CAP.MCG_ELOG_P[bit 25] is 0, bits 37:32 contain "Other Information" that is implementation-specific and is not part of the machine-check architecture.
    - If IA32_MCG_CAP.MCG_ELOG_P is 1, "Other Information" is in bits 36:32. If bit 37 is 0, system firmware has not changed the contents of IA32_MCi_STATUS. If bit 37 is 1, system firmware may have edited the contents of IA32_MCi_STATUS.
    - If IA32_MCG_CAP.MCG_CMCI_P[bit 10] is 0, bits 52:38 also contain "Other Information" (in the same sense as bits 37:32).
    - If IA32_MCG_CAP[10] is 1, bits 52:38 are architectural (not model-specific). In this case, bits 52:38 reports the value of a 15 bit counter that increments each time a corrected error is observed by the MCA recording bank. This count value will continue to increment until cleared by software. The most significant bit, 52, is a sticky count overflow bit.
    - If IA32_MCG_CAP[11] is 0, bits 56:53 also contain "Other Information" (in the same sense).
    - If IA32_MCG_CAP[11] is 1, bits 56:53 are architectural (not model-specific). In this case, bits 56:53 have the following functionality:
        - If IA32_MCG_CAP[24] is 0, bits 56:55 are reserved.
        - If IA32_MCG_CAP[24] is 1, bits 56:55 are defined as follows:

- S (Signaling) flag, bit 56 - Signals the reporting of UCR errors in this MC bank. See Section 15.6.2 for additional detail.
- AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. See Section 15.6.2 for additional detail.
- If the UC bit (Figure 15-6) is 1, bits 54:53 are undefined.
- If the UC bit (Figure 15-6) is 0, bits 54:53 indicate the status of the hardware structure that reported the threshold-based error. See Table 15-1.

Table 15-1. Bits 54:53 in IA32_MCi_STATUS MSRs when IA32_MCG_CAP[11] = 1 and UC = 0

Bits 54:53	Meaning
00	**No tracking** - No hardware status tracking is provided for the structure reporting this event.
01	**Green** - Status tracking is provided for the structure posting the event; the current status is green (below threshold). For more information, see Section 15.4, "Enhanced Cache Error reporting".
10	**Yellow** - Status tracking is provided for the structure posting the event; the current status is yellow (above threshold). For more information, see Section 15.4, "Enhanced Cache Error reporting".
11	Reserved

- **PCC (processor context corrupt) flag, bit 57** — Indicates (when set) that the state of the processor might have been corrupted by the error condition detected and that reliable restarting of the processor may not be possible. When clear, this flag indicates that the error did not affect the processor's state, and software may be able to restart. When system software supports recovery, consult Section 15.10.4 for additional rules that apply.
- **ADDRV (IA32_MC*i*_ADDR register valid) flag, bit 58** — Indicates (when set) that the IA32_MCi_ADDR register contains the address where the error occurred (see Section 15.3.2.3, "IA32_MCi_ADDR MSRs"). When clear, this flag indicates that the IA32_MCi_ADDR register is either not implemented or does not contain the address where the error occurred. Do not read these registers if they are not implemented in the processor.
- **MISCV (IA32_MC*i*_MISC register valid) flag, bit 59** — Indicates (when set) that the IA32_MC*i*_MISC register contains additional information regarding the error. When clear, this flag indicates that the IA32_MC*i*_MISC register is either not implemented or does not contain additional information regarding the error. Do not read these registers if they are not implemented in the processor.
- **EN (error enabled) flag, bit 60** — Indicates (when set) that the error was enabled by the associated EEj bit of the IA32_MC*i*_CTL register.
- **UC (error uncorrected) flag, bit 61** — Indicates (when set) that the processor did not or was not able to correct the error condition. When clear, this flag indicates that the processor was able to correct the error condition.
- **OVER (machine check overflow) flag, bit 62** — Indicates (when set) that a machine-check error occurred while the results of a previous error were still in the error-reporting register bank (that is, the VAL bit was already set in the IA32_MC*i*_STATUS register). The processor sets the OVER flag and software is responsible for clearing it. In general, enabled errors are written over disabled errors, and uncorrected errors are written over corrected errors. Uncorrected errors are not written over previous valid uncorrected errors. For more information, see Section 15.3.2.2.1, "Overwrite Rules for Machine Check Overflow".
- **VAL (IA32_MC*i*_STATUS register valid) flag, bit 63** — Indicates (when set) that the information within the IA32_MCi_STATUS register is valid. When this flag is set, the processor follows the rules given for the OVER flag in the IA32_MCi_STATUS register when overwriting previously valid entries. The processor sets the VAL flag and software is responsible for clearing it.

...

15.5.2.3 CMCI Interrupt Handler

The following describes techniques system software may consider to implement a CMCI service routine:

- The service routine examines its private per-thread data structure to check which set of MC banks it has ownership. If the thread does not have ownership of a given MC bank, proceed to the next MC bank. Ownership is determined at initialization time which is described in Section [Cross Reference to 14.5.2.1].

If the thread had claimed ownership to an MC bank, this technique will allow each logical processors to handle corrected MC errors independently and requires no synchronization to access shared MSR resources. Consult Example 15-5 for guidelines on logging when processing CMCI.

...

15.6.3 UCR Error Classification

With the S and AR flag encoding in the IA32_MCi_STATUS register, UCR errors can be classified as:

- Uncorrected no action required (UCNA) - is a UCR error that is not signaled via a machine check exception and, instead, is reported to system software as a corrected machine check error. UCNA errors indicate that some data in the system is corrupted, but the data has not been consumed and the processor state is valid and you may continue execution on this processor. UCNA errors require no action from system software to continue execution. A UNCA error is indicated with UC=1, PCC=0, S=0 and AR=0 in the IA32_MCi_STATUS register.
- Software recoverable action optional (SRAO) - a UCR error is signaled via a machine check exception and a system software recovery action is optional and not required to continue execution from this machine check exception. SRAO errors indicate that some data in the system is corrupt, but the data has not been consumed and the processor state is valid. SRAO errors provide the additional error information for system software to perform a recovery action. An SRAO error is indicated with UC=1, PCC=0, S=1, EN=1 and AR=0 in the IA32_MCi_STATUS register. Recovery actions for SRAO errors are MCA error code specific. The MISCV and the ADDRV flags in the IA32_MCi_STATUS register are set when the additional error information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MCi_STATUS register to identify the specific recovery action for a given SRAO error. If MISCV and ADDRV are not set, it is recommended that no system software error recovery be performed however, you can resume execution.
- Software recoverable action required (SRAR) - a UCR error that requires system software to take a recovery action on this processor before scheduling another stream of execution on this processor. SRAR errors indicate that the error was detected and raised at the point of the consumption in the execution flow. An SRAR error is indicated with UC=1, PCC=0, S=1, EN=1 and AR=1 in the IA32_MCi_STATUS register. Recovery actions are MCA error code specific. The MISCV and the ADDRV flags in the IA32_MCi_STATUS register are set when the additional error information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MCi_STATUS register to identify the specific recovery action for a given SRAR error. If MISCV and ADDRV are not set, it is recommended that system software shutdown the system.

Table 15-6 summarizes UCR, corrected, and uncorrected errors.

Table 15-6. MC Error Classifications

Type of Error[1]	UC	EN	PCC	S	AR	Signaling	Software Action	Example
Uncorrected Error (UC)	1	1	1	x	x	MCE	If EN=1, reset the system, else log and OK to keep the system running.	
SRAR	1	1	0	1	1	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, must bugcheck. If OVER=1, reset system, else take specific recovery action.	Cache to processor load error.
SRAO	1	1	0	1	0	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, OK to keep the system running.	Patrol scrub and explicit writeback poison errors.
UCNA	1	x	0	0	0	CMC	Log the error and Ok to keep the system running.	Poison detection error.
Corrected Error (CE)	0	x	x	x	x	CMC	Log the error and no corrective action required.	ECC in caches and memory.

NOTES:
1. SRAR, SRAO and UCNA errors are supported by the processor only when IA32_MCG_CAP[24] (MCG_SER_P) is set.

...

15.9.2.1 Correction Report Filtering (F) Bit

Starting with Intel Core Duo processors, bit 12 in the "Form" column in Table 15-9 is used to indicate that a particular posting to a log may be the last posting for corrections in that line/entry, at least for some time:

- 0 in bit 12 indicates "normal" filtering (original P6/Pentium4/Atom/Xeon processor meaning).
- 1 in bit 12 indicates "corrected" filtering (filtering is activated for the line/entry in the posting). Filtering means that some or all of the subsequent corrections to this entry (in this structure) will not be posted. The enhanced error reporting introduced with the Intel Core Duo processors is based on tracking the lines affected by repeated corrections (see Section 15.4, "Enhanced Cache Error reporting"). This capability is indicated by IA32_MCG_CAP[11]. Only the first few correction events for a line are posted; subsequent redundant correction events to the same line are not posted. Uncorrected events are always posted.

The behavior of error filtering after crossing the yellow threshold is model-specific. Filtering has meaning only for corrected errors (UC=0 in IA32_MCi_STATUS MSR). System software must ignore filtering bit (12) for uncorrected errors.

...

15.9.3.1 Architecturally Defined SRAO Errors

The following two SRAO errors are architecturally defined.

- UCR Errors detected by memory controller scrubbing; and
- UCR Errors detected during L3 cache (L3) explicit writebacks.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 15-9). Their values and compound encoding format are given in Table 15-15.

Table 15-15. MCA Compound Error Code Encoding for SRAO Errors

Type	MCACOD Value	MCA Error Code Encoding[1]
Memory Scrubbing	C0H - CFH	0000_0000_1100_CCCC 000F 0000 1MMM CCCC (Memory Controller Error), where Memory subfield MMM = 100B (memory scrubbing) Channel subfield CCCC = channel # or generic
L3 Explicit Writeback	17AH	0000_0001_0111_1010 000F 0001 RRRR TTLL (Cache Hierarchy Error) where Request subfields RRRR = 0111B (Eviction) Transaction Type subfields TT = 10B (Generic) Level subfields LL = 10B

NOTES:
1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 15-16 lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAO errors.

Table 15-16. IA32_MCi_STATUS Values for SRAO Errors

SRAO Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Memory Scrubbing	1	0	1	1	1	1	0	1	0	C0H-CFH
L3 Explicit Writeback	1	0	1	1	1	1	0	1	0	17AH

For both the memory scrubbing and L3 explicit writeback errors, the ADDRV and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

MCE signal is broadcast to all logical processors as outlined in Section 15.10.4.1. If LMCE is supported and enabled, some errors (not limited to UCR errors) may be delivered to only a single logical processor. System software should consult IA32_MCG_STATUS.LMCE_S to determine if the MCE signaled is only to this logical processor.

IA32_MCi_STATUS banks can be shared by logical processors within a core or within the same package. So several logical processors may find an SRAO error in the shared IA32_MCi_STATUS bank but other processors do not find it in any of the IA32_MCi_STATUS banks. Table 15-17 shows the RIPV and EIPV flag indication in the IA32_MCG_STATUS register for the memory scrubbing and L3 explicit writeback errors on both the reporting and non-reporting logical processors.

Table 15-17. IA32_MCG_STATUS Flag Indication for SRAO Errors

SRAO Type	Reporting Logical Processors		Non-reporting Logical Processors	
	RIPV	EIPV	RIPV	EIPV
Memory Scrubbing	1	0	1	0
L3 Explicit Writeback	1	0	1	0

15.9.3.2 Architecturally Defined SRAR Errors

The following two SRAR errors are architecturally defined.

- UCR Errors detected on data load; and
- UCR Errors detected on instruction fetch.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 15-9). Their values and compound encoding format are given in Table 15-18.

Table 15-18. MCA Compound Error Code Encoding for SRAR Errors

Type	MCACOD Value	MCA Error Code Encoding[1]
Data Load	134H	0000_0001_0011_0100 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0011B (Data Load) Transaction Type subfield TT= 01B (Data) Level subfield LL = 00B (Level 0)
Instruction Fetch	150H	0000_0001_0101_0000 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0101B (Instruction Fetch) Transaction Type subfield TT= 00B (Instruction) Level subfield LL = 00B (Level 0)

NOTES:

1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 15-19 lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAR errors.

Table 15-19. IA32_MCi_STATUS Values for SRAR Errors

SRAR Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Data Load	1	0	1	1	1	1	0	1	1	134H
Instruction Fetch	1	0	1	1	1	1	0	1	1	150H

For both the data load and instruction fetch errors, the ADDRV and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

MCE signal is broadcast to all logical processors on the system on which the UCR errors are supported, except when the processor supports LMCE and LMCE is enabled by system software (see Section 15.3.1.5). The IA32_MCG_STATUS MSR allows system software to distinguish the affected logical processor of an SRAR error amongst logical processors that observed SRAR via MCi_STATUS bank.

Table 15-20 shows the RIPV and EIPV flag indication in the IA32_MCG_STATUS register for the data load and instruction fetch errors on both the reporting and non-reporting logical processors. The recoverable SRAR error reported by a processor may be continuable, where the system software can interpret the context of continuable as follows: the error was isolated, contained. If software can rectify the error condition in the current instruction stream, the execution context on that logical processor can be continued without loss of information.

Table 15-20. IA32_MCG_STATUS Flag Indication for SRAR Errors

SRAR Type	Affected Logical Processor			Non-Affected Logical Processors		
	RIPV	EIPV	Continuable	RIPV	EIPV	Continuable
Recoverable-continuable	1	1	Yes[1]	1	0	Yes
Recoverable-not-continuable	0	x	No			

NOTES:
1. see the definition of the context of "continuable" above and additional detail below.

...

15.10.1 Machine-Check Exception Handler

The machine-check exception (#MC) corresponds to vector 18. To service machine-check exceptions, a trap gate must be added to the IDT. The pointer in the trap gate must point to a machine-check exception handler. Two approaches can be taken to designing the exception handler:

1. The handler can merely log all the machine status and error information, then call a debugger or shut down the system.
2. The handler can analyze the reported error information and, in some cases, attempt to correct the error and restart the processor.

For Pentium 4, Intel Xeon, Intel Atom, P6 family, and Pentium processors; virtually all machine-check conditions cannot be corrected (they result in abort-type exceptions). The logging of status and error information is therefore a baseline implementation requirement.

When IA32_MCG_CAP[24] is clear, consider the following when writing a machine-check exception handler:

- To determine the nature of the error, the handler must read each of the error-reporting register banks. The count field in the IA32_MCG_CAP register gives number of register banks. The first register of register bank 0 is at address 400H.
- The VAL (valid) flag in each IA32_MC*i*_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and do not need to be checked.
- To write a portable exception handler, only the MCA error code field in the IA32_MC*i*_STATUS register should be checked. See Section 15.9, "Interpreting the MCA Error Codes," for information that can be used to write an algorithm to interpret this field.
- Correctable errors are corrected automatically by the processor. The UC flag in each IA32_MCi_STATUS register indicates whether the processor automatically corrected an error.
- The RIPV, PCC, and OVER flags in each IA32_MCi_STATUS register indicate whether recovery from the error is possible. If PCC or OVER are set, recovery is not possible. If RIPV is not set, program execution can not be restarted reliably. When recovery is not possible, the handler typically records the error information and signals an abort to the operating system.
- The RIPV flag in the IA32_MCG_STATUS register indicates whether the program can be restarted at the instruction indicated by the instruction pointer (the address of the instruction pushed on the stack when the

exception was generated). If this flag is clear, the processor may still be able to be restarted (for debugging purposes) but not without loss of program continuity.

- For unrecoverable errors, the EIPV flag in the IA32_MCG_STATUS register indicates whether the instruction indicated by the instruction pointer pushed on the stack (when the exception was generated) is related to the error. If the flag is clear, the pushed instruction may not be related to the error.
- The MCIP flag in the IA32_MCG_STATUS register indicates whether a machine-check exception was generated. Before returning from the machine-check exception handler, software should clear this flag so that it can be used reliably by an error logging utility. The MCIP flag also detects recursion. The machine-check architecture does not support recursion. When the processor detects machine-check recursion, it enters the shutdown state.

...

15.10.3 Logging Correctable Machine-Check Errors

The error handling routine for servicing the machine-check exceptions is responsible for logging uncorrected errors.

If a machine-check error is correctable, the processor does not generate a machine-check exception for it. To detect correctable machine-check errors, a utility program must be written that reads each of the machine-check error-reporting register banks and logs the results in an accounting file or data structure. This utility can be implemented in either of the following ways.

- A system daemon that polls the register banks on an infrequent basis, such as hourly or daily.
- A user-initiated application that polls the register banks and records the exceptions. Here, the actual polling service is provided by an operating-system driver or through the system call interface.
- An interrupt service routine servicing CMCI can read the MC banks and log the error. Please refer to Section 15.10.4.2 for guidelines on logging correctable machine checks.

Example 15-3 gives pseudocode for an error logging utility.

Example 15-3. Machine-Check Error Logging Pseudocode

```
Assume that execution is restartable;
IF the processor supports MCA
    THEN
    FOR each bank of machine-check registers
        DO
            READ IA32_MCi_STATUS;
            IF VAL flag in IA32_MCi_STATUS = 1
                THEN
                    IF ADDRV flag in IA32_MCi_STATUS = 1
                        THEN READ IA32_MCi_ADDR;
                    FI;
                    IF MISCV flag in IA32_MCi_STATUS = 1
                        THEN READ IA32_MCi_MISC;
                    FI;
                    IF MCIP flag in IA32_MCG_STATUS = 1
                        (* Machine-check exception is in progress *)
                        AND PCC flag in IA32_MCi_STATUS = 1
                        OR RIPV flag in IA32_MCG_STATUS = 0
                        (* execution is not restartable *)
                            THEN
                                RESTARTABILITY = FALSE;
                                return RESTARTABILITY to calling procedure;
                    FI;
```

```
                Save time-stamp counter and processor ID;
                Set IA32_MCi_STATUS to all 0s;
                Execute serializing instruction (i.e., CPUID);
            FI;
        OD;
FI;
```

If the processor supports the machine-check architecture, the utility reads through the banks of error-reporting registers looking for valid register entries. It then saves the values of the IA32_MC*i*_STATUS, IA32_MC*i*_ADDR, IA32_MC*i*_MISC and IA32_MCG_STATUS registers for each bank that is valid. The routine minimizes processing time by recording the raw data into a system data structure or file, reducing the overhead associated with polling. User utilities analyze the collected data in an off-line environment.

When the MCIP flag is set in the IA32_MCG_STATUS register, a machine-check exception is in progress and the machine-check exception handler has called the exception logging routine.

Once the logging process has been completed the exception-handling routine must determine whether execution can be restarted, which is usually possible when damage has not occurred (The PCC flag is clear, in the IA32_MC*i*_STATUS register) and when the processor can guarantee that execution is restartable (the RIPV flag is set in the IA32_MCG_STATUS register). If execution cannot be restarted, the system is not recoverable and the exception-handling routine should signal the console appropriately before returning the error status to the Operating System kernel for subsequent shutdown.

The machine-check architecture allows buffering of exceptions from a given error-reporting bank although the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors do not implement this feature. The error logging routine should provide compatibility with future processors by reading each hardware error-reporting bank's IA32_MC*i*_STATUS register and then writing 0s to clear the OVER and VAL flags in this register. The error logging utility should re-read the IA32_MC*i*_STATUS register for the bank ensuring that the valid bit is clear. The processor will write the next error into the register bank and set the VAL flags.

Additional information that should be stored by the exception-logging routine includes the processor's time-stamp counter value, which provides a mechanism to indicate the frequency of exceptions. A multiprocessing operating system stores the identity of the processor node incurring the exception using a unique identifier, such as the processor's APIC ID (see Section 10.8, "Handling Interrupts").

The basic algorithm given in Example 15-3 can be modified to provide more robust recovery techniques. For example, software has the flexibility to attempt recovery using information unavailable to the hardware. Specifically, the machine-check exception handler can, after logging carefully analyze the error-reporting registers when the error-logging routine reports an error that does not allow execution to be restarted. These recovery techniques can use external bus related model-specific information provided with the error report to localize the source of the error within the system and determine the appropriate recovery strategy.

15.10.4 Machine-Check Software Handler Guidelines for Error Recovery

15.10.4.1 Machine-Check Exception Handler for Error Recovery

When writing a machine-check exception (MCE) handler to support software recovery from Uncorrected Recoverable (UCR) errors, consider the following:

- When IA32_MCG_CAP [24] is zero, there are no recoverable errors supported and all machine-check are fatal exceptions. The logging of status and error information is therefore a baseline implementation requirement.
- When IA32_MCG_CAP [24] is 1, certain uncorrected errors called uncorrected recoverable (UCR) errors may be software recoverable. The handler can analyze the reported error information, and in some cases attempt to recover from the uncorrected error and continue execution.
- For processors on which CPUID reports DisplayFamily_DisplayModel as 06H_0EH and onward, an MCA signal is broadcast to all logical processors in the system (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-M" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Due to

the potentially shared machine check MSR resources among the logical processors on the same package/core, the MCE handler may be required to synchronize with the other processors that received a machine check error and serialize access to the machine check registers when analyzing, logging and clearing the information in the machine check registers.

  — On processors that indicate ability for local machine-check exception (MCG_LMCE_P), hardware can choose to report the error to only a single logical processor if system software has enabled LMCE by setting IA32_MCG_EXT_CTL[LMCE_EN] = 1 as outlined in Section 15.3.1.5.

- The VAL (valid) flag in each IA32_MCi_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and should not be checked.
- The MCE handler is primarily responsible for processing uncorrected errors. The UC flag in each IA32_MCi_Status register indicates whether the reported error was corrected (UC=0) or uncorrected (UC=1). The MCE handler can optionally log and clear the corrected errors in the MC banks if it can implement software algorithm to avoid the undesired race conditions with the CMCI or CMC polling handler.
- For uncorrectable errors, the EIPV flag in the IA32_MCG_STATUS register indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.
- The MCIP flag in the IA32_MCG_STATUS register indicates whether a machine-check exception was generated. When a machine check exception is generated, it is expected that the MCIP flag in the IA32_MCG_STATUS register is set to 1. If it is not set, this machine check was generated by either an INT 18 instruction or some piece of hardware signaling an interrupt with vector 18.

When IA32_MCG_CAP [24] is 1, the following rules can apply when writing a machine check exception (MCE) handler to support software recovery:

- The PCC flag in each IA32_MCi_STATUS register indicates whether recovery from the error is possible for uncorrected errors (UC=1). If the PCC flag is set for enabled uncorrected errors (UC=1 and EN=1), recovery is not possible. When recovery is not possible, the MCE handler typically records the error information and signals the operating system to reset the system.
- The RIPV flag in the IA32_MCG_STATUS register indicates whether restarting the program execution from the instruction pointer saved on the stack for the machine check exception is possible. When the RIPV is set, program execution can be restarted reliably when recovery is possible. If the RIPV flag is not set, program execution cannot be restarted reliably. In this case the recovery algorithm may involve terminating the current program execution and resuming an alternate thread of execution upon return from the machine check handler when recovery is possible. When recovery is not possible, the MCE handler signals the operating system to reset the system.
- When the EN flag is zero but the VAL and UC flags are one in the IA32_MCi_STATUS register, the reported uncorrected error in this bank is not enabled. As uncorrected errors with the EN flag = 0 are not the source of machine check exceptions, the MCE handler should log and clear non-enabled errors when the S bit is set and should continue searching for enabled errors from the other IA32_MCi_STATUS registers. Note that when IA32_MCG_CAP [24] is 0, any uncorrected error condition (VAL =1 and UC=1) including the one with the EN flag cleared are fatal and the handler must signal the operating system to reset the system. For the errors that do not generate machine check exceptions, the EN flag has no meaning.
- When the VAL flag is one, the UC flag is one, the EN flag is one and the PCC flag is zero in the IA32_MCi_STATUS register, the error in this bank is an uncorrected recoverable (UCR) error. The MCE handler needs to examine the S flag and the AR flag to find the type of the UCR error for software recovery and determine if software error recovery is possible.
- When both the S and the AR flags are clear in the IA32_MCi_STATUS register for the UCR error (VAL=1, UC=1, EN=x and PCC=0), the error in this bank is an uncorrected no-action required error (UCNA). UCNA errors are uncorrected but do not require any OS recovery action to continue execution. These errors indicate that some data in the system is corrupt, but that data has not been consumed and may not be consumed. If that data is consumed a non-UNCA machine check exception will be generated. UCNA errors are signaled in

the same way as corrected machine check errors and the CMCI and CMC polling handler is primarily responsible for handling UCNA errors. Like corrected errors, the MCA handler can optionally log and clear UCNA errors as long as it can avoid the undesired race condition with the CMCI or CMC polling handler. As UCNA errors are not the source of machine check exceptions, the MCA handler should continue searching for uncorrected or software recoverable errors in all other MC banks.

- When the S flag in the IA32_MCi_STATUS register is set for the UCR error ((VAL=1, UC=1, EN=1 and PCC=0), the error in this bank is software recoverable and it was signaled through a machine-check exception. The AR flag in the IA32_MCi_STATUS register further clarifies the type of the software recoverable errors.
- When the AR flag in the IA32_MCi_STATUS register is clear for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action optional (SRAO) error. The MCE handler and the operating system can analyze the IA32_MCi_STATUS [15:0] to implement MCA error code specific optional recovery action, but this recovery action is optional. System software can resume the program execution from the instruction pointer saved on the stack for the machine check exception when the RIPV flag in the IA32_MCG_STATUS register is set.
- Even if the OVER flag in the IA32_MCi_STATUS register is set for the SRAO error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=0), the MCE handler can take recovery action for the SRAO error logged in the IA32_MCi_STATUS register. Since the recovery action for SRAO errors is optional, restarting the program execution from the instruction pointer saved on the stack for the machine check exception is still possible for the overflowed SRAO error if the RIPV flag in the IA32_MCG_STATUS is set.
- When the AR flag in the IA32_MCi_STATUS register is set for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action required (SRAR) error. The MCE handler and the operating system must take recovery action in order to continue execution after the machine-check exception. The MCA handler and the operating system need to analyze the IA32_MCi_STATUS [15:0] to determine the MCA error code specific recovery action. If no recovery action can be performed, the operating system must reset the system.
- When the OVER flag in the IA32_MCi_STATUS register is set for the SRAR error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=1), the MCE handler cannot take recovery action as the information of the SRAR error in the IA32_MCi_STATUS register was potentially lost due to the overflow condition. Since the recovery action for SRAR errors must be taken, the MCE handler must signal the operating system to reset the system.
- When the MCE handler cannot find any uncorrected (VAL=1, UC=1 and EN=1) or any software recoverable errors (VAL=1, UC=1, EN=1, PCC=0 and S=1) in any of the IA32_MCi banks of the processors, this is an unexpected condition for the MCE handler and the handler should signal the operating system to reset the system.
- Before returning from the machine-check exception handler, software must clear the MCIP flag in the IA32_MCG_STATUS register. The MCIP flag is used to detect recursion. The machine-check architecture does not support recursion. When the processor receives a machine check when MCIP is set, it automatically enters the shutdown state.

Example 15-4 gives pseudocode for an MC exception handler that supports recovery of UCR.

Example 15-4. Machine-Check Error Handler Pseudocode Supporting UCR

```
MACHINE CHECK HANDLER:  (* Called from INT 18 handler *)
NOERROR = TRUE;
ProcessorCount = 0;
IF CPU supports MCA
    THEN
        RESTARTABILITY = TRUE;
        IF (Processor Family = 6 AND DisplayModel ≥ 0EH) OR (Processor Family > 6)
            THEN
                IF ( MCG_LMCE = 1)
                    MCA_BROADCAST = FALSE;
                ELSE
                    MCA_BROADCAST = TRUE;
```

```
                FI;
                Acquire SpinLock;
                ProcessorCount++;  (* Allowing one logical processor at a time to examine machine check registers *)
                CALL MCA ERROR PROCESSING; (* returns RESTARTABILITY and NOERROR *)
            ELSE
                MCA_BROADCAST = FALSE;
                (* Implement a rendezvous mechanism with the other processors if necessary *)
                CALL MCA ERROR PROCESSING;
        FI;
    ELSE (* Pentium(R) processor compatible *)
        READ P5_MC_ADDR
        READ P5_MC_TYPE;
        RESTARTABILITY = FALSE;
FI;

IF NOERROR = TRUE
    THEN
        IF NOT (MCG_RIPV = 1 AND MCG_EIPV = 0)
            THEN
                RESTARTABILITY = FALSE;
        FI
FI;

IF RESTARTABILITY = FALSE
    THEN
        Report RESTARTABILITY to console;
        Reset system;
FI;

IF MCA_BROADCAST = TRUE
    THEN
        IF ProcessorCount = MAX_PROCESSORS
          AND NOERROR = TRUE
            THEN
                Report RESTARTABILITY to console;
                Reset system;
        FI;
        Release SpinLock;
        Wait till ProcessorCount = MAX_PROCESSRS on system;
        (* implement a timeout and abort function if necessary *)
FI;
CLEAR IA32_MCG_STATUS;
RESUME Execution;
(* End of MACHINE CHECK HANDLER*)


MCA ERROR PROCESSING:    (* MCA Error Processing Routine called from MCA Handler *)
IF MCIP flag in IA32_MCG_STATUS = 0
    THEN (* MCIP=0 upon MCA is unexpected *)
        RESTARTABILITY = FALSE;
FI;
FOR each bank of machine-check registers
    DO
        CLEAR_MC_BANK = FALSE;
        READ IA32_MCi_STATUS;
        IF VAL Flag in IA32_MCi_STATUS = 1
            THEN
                IF UC Flag in IA32_MCi_STATUS = 1
                    THEN
                        IF Bit 24 in IA32_MCG_CAP = 0
```

```
            THEN (* the processor does not support software error recovery *)
                RESTARTABILITY = FALSE;
                NOERROR = FALSE;
                GOTO LOG MCA REGISTER;
        FI;
        (* the processor supports software error recovery *)
        IF EN Flag in IA32_MCi_STATUS = 0 AND OVER Flag in IA32_MCi_STATUS=0
            THEN (* It is a spurious MCA Log. Log and clear the register *)
                CLEAR_MC_BANK = TRUE;
                GOTO LOG MCA REGISTER;
        FI;
        IF PCC = 1 and EN = 1 in IA32_MCi_STATUS
            THEN (* processor context might have been corrupted *)
                RESTARTABILITY = FALSE;
            ELSE (* It is a uncorrected recoverable (UCR) error *)
                IF S Flag in IA32_MCi_STATUS = 0
                    THEN
                        IF AR Flag in IA32_MCi_STATUS = 0
                            THEN (* It is a uncorrected no action required (UCNA) error *)
                                GOTO CONTINUE; (* let CMCI and CMC polling handler to process *)
                            ELSE
                                RESTARTABILITY = FALSE; (* S=0, AR=1 is illegal *)
                        FI
                FI;
                IF RESTARTABILITY = FALSE
                    THEN (* no need to take recovery action if RESTARTABILITY is already false *)
                        NOERROR = FALSE;
                        GOTO LOG MCA REGISTER;
                FI;
                (* S in IA32_MCi_STATUS = 1 *)
                IF AR Flag in IA32_MCi_STATUS = 1
                    THEN (* It is a software recoverable and action required (SRAR) error *)
                        IF OVER Flag in IA32_MCi_STATUS = 1
                            THEN
                                RESTARTABILITY = FALSE;
                                NOERROR = FALSE;
                                GOTO LOG MCA REGISTER;
                        FI
                        IF MCACOD Value in IA32_MCi_STATUS is recognized
                           AND Current Processor is an Affected Processor
                            THEN
                                Implement MCACOD specific recovery action;
                                CLEAR_MC_BANK = TRUE;
                            ELSE
                                RESTARTABILITY = FALSE;
                        FI;
                    ELSE (* It is a software recoverable and action optional (SRAO) error *)
                        IF OVER Flag in IA32_MCi_STATUS = 0 AND
                         MCACOD in IA32_MCi_STATUS is recognized
                            THEN
                                Implement MCACOD specific recovery action;
                        FI;
                        CLEAR_MC_BANK = TRUE;
                FI; AR
        FI; PCC
        NOERROR = FALSE;
        GOTO LOG MCA REGISTER;
    ELSE  (* It is a corrected error; continue to the next IA32_MCi_STATUS *)
        GOTO CONTINUE;
FI; UC
```

```
        FI; VAL
LOG MCA REGISTER:
        SAVE IA32_MCi_STATUS;
        If MISCV in IA32_MCi_STATUS
            THEN
                SAVE IA32_MCi_MISC;
        FI;
        IF ADDRV in IA32_MCi_STATUS
            THEN
                SAVE IA32_MCi_ADDR;
        FI;
        IF CLEAR_MC_BANK = TRUE
            THEN
                SET all 0 to IA32_MCi_STATUS;
                If MISCV in IA32_MCi_STATUS
                    THEN
                        SET all 0 to IA32_MCi_MISC;
                FI;
                IF ADDRV in IA32_MCi_STATUS
                    THEN
                        SET all 0 to IA32_MCi_ADDR;
                FI;
        FI;
        CONTINUE:
    OD;
( *END FOR *)
RETURN;
(* End of MCA ERROR PROCESSING*)
```

...

17. 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.

...

18.12 SIXTH 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 events and non-architectural monitoring events are programmed using fixed counters and programmable counters/event select MSRS as described in Section 18.2.3.

The core PMU's capability is similar to those described in Section 18.8 through Section 18.8.5, with some differences and enhancements summarized in Table 18-37. 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 18.10.5. 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*.

...

18.12.1 Precise 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 18-49.

...

18.12.1.1 PEBS Data Format

The PEBS record format for the 6th generation Intel Core processor is reporting with encoding 0011b in IA32_PERF_CAPABILITIES[11:8]. The lay out is shown in Table 18-50. The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 18-50 PEBS Record Format for 6th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counter
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Data Source Encoding
40H	R/EBP	A8H	Latency value (core cycles)
48H	R/ESP	B0H	EventingIP
50H	R8	B8H	TX Abort Information (Section 18.10.5.1)
58H	R9	C0H	TSC
60H	R10		

The layout of PEBS records are largely identical to those shown in Table 18-39.

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 18.8.4.2), PDIR (Section 18.8.4.4), and data address profiling (Section 18.10.3).

In the core PMU of the 6th generation processor, load latency facility and PDIR capabilities and data address profiling are unchanged relative to the 4th and 5th generation Intel Core processors. Similarly, precise store is replaced by data address profiling.

With format 0010b, a snapshot of the IA32_PERF_GLOBAL_STATUS may be useful to resolve the situations when more than one of IA32_PMICx have been configured to collect PEBS data and two consecutive overflows of the PEBS-enabled counters are sufficiently far apart in time. It is also possible for the image at 90H to indicate

multiple PEBS-enabled counters have overflowed. In the latter scenario, software cannot to correlate the PEBS record entry to the multiple overflowed bits.

With PEBS record format encoding 0011b, offset 90H reports the "applicable counter" field, which is a multi-counter PEBS resolution index allowing software to correlate the PEBS record entry with the eventing PEBS overflow when multiple counters are configured to record PEBS records. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

...

18.12.1.3 Data Address Profiling

The PEBS Data address profiling on the 6th generation Intel Core processor is largely unchanged from prior generation. When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H and A8H, as shown in Table 18-41.

Table 18-52 Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	▪ **DCU Hit** (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_INST_RETIRED.STLB_MISS_STORES, MEM_INST_RETIRED.ALL_STORES, MEM_INST_RETIRED.SPLIT_STORES. ▪ Other bits are zero.
Reserved	A8H	Always zero.

18.12.1.4 PEBS Facility for Front End Events

In the 6th generation Intel Core processor, the PEBS facility has been extended to allow capturing PEBS data for some microarchitectural conditions related to front end events. The frontend microarchitectural conditions supported by PEBS requires the following interfaces:

- The IA32_PERFEVTSELx MSR must select "FrontEnd_Retired" (C6H) in the EventSelect field (bits 7:0) and umask = 01H,
- The "FRONTEND_RETIRED" event employs a new MSR, MSR_PEBS_FRONTEND, to specify the supported frontend event details, see Table 18-53.
- Program the PEBS_EN_PMCx field of IA32_PEBS_ENABLE MSR as required.

Note the AnyThread field of IA32_PERFEVTSELx is ignored by the processor for the "FRONTEND_RETIRED" event.

...

18.12.2 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 18.8.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 18-55.

- Supplier information (bits 30:16): see Table 18-56.
- Snoop response information (bits 37:31): see Table 18-57.

Table 18-55 MSR_OFFCORE_RSP_x Request_Type Definition (Skylake microarchitecture)

Bit Name	Offset	Description
DMND_DATA_RD	0	(R/W). Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count hw or sw prefetches.
DMND_RFO	1	(R/W). Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	(R/W). Counts the number of demand and DCU prefetch instruction cacheline reads. Does not count L2 code read prefetches.
Reserved	6:3	Reserved
PF_L3_DATA_RD	7	(R/W). Counts the number of MLC prefetches into L3.
PF_L3_RFO	8	(R/W). Counts the number of RFO requests generated by MLC prefetches to L3.
Reserved	10:9	Reserved
STRM_ST	11	(R/W). Counts the number of streaming store requests.
Reserved	14:12	Reserved
OTHER	15	(R/W). Any other request that crosses IDI, including I/O.

Table 18-56 lists the supplier information field that apply to 6th generation Intel Core processors. (CPUID signature 06_4EH, 06_5EH).

Table 18-56 MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_4EH, 06_5EH)

Subtype	Bit Name	Offset	Description
Common	Any	16	(R/W). Catch all value for any response types.
Supplier Info	NO_SUPP	17	(R/W). No Supplier Information available.
	L3_HITM	18	(R/W). M-state initial lookup stat in L3.
	L3_HITE	19	(R/W). E-state
	L3_HITS	20	(R/W). S-state
	Reserved	21	Reserved
	L4_HIT	22	(R/W). L4 Cache (if L4 is present in the processor)
	Reserved	25:23	Reserved
	DRAM	26	(R/W). Local Node
	Reserved	29:27	Reserved
	SPL_HIT	30	(R/W). L4 cache super line hit (if L4 is present in the processor)

Table 18-57 lists the snoop information field that apply to processors with CPUID signature 06_4EH, 06_5EH.

Table 18-57 MSR_OFFCORE_RSP_x Snoop Info Field Definition (CPUID Signature 06_4EH, 06_5EH)

Subtype	Bit Name	Offset	Description
Snoop Info	SNOOP_NONE	31	(R/W). No details on snoop-related information
	SNOOP_NOT_NEEDED	32	(R/W). No snoop was needed to satisfy the request.
	SNOOP_MISS	33	(R/W). A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitI was returned by all cores -For LLC Miss, RspI was returned by all sockets and data was returned from DRAM.
	SNOOP_HIT_NO_FWD	34	(R/W). A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO) -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD) -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S) In the LLC Miss case, data is returned from DRAM.
	SNOOP_HIT_WITH_FWD	35	(R/W). A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	SNOOP_HITM	36	(R/W). A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD) -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO) -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	SNOOP_NON_DRAM	37	(R/W). Target was non-DRAM system address. This includes MMIO transactions.

...

18. 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.

...

19.2 PERFORMANCE MONITORING EVENTS FOR NEXT GENERATION INTEL CORE PROCESSOR

The next generation Intel Core processors are based on the Skylake microarchitecture. They support the architectural performance-monitoring events listed in Table 19-1. 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 19-7 lists performance events supporting Intel TSX (see Section 18.10.5) and are applicable to processors based on Skylake microarchitecture. Where Skylake microarchitecture implements TSX-related event semantics that differ from Table 19-7, they are listed inTable 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.

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LD_BLOCKS.STORE_FORWARD	Loads blocked by overlapping with store buffer that cannot be forwarded.	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	
07H	01H	LD_BLOCKS_PARTIAL.ADDRESS_ALIAS	False dependencies in MOB due to partial compare on address.	
08H	01H	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Load misses in all TLB levels that cause a page walk of any page size.	
08H	0EH	DTLB_LOAD_MISSES.WALK_COMPLETED	Load miss in all TLB levels causes a page walk that completes. (All page sizes)	
08H	10H	DTLB_LOAD_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for a load.	
08H	10H	DTLB_LOAD_MISSES.WALK_ACTIVE	Cycles when at least one PMH is busy with a walk for a load.	CMSK1
08H	20H	DTLB_LOAD_MISSES.STLB_HIT	Loads that miss the DTLB but hit STLB.	
0DH	01H	INT_MISC.RECOVERY_CYCLES	Core cycles the allocator was stalled due to recovery from earlier machine clear event for this thread (e.g. misprediction or memory order conflict)	
0DH	01H	INT_MISC.RECOVERY_CYCLES_ANY	Core cycles the allocator was stalled due to recovery from earlier machine clear event for any logical thread in this processor core.	AnyT
0DH	80H	INT_MISC.CLEAR_RESTEER_CYCLES	Cycles the issue-stage is waiting for front-end to fetch from resteered path following branch misprediction or machine clear events.	
0EH	01H	UOPS_ISSUED.ANY	The number of Uops issued by the RAT to RS.	
0EH	01H	UOPS_ISSUED.STALL_CYCLES	Cycles when the RAT does not issue uops to RS for the thread.	CMSK1, INV
0EH	02H	UOPS_ISSUED.VECTOR_WIDTH_MISMATCH	Uops inserted at issue-stage in order to preserve upper bits of vector registers.	
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA or similar uops allocated. Such uop has 3 sources (e.g. 2 sources + immediate) regardless if as a result of LEA instruction or not.	
14H	01H	ARITH.FPU_DIVIDER_ACTIVE	Cycles when divider is busy executing divide or square root operations. Accounts for FP operations including integer divides.	

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	21H	L2_RQSTS.DEMAND_DATA_RD_MISS	Demand Data Read requests that missed L2, no rejects.	
24H	22H	L2_RQSTS.RFO_MISS	RFO requests that missed L2,	
24H	24H	L2_RQSTS.CODE_RD_MISS	L2 cache misses when fetching instructions,	
24H	27H	L2_RQSTS.ALL_DEMAND_MISS	Demand requests that missed L2,	
24H	38H	L2_RQSTS.PF_MISS	Requests from the L1/L2/L3 hardware prefetchers or Load software prefetches that miss L2 cache	
24H	3FH	L2_RQSTS.MISS	All requests that missed L2,	
24H	41H	L2_RQSTS.DEMAND_DATA_RD_HIT	Demand Data Read requests that hit L2 cache.	
24H	42H	L2_RQSTS.RFO_HIT	RFO requests that hit L2 cache.	
24H	44H	L2_RQSTS.CODE_RD_HIT	L2 cache hits when fetching instructions,	
24H	D8H	L2_RQSTS.PF_HIT	Prefetches that hit L2.	
24H	E1H	L2_RQSTS.ALL_DEMAND_DATA_RD	All demand data read requests to L2.	
24H	E2H	L2_RQSTS.ALL_RFO	All L RFO requests to L2.	
24H	E4H	L2_RQSTS.ALL_CODE_RD	All L2 code requests.	
24H	E7H	L2_RQSTS.ALL_DEMAND_REFERENCES	All demand requests to L2.	
24H	F8H	L2_RQSTS.ALL_PF	All requests from the L1/L2/L3 hardware prefetchers or Load software prefetches	
24H	EFH	L2_RQSTS.REFERENCES	All requests to L2.	
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the L3 cache.	See Table 19-1.
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the L3 cache.	See Table 19-1.
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Cycles while the logical processor is not in a halt state.	See Table 19-1.
3CH	00H	CPU_CLK_UNHALTED.THREAD_P_ANY	Cycles while at least one logical processor is not in a halt state.	AnyT
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Reference cycles when the logical processor is unhalted (counts at 100 MHz rate)	See Table 19-1.
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK_ANY	Reference cycles when at least one logical processor in the processor core is unhalted (counts at 100 MHz rate)	AnyT
3CH	02H	CPU_CLK_THREAD_UNHALTED.ONE_THREAD_ACTIVE	Count XClk pulses when this thread is unhalted and the other thread is halted.	
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle.	
48H	01H	L1D_PEND_MISS.PENDING_CYCLES	Cycles with at least one outstanding L1D misses from this logical processor	CMSK1
48H	01H	L1D_PEND_MISS.PENDING_CYCLES_ANY	Cycles with at least one outstanding L1D misses from any logical processor in this core.	CMSK1, AnyT

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
48H	02H	L1D_PEND_MISS.FB_FULL	Number of times a request needed a FB entry but there was no entry available for it. That is the FB unavailability was dominant reason for blocking the request. A request includes cacheable/uncacheable demands that is load, store or SW prefetch. HWP are excluded.	
49H	01H	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Store misses in all TLB levels that cause page walks	
49H	0EH	DTLB_STORE_MISSES.WALK_COMPLETED	Counts completed page walks in any TLB levels due to store misses (All page sizes).	
49H	10H	DTLB_STORE_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for a store.	
49H	10H	DTLB_STORE_MISSES.WALK_ACTIVE	Cycles when at least one PMH is busy with a page walk for a store.	CMSK1
49H	20H	DTLB_STORE_MISSES.STLB_HIT	Store misses that missed DTLB but hit STLB.	
4CH	01H	LOAD_HIT_PRE.HW_PF	Demand load dispatches that hit fill buffer allocated for software prefetch.	
4FH	10H	EPT.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a EPT walk for any request type.	
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
5EH	01H	RS_EVENTS.EMPTY_CYCLES	Cycles the RS is empty for the thread.	
5EH	01H	RS_EVENTS.EMPTY_END	Counts end of periods where the Reservation Station (RS) was empty. Could be useful to precisely locate Frontend Latency Bound issues.	CMSK1, INV
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Increment each cycle of the number of offcore outstanding Demand Data Read transactions in SQ to uncore.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_DATA_RD	Cycles with at least one offcore outstanding Demand Data Read transactions in SQ to uncore.	CMSK1
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD_GE_6	Cycles with at least 6 offcore outstanding Demand Data Read transactions in SQ to uncore.	CMSK6
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Increment each cycle of the number of Offcore outstanding Demand code Read transactions in SQ to uncore.	
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_CODE_RD	Cycles with at least one offcore outstanding Demand code Read transactions in SQ to uncore.	CMSK1
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Increment each cycle of the number of Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_RFO	Cycles with at least one offcore outstanding RFO transactions in SQ to uncore.	CMSK1

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Increment each cycle of the number of Offcore outstanding cacheable data read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DATA_RD	Cycles with at least one offcore outstanding data read transactions in SQ to uncore.	CMSK1
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD	Increment each cycle of the number of Offcore outstanding demand data read requests from SQ that missed L3.	
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_L3_MISS_DEMAND_DATA_RD	Cycles with at least one offcore outstanding Demand Data Read requests from SQ that missed L3.	CMSK1
60H	10H	OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD_GE_6	Cycles with at least one offcore outstanding Demand Data Read requests from SQ that missed L3.	CMSK6
63H	02H	LOCK_CYCLES.CACHE_LOCK_DURATION	Cycles in which the L1D is locked.	
79H	04H	IDQ.MITE_UOPS	Increment each cycle # of uops delivered to IDQ from MITE path.	
79H	04H	IDQ.MITE_CYCLES	Cycles when uops are being delivered to IDQ from MITE path	CMSK1
79H	08H	IDQ.DSB_UOPS	Increment each cycle. # of uops delivered to IDQ from DSB path.	
79H	08H	IDQ.DSB_CYCLES	Cycles when uops are being delivered to IDQ from DSB path	CMSK1
79H	10H	IDQ.MS_DSB_UOPS	Increment each cycle # of uops delivered to IDQ by DSB when MS_busy.	
79H	18H	IDQ.ALL_DSB_CYCLES_ANY_UOPS	Cycles DSB is delivered at least one uops.	CMSK1
79H	18H	IDQ.ALL_DSB_CYCLES_4_UOPS	Cycles DSB is delivered four uops.	CMSK4
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ by MITE when MS_busy.	
79H	24H	IDQ.ALL_MITE_CYCLES_ANY_UOPS	Counts cycles MITE is delivered at least one uops.	CMSK1
79H	24H	IDQ.ALL_MITE_CYCLES_4_UOPS	Counts cycles MITE is delivered four uops.	CMSK4
79H	30H	IDQ.MS_UOPS	Increment each cycle # of uops delivered to IDQ while MS is busy.	
79H	30H	IDQ.MS_SWITCHES	Number of switches from DSB or MITE to MS.	EDG
79H	30H	IDQ.MS_CYCLES	Cycles MS is delivered at least one uops.	CMSK1
80H	04H	ICACHE_16B.IFDATA_STALL	Cycles where a code fetch is stalled due to L1 instruction cache miss.	
80H	04H	ICACHE_64B.IFDATA_STALL	Cycles where a code fetch is stalled due to L1 instruction cache tag miss.	

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
83H	01H	ICACHE_64B.IFTAG_HIT	Instruction fetch tag lookups that hit in the instruction cache (L1I). Counts at 64-byte cache-line granularity.	
83H	02H	ICACHE_64B.IFTAG_MISS	Instruction fetch tag lookups that miss in the instruction cache (L1I). Counts at 64-byte cache-line granularity.	
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses at all ITLB levels that cause page walks	
85H	0EH	ITLB_MISSES.WALK_COMPLETED	Counts completed page walks in any TLB levels due to code fetch misses (All page sizes).	
85H	10H	ITLB_MISSES.WALK_PENDING	Counts 1 per cycle for each PMH that is busy with a page walk for an instruction fetch request.	
85H	20H	ITLB_MISSES.STLB_HIT	ITLB misses that hit STLB.	
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Count issue pipeline slots where no uop was delivered from the frontend to the backend when there is no backend stall.	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_0_UOP_DELIV.CORE	Cycles which 4 issue pipeline slots had no uop delivered from the frontend to the backend when there is no backend stall.	CMSK4
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_n_UOP_DELIV.CORE	Cycles which “4-n” issue pipeline slots had no uop delivered from the frontend to the backend when there is no backend stall.	Set CMSK = 4-n, n = 1, 2, 3
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CYCLES_FE_WAS_OK	Cycles which frontend delivered 4 uops or the RAT was stalling FE.	CMSK, INV
A1H	01H	UOPS_DISPATCHED_PORT.PORT_0	Counts the number of cycles in which a uop is dispatched to port 0.	
A1H	02H	UOPS_DISPATCHED_PORT.PORT_1	Counts the number of cycles in which a uop is dispatched to port 1.	
A1H	04H	UOPS_DISPATCHED_PORT.PORT_2	Counts the number of cycles in which a uop is dispatched to port 2.	
A1H	08H	UOPS_DISPATCHED_PORT.PORT_3	Counts the number of cycles in which a uop is dispatched to port 3.	
A1H	10H	UOPS_DISPATCHED_PORT.PORT_4	Counts the number of cycles in which a uop is dispatched to port 4.	
A1H	20H	UOPS_DISPATCHED_PORT.PORT_5	Counts the number of cycles in which a uop is dispatched to port 5.	
A1H	40H	UOPS_DISPATCHED_PORT.PORT_6	Counts the number of cycles in which a uop is dispatched to port 6.	
A1H	80H	UOPS_DISPATCHED_PORT.PORT_7	Counts the number of cycles in which a uop is dispatched to port 7.	
A2H	01H	RESOURCE_STALLS.ANY	Resource-related stall cycles	

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A2H	08H	RESOURCE_STALLS.SB	Cycles stalled due to no store buffers available (not including draining form sync).	
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_MISS	Cycles while L2 cache miss demand load is outstanding.	CMSK1
A3H	02H	CYCLE_ACTIVITY.CYCLES_L3_MISS	Cycles while L3 cache miss demand load is outstanding.	CMSK2
A3H	04H	CYCLE_ACTIVITY.STALLS_TOTAL	Total execution stalls	CMSK4
A3H	05H	CYCLE_ACTIVITY.STALLS_L2_MISS	Execution stalls while L2 cache miss demand load is outstanding.	CMSK5
A3H	06H	CYCLE_ACTIVITY.STALLS_L3_MISS	Execution stalls while L3 cache miss demand load is outstanding.	CMSK6
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_MISS	Cycles while L1 data cache miss demand load is outstanding.	CMSK8
A3H	0CH	CYCLE_ACTIVITY.STALLS_L1D_MISS	Execution stalls while L1 data cache miss demand load is outstanding.	CMSK12
A3H	10H	CYCLE_ACTIVITY.CYCLES_MEM_ANY	Cycles while memory subsystem has an outstanding load.	CMSK16
A3H	14H	CYCLE_ACTIVITY.STALLS_MEM_ANY	Execution stalls while memory subsystem has an outstanding load.	CMSK20
A6H	01H	EXE_ACTIVITY.EXE_BOUND_0_PORTS	Cycles for which no uops began execution, the Reservation Station was not empty, the Store Buffer was full and there was no outstanding load.	
A6H	02H	EXE_ACTIVITY.1_PORTS_UTIL	Cycles for which one uop began execution on any port, and the Reservation Station was not empty.	
A6H	04H	EXE_ACTIVITY.2_PORTS_UTIL	Cycles for which two uops began execution, and the Reservation Station was not empty.	
A6H	08H	EXE_ACTIVITY.3_PORTS_UTIL	Cycles for which three uops began execution, and the Reservation Station was not empty.	
A6H	04H	EXE_ACTIVITY.4_PORTS_UTIL	Cycles for which four uops began execution, and the Reservation Station was not empty.	
A6H	40H	EXE_ACTIVITY.BOUND_ON_STORES	Cycles where the Store Buffer was full and no outstanding load.	
A8H	01H	LSD.UOPS	Number of uops delivered by the LSD.	
A8H	01H	LSD.CYCLES_ACTIVE	Cycles with at least one uop delivered by the LSD and none from the decoder.	CMSK1
A8H	01H	LSD.CYCLES_4_UOPS	Cycles with 4 uops delivered by the LSD and none from the decoder.	CMSK4
ABH	02H	DSB2MITE_SWITCHES.PENALTY_CYCLES	DSB-to-MITE switch true penalty cycles.	
AEH	01H	ITLB.ITLB_FLUSH	Flushing of the Instruction TLB (ITLB) pages, includes 4k/2M/4M pages.	
B0H	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.	

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B0H	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Demand code read requests sent to uncore.	
B0H	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM.	
B0H	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	
B0H	10H	OFFCORE_REQUESTS.L3_MISS_DEMAND_DATA_RD	Demand data read requests that missed L3	
B0H	80H	OFFCORE_REQUESTS.ALL_REQUESTS	Any memory transaction that reached the SQ.	
B1H	01H	UOPS_EXECUTED.THREAD	Counts the number of uops that begin execution across all ports.	
B1H	01H	UOPS_EXECUTED.STALL_CYCLES	Cycles which there were no uops began execution.	CMSK, INV
B1H	01H	UOPS_EXECUTED.CYCLES_GE_1_UOP_EXEC	Cycles which there was al least one uop began execution.	CMSK1
B1H	01H	UOPS_EXECUTED.CYCLES_GE_2_UOP_EXEC	Cycles which there were at least two uop began execution.	CMSK2
B1H	01H	UOPS_EXECUTED.CYCLES_GE_3_UOP_EXEC	Cycles which there were at least three uop began execution.	CMSK3
B1H	01H	UOPS_EXECUTED.CYCLES_GE_4_UOP_EXEC	Cycles which there were at least four uop began execution.	CMSK4
B1H	02H	UOPS_EXECUTED.CORE	Counts the number of uops from any logical processor in this core that begin execution.	
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_1	Cycles which there was al least one uop, from any logical processor in this core, began execution.	CMSK1
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_2	Cycles which there were al least two uops, from any logical processor in this core, began execution.	CMSK2
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_3	Cycles which there were al least three uops, from any logical processor in this core, began execution.	CMSK3
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_GE_4	Cycles which there were al least four uops, from any logical processor in this core, began execution.	CMSK4
B1H	02H	UOPS_EXECUTED.CORE_CYCLES_NONE	Cycles which there were no uops from any logical processor in this core that began execution.	CMSK1, INV
B1H	10H	UOPS_EXECUTED.X87	Counts the number of X87 uops that begin execution.	
B2H	01H	OFF_CORE_REQUEST_BUFFER.SQ_FULL	Offcore requests buffer cannot take more entries for this core.	
B7H	01H	OFF_CORE_RESPONSE_0	see Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A7H
BDH	01H	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries	
BDH	01H	TLB_FLUSH.STLB_ANY	STLB flush attempts	

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C0H	00H	INST_RETIRED.ANY_P	Number of instructions at retirement.	See Table 19-1.
C0H	01H	INST_RETIRED.PREC_DIST	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution.	PMC1 only;
C0H	01H	INST_RETIRED.TOTAL_CYCLES	Number of cycles using always true condition applied to PEBS instructions retired event.	CMSK10, PS
C1H	3FH	OTHER_ASSISTS.ANY	Number of times a microcode assist is invoked by HW other than FP-assist. Examples include AD (page Access Dirty) and AVX* related assists.	
C2H	01H	UOPS_RETIRED.STALL_CYCLES	Cycles without actually retired uops.	CMSK1, INV
C2H	01H	UOPS_RETIRED.TOTAL_CYCLES	Cycles with less than 10 actually retired uops.	CMSK10, INV
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Retirement slots used.	
C3H	01H	MACHINE_CLEARS.COUNT	Number of machine clears of any type.	CMSK1, EDG
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEARS.SMC	Number of self-modifying-code machine clears detected.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions that retired.	See Table 19-1.
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	PS
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.	PS
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	PS
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	PS
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions retired.	
C4H	20H	BR_INST_RETIRED.NEAR_TAKEN	Number of near taken branches retired.	PS
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.	PS
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement	See Table 19-1.
C5H	01H	BR_MISP_RETIRED.CONDITIONAL	Mispredicted conditional branch instructions retired.	PS
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	PS
C5H	20H	BR_MISP_RETIRED.NEAR_TAKEN	Number of near branch instructions retired that were mispredicted and taken.	PS
C6H	01H	FRONTEND_RETIRED.DSB_MISS	Retired Instructions which experienced DSB miss. Specify MSR_PEBS_FRONTEND.EVTSEL=11H	PS

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C6H	01H	FRONTEND_RETIRED.L1I_MISS	Retired Instructions which experienced Instruction L1 cache true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=12H	PS
C6H	01H	FRONTEND_RETIRED.L2_MISS	Retired Instructions which experienced L2 cache true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=13H	PS
C6H	01H	FRONTEND_RETIRED.ITLB_MISS	Retired Instructions which experienced ITLB true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=14H	PS
C6H	01H	FRONTEND_RETIRED.STLB_MISS	Retired Instructions which experienced STLB true miss. Specify MSR_PEBS_FRONTEND.EVTSEL=15H	PS
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_16	Retired Instructions that are fetched after an interval where the front end delivered no uops for at least 16 cycles. Specify the following fields in MSR_PEBS_FRONTEND: EVTSEL=16H, IDQ_Bubble_Length =16, IDQ_Bubble_Width = 4.	PS
C6H	01H	FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_m	Retired Instructions that are fetched after an interval where the front end had 'm' IDQ slots delivered no uops for at least 2 cycles. Specify the following fields in MSR_PEBS_FRONTEND: EVTSEL=16H, IDQ_Bubble_Length =2, IDQ_Bubble_Width = m	PS, m = 1, 2, 3
C7H	01H	FP_ARITH_INST_RETIRED.SCALAR_DOUBLE	Number of double-precision, floating-point, scalar SSE/AVX computational instructions that are retired. Each scalar FMA instruction count as 2.	Software may treat each count as one DP FLOP.
C7H	02H	FP_ARITH_INST_RETIRED.SCALAR_SINGLE	Number of single-precision, floating-point, scalar SSE/AVX computational instructions that are retired. Each scalar FMA instruction count as 2.	Software may treat each count as one SP FLOP.
C7H	04H	FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	Number of double-precision, floating-point, 128-bit SSE/AVX computational instructions that are retired. Each 128-bit FMA or (V)DPPD instruction count as 2.	Software may treat each count as two DP FLOPs.
C7H	08H	FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	Number of single-precision, floating-point, 128-bit SSE/AVX computational instructions that are retired. Each 128-bit FMA or (V)DPPS instruction count as 2.	Software may treat each count as four SP FLOPs.
C7H	10H	FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	Number of double-precision, floating-point, 256-bit SSE/AVX computational instructions that are retired. Each 256-bit FMA instruction count as 2.	Software may treat each count as four DP FLOPs.
C7H	20H	FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	Number of single-precision, floating-point, 256-bit SSE/AVX computational instructions that are retired. Each 256-bit FMA or VDPPS instruction count as 2.	Software may treat each count as eight SP FLOPs.
CAH	1EH	FP_ASSIST.ANY	Cycles with any input/output SSE* or FP assists.	CMSK1
CBH	01H	HW_INTERRUPTS.RECEIVED	Number of hardware interrupts received by the processor.	
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY	Randomly sampled loads whose latency is above a user defined threshold. A small fraction of the overall loads are sampled due to randomization.	Specify threshold in MSR 3F6H. PSDLA

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D0H	11H	MEM_INST_RETIRED.STLB_MISS_LOADS	Retired load instructions that miss the STLB.	PSDLA
D0H	12H	MEM_INST_RETIRED.STLB_MISS_STORES	Retired store instructions that miss the STLB.	PSDLA
D0H	21H	MEM_INST_RETIRED.LOCK_LOADS	Retired load instructions with locked access.	PSDLA
D0H	41H	MEM_INST_RETIRED.SPLIT_LOADS	Number of load instructions retired with cache-line splits that may impact performance.	PSDLA
D0H	42H	MEM_INST_RETIRED.SPLIT_STORES	Number of store instructions retired with line-split.	PSDLA
D0H	81H	MEM_INST_RETIRED.ALL_LOADS	All retired load instructions.	PSDLA
D0H	82H	MEM_INST_RETIRED.ALL_STORES	All retired store instructions.	PSDLA
D1H	01H	MEM_LOAD_RETIRED.L1_HIT	Retired load Instructions with L1 cache hits as data sources.	PSDLA
D1H	02H	MEM_LOAD_RETIRED.L2_HIT	Retired load Instructions with L2 cache hits as data sources.	PSDLA
D1H	04H	MEM_LOAD_RETIRED.L3_HIT	Retired load Instructions with L3 cache hits as data sources.	PSDLA
D1H	08H	MEM_LOAD_RETIRED.L1_MISS	Retired load Instructions missed L1 cache as data sources.	PSDLA
D1H	10H	MEM_LOAD_RETIRED.L2_MISS	Retired load Instructions missed L2. Unknown data source excluded.	PSDLA
D1H	20H	MEM_LOAD_RETIRED.L3_MISS	Retired load Instructions missed L3. Excludes unknown data source.	PSDLA
D1H	40H	MEM_LOAD_RETIRED.FB_HIT	Retired load Instructions which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.	PSDLA
D2H	01H	MEM_LOAD_L3_HIT_RETIRED.XSNP_MISS	Retired load Instructions which data sources were L3 hit and cross-core snoop missed in on-pkg core cache.	PSDLA
D2H	02H	MEM_LOAD_L3_HIT_RETIRED.XSNP_HIT	Retired load Instructions which data sources were L3 and cross-core snoop hits in on-pkg core cache.	PSDLA
D2H	04H	MEM_LOAD_L3_HIT_RETIRED.XSNP_HITM	Retired load Instructions which data sources were HitM responses from shared L3.	PSDLA
D2H	08H	MEM_LOAD_L3_HIT_RETIRED.XSNP_NONE	Retired load Instructions which data sources were hits in L3 without snoops required.	PSDLA
E6H	01H	BACLEARS.ANY	Number of front end re-steers due to BPU misprediction.	
F0H	40H	L2_TRANS.L2_WB	L2 writebacks that access L2 cache.	

Table 19-3. Non-Architectural Performance Events of the Processor Core Supported by Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
F1H	07H	L2_LINES_IN.ALL	L2 cache lines filling L2.	

CMSK1: Counter Mask = 1 required; CMSK4: CounterMask = 4 required; CMSK6: CounterMask = 6 required; CMSK8: CounterMask = 8 required; CMSK10: CounterMask = 10 required; CMSK12: CounterMask = 12 required; CMSK16: CounterMask = 16 required; CMSK20: CounterMask = 20 required.

AnyT: AnyThread = 1 required.

INV: Invert = 1 required.

EDG: EDGE = 1 required.

PSDLA: Also supports PEBS and DataLA.

PS: Also supports PEBS.

Table 19-7 lists performance events supporting Intel TSX (see Section 18.10.5) and are applicable to processors based on Skylake microarchitecture. Where Skylake microarchitecture implements TSX-related event semantics that differ from Table 19-7, they are listed in Table 19-4.

Table 19-4. Intel TSX Performance Event Addendum in Processors based on Skylake Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
54H	02H	TX_MEM.ABORT_CAPACITY	Number of times a transactional abort was signaled due to a data capacity limitation for transactional reads or writes	

…

19.3 PERFORMANCE MONITORING EVENTS FOR THE INTEL® CORE™ M AND FIFTH GENERATION INTEL CORE PROCESSORS

…

Table 19-8 lists performance events supporting Intel TSX (see Section 18.10.5) and are applicable to processors based on Broadwell microarchitecture. Where Broadwell microarchitecture implements TSX-related event semantics that differ from Table 19-8, they are listed in Table 19-6.

Table 19-6 Intel® TSX Performance Event Addendum in Processors based on Broadwell Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
54H	02H	TX_MEM.ABORT_CAPACITY	Number of times a transactional abort was signaled due to a data capacity limitation for transactional reads or writes	

…

19. Updates to Chapter 24, Volume 3C

Change bars show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

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.
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.

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
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.

Table 24-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
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.
16	RDSEED exiting	This control determines whether executions of RDSEED cause VM exits.
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.
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).

...

24.6.5 Time-Stamp Counter Offset and Multiplier

The VM-execution control fields include a 64-bit **TSC-offset** field. If the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1, this field controls executions of the RDTSC and RDTSCP instructions. It also controls executions of the RDMSR instruction that read from the IA32_TIME_STAMP_COUNTER MSR. For all of these, the value of the TSC offset is added to the value of the time-stamp counter, and the sum is returned to guest software in EDX:EAX.

Processors that support the 1-setting of the "use TSC scaling" control also support a 64-bit **TSC-multiplier** field. If this control is 1 (and the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1), this field also affects the executions of the RDTSC, RDTSCP, and RDMSR instructions identified above. Specifically, the contents of the time-stamp counter is first multiplied by the TSC multiplier before adding the TSC offset.

See Chapter 27 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation

...

20. Updates to Chapter 25, Volume 3C

Change bars show changes to Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

25.3 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation. Some of these changes are determined by the settings of certain VM-execution control fields. The following items detail such changes:[1]

- **CLTS.** Behavior of the CLTS instruction is determined by the bits in position 3 (corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:
  - If bit 3 in the CR0 guest/host mask is 0, CLTS clears CR0.TS normally (the value of bit 3 in the CR0 read shadow is irrelevant in this case), unless CR0.TS is fixed to 1 in VMX operation (see Section 23.8), in which case CLTS causes a general-protection exception.
  - If bit 3 in the CR0 guest/host mask is 1 and bit 3 in the CR0 read shadow is 0, CLTS completes but does not change the contents of CR0.TS.
  - If the bits in position 3 in the CR0 guest/host mask and the CR0 read shadow are both 1, CLTS causes a VM exit.
- **INVPCID.** Behavior of the INVPCID instruction is determined first by the setting of the "enable INVPCID" VM-execution control:
  - If the "enable INVPCID" VM-execution control is 0, INVPCID causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
  - If the "enable INVPCID" VM-execution control is 1, treatment is based on the setting of the "INVLPG exiting" VM-execution control:
    - If the "INVLPG exiting" VM-execution control is 0, INVPCID operates normally.
    - If the "INVLPG exiting" VM-execution control is 1, INVPCID causes a VM exit.
- **IRET.** Behavior of IRET with regard to NMI blocking (see Table 24-3) is determined by the settings of the "NMI exiting" and "virtual NMIs" VM-execution controls:
  - If the "NMI exiting" VM-execution control is 0, IRET operates normally and unblocks NMIs. (If the "NMI exiting" VM-execution control is 0, the "virtual NMIs" control must be 0; see Section 26.2.1.1.)
  - If the "NMI exiting" VM-execution control is 1, IRET does not affect blocking of NMIs. If, in addition, the "virtual NMIs" VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, IRET removes any virtual-NMI blocking.

  The unblocking of NMIs or virtual NMIs specified above occurs even if IRET causes a fault.
- **LMSW.** Outside of VMX non-root operation, LMSW loads its source operand into CR0[3:0], but it does not clear CR0.PE if that bit is set. In VMX non-root operation, an execution of LMSW that does not cause a VM exit (see Section 25.1.3) leaves unmodified any bit in CR0[3:0] corresponding to a bit set in the CR0 guest/host mask. An attempt to set any other bit in CR0[3:0] to a value not supported in VMX operation (see Section 23.8) causes a general-protection exception. Attempts to clear CR0.PE are ignored without fault.

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.

- **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.

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*.

- **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.
    - 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.
- **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.
  - — 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 24.6.17):
    - 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 24.6.17):

- 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.

...

21. 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.

...

26.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:[1]

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.1).
- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.2).
- If the "activate secondary controls" primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.3).

  If the "activate secondary controls" primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.
- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix A.6).
- If the "use I/O bitmaps" VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. Neither address should set any bits beyond the processor's physical-address width.[2,3]
- If the "use MSR bitmaps" VM-execution control is 1, bits 11:0 of the MSR-bitmap address must be 0. The address should not set any bits beyond the processor's physical-address width.[4]
- If the "use TPR shadow" VM-execution control is 1, the virtual-APIC address must satisfy the following checks:
  — Bits 11:0 of the address must be 0.
  — The address should not set any bits beyond the processor's physical-address width.[5]

1. If the "activate secondary controls" primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. If IA32_VMX_BASIC[48] is read as 1, these addresses must not set any bits in the range 63:32; see Appendix A.1.
4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
5. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

If all of the above checks are satisfied and the "use TPR shadow" VM-execution control is 1, bytes 3:1 of VTPR (see Section 29.1.1) may be cleared (behavior may be implementation-specific).

The clearing of these bytes may occur even if the VM entry fails. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it causes processor state to be loaded from the host-state area of the VMCS.

- If the "use TPR shadow" VM-execution control is 1 and the "virtual-interrupt delivery" VM-execution control is 0, bits 31:4 of the TPR threshold VM-execution control field must be 0.[1]
- The following check is performed if the "use TPR shadow" VM-execution control is 1 and the "virtualize APIC accesses" and "virtual-interrupt delivery" VM-execution controls are both 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 of VTPR (see Section 29.1.1).
- If the "NMI exiting" VM-execution control is 0, the "virtual NMIs" VM-execution control must be 0.
- If the "virtual NMIs" VM-execution control is 0, the "NMI-window exiting" VM-execution control must be 0.
- If the "virtualize APIC-accesses" VM-execution control is 1, the APIC-access address must satisfy the following checks:
  - Bits 11:0 of the address must be 0.
  - The address should not set any bits beyond the processor's physical-address width.[2]
- If the "use TPR shadow" VM-execution control is 0, the following VM-execution controls must also be 0: "virtualize x2APIC mode", "APIC-register virtualization", and "virtual-interrupt delivery".[3]
- If the "virtualize x2APIC mode" VM-execution control is 1, the "virtualize APIC accesses" VM-execution control must be 0.
- If the "virtual-interrupt delivery" VM-execution control is 1, the "external-interrupt exiting" VM-execution control must be 1.
- If the "process posted interrupts" VM-execution control is 1, the following must be true:[4]
  - The "virtual-interrupt delivery" VM-execution control is 1.
  - The "acknowledge interrupt on exit" VM-exit control is 1.
  - The posted-interrupt notification vector has a value in the range 0–255 (bits 15:8 are all 0).
  - Bits 5:0 of the posted-interrupt descriptor address are all 0.
  - The posted-interrupt descriptor address does not set any bits beyond the processor's physical-address width.[5]
- If the "enable VPID" VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.[6]
- If the "enable EPT" VM-execution control is 1, the EPTP VM-execution control field (see Table 24-8 in Section 24.6.11) must satisfy the following checks:[7]

---

1. "Virtual-interrupt delivery" 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 "virtual-interrupt delivery" VM-execution control were 0. See Section 24.6.2.
2. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
3. "Virtualize x2APIC mode" and "APIC-register virtualization" are secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if these controls were 0. See Section 24.6.2.
4. "Process posted interrupts" 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 "process posted interrupts" VM-execution control were 0. See Section 24.6.2.
5. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
6. "Enable VPID" 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 VPID" VM-execution control were 0. See Section 24.6.2.

— The EPT memory type (bits 2:0) must be a value supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10).

— Bits 5:3 (1 less than the EPT page-walk length) must be 3, indicating an EPT page-walk length of 4; see Section 28.2.2.

— Bit 6 (enable bit for accessed and dirty flags for EPT) must be 0 if bit 21 of the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10) is read as 0, indicating that the processor does not support accessed and dirty flags for EPT.

— Reserved bits 11:7 and 63:N (where N is the processor's physical-address width) must all be 0.

- If the "unrestricted guest" VM-execution control is 1, the "enable EPT" VM-execution control must also be 1.[1]
- If the "enable VM functions" processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear.[2] Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.11). In addition, the following check is performed based on the setting of bits in the VM-function controls (see Section 24.6.14):

  — If "EPTP switching" VM-function control is 1, the "enable EPT" VM-execution control must also be 1. In addition, the EPTP-list address must satisfy the following checks:

    - Bits 11:0 of the address must be 0.
    - The address must not set any bits beyond the processor's physical-address width.

  If the "enable VM functions" processor-based VM-execution control is 0, no checks are performed on the VM-function controls.
- If the "VMCS shadowing" VM-execution control is 1, the VMREAD-bitmap and VMWRITE-bitmap addresses must each satisfy the following checks:[3]

  — Bits 11:0 of the address must be 0.

  — The address must not set any bits beyond the processor's physical-address width.
- If the "EPT-violation #VE" VM-execution control is 1, the virtualization-exception information address must satisfy the following checks:[4]

  — Bits 11:0 of the address must be 0.

  — The address must not set any bits beyond the processor's physical-address width.

...

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:

7. "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.

1. "Unrestricted guest" and "enable EPT" are both secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if both these controls were 0. See Section 24.6.2.
2. "Enable VM functions" 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 VM functions" VM-execution control were 0. See Section 24.6.2.
3. "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.
4. "EPT-violation #VE" 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 "EPT-violation #VE" VM-execution control were 0. See Section 24.6.2.

- 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).
- If the "load IA32_PAT" VM-entry control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the "load IA32_EFER" VM-entry control is 1, the following checks are performed on the field for the IA32_EFER MSR:
  - Bits reserved in the IA32_EFER MSR must be 0.

Bit 10 (corresponding to IA32_EFER.LMA) must equal the value of the "IA-32e mode guest" VM-entry control. It must also be identical to bit 8 (LME) if bit 31 in the CR0 field (corresponding to CR0.PG) is 1.[6]

---

1. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "unrestricted guest" VM-execution control were 0. See Section 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.
6. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

...

22. 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.

...

27.2.1 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.
  - The remainder of the field (bits 31:16) 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 (Section 29.1.4); and APIC-write emulation (see Section 29.4.3.3). 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.

  - 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.

1. Bit 13 of this field is set on certain VM-entry failures; see Section 26.7.

— 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]

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."

...

23. 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 3C:* System Programming Guide, Part 3.

...

29.4.4 Instruction-Specific Considerations

Certain instructions that use linear address may cause page faults even though they do not use those addresses to access memory. The APIC-virtualization features may affect these instructions as well:

- **CLFLUSH, CLFLUSHOPT.** With regard to faulting, the processor operates as if each of these instructions reads from the linear address in its source operand. If that address translates to one on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will flush the corresponding cache line on the virtual-APIC page instead of the APIC-access page.
- **ENTER.** With regard to faulting, the processor operates if ENTER writes to the byte referenced by the final value of the stack pointer (even though it does not if its size operand is non-zero). If that value translates to an address on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will cause the APIC-write emulation appropriate to the address's page offset.
- **MASKMOVQ and MAKSMOVDQU.** Even if the instruction's mask is zero, the processor may operate with regard to faulting as if MASKMOVQ or MASKMOVDQU writes to memory (the behavior is implementation-specific). In such a situation, an APIC-access VM exit may occur.
- **MONITOR.** With regard to faulting, the processor operates as if MONITOR reads from the effective address in RAX. If the resulting linear address translates to one on the APIC-access page, the instruction may cause an APIC-access VM exit.[2] If it does not, it will monitor the corresponding address on the virtual-APIC page instead of the APIC-access page.

---

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.
2. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

- **PREFETCH.** An execution of the PREFETCH instruction that would result in an access to the APIC-access page does not cause an APIC-access VM exit. Such an access may prefetch data; if so, it is from the corresponding address on the virtual-APIC page.

Virtualization of accesses to the APIC-access page is principally intended for basic instructions such as AND, MOV, OR, TEST, XCHG, and XOR. Use of an instruction that normally operates on floating-point, SSE, AVX, or AVX-512 registers may cause an APIC-access VM exit unconditionally regardless of the page offset it accesses on the APIC-access page.

...

24. 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_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family based on Skylake microarchitecture
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture
06_4FH	Future Generation Intel Xeon processor 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
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

Table 35-1. CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
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_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
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

…

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 35.20, "MSRs in Pentium Processors."	**Pentium Processor (05_01H)**
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 35.20, "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 17.14, "Time-Stamp Counter."	05_01H
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.	

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			
		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
		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

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			
		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
		17:16	Reserved	
		18	SGX Global Enable (R/WL): This bit must be set to enable SGX leaf functions. This bit is supported only if CPUID.1:ECX.[bit 6] is set.	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		19	Reserved	
		20	LMCE On (R/WL): When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.	If IA32_MCG_CAP[27] = 1
		63:21	Reserved	
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H): 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			
		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.	
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."	0F_0H
		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.	
		4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	
		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."	0F_0H
		0	High-Temperature Interrupt Enable	
		1	Low-Temperature Interrupt Enable	
		2	PROCHOT# Interrupt Enable	
		3	FORCEPR# Interrupt Enable	
		4	Critical Temperature Interrupt Enable	
		7:5	Reserved.	
		14:8	Threshold #1 Value	
		15	Threshold #1 Interrupt Enable	

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	
		23	Threshold #2 Interrupt Enable	
		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"	0F_0H
		0	Thermal Status (RO):	
		1	Thermal Status Log (R/W):	
		2	PROCHOT # or FORCEPR# event (RO)	
		3	PROCHOT # or FORCEPR# log (R/WC0)	
		4	Critical Temperature Status (RO)	
		5	Critical Temperature Status log (R/WC0)	
		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 (default). 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.	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	**Precise 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 3. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 3. 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 3, the bit is supported. Otherwise, the bit is not supported. Writing to this bit when the maximum value is greater than 3 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 3.	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 0 (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 0 0 (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 42.12.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1	Reserved.	
		23:16	SGX_SVN_SINIT.	See Section 42.12.3, "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 18.13.4, "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)
		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 Platform Enforcement Mask Registers	**See Section 17.16.3.1, "Enumeration and Detection Support of Cache Allocation Technology"**	
C90H	3216	IA32_L3_MASK_0	**L3 CQE 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 CQE Mask for COSn (R/W)**	n = CPUID.(EAX=10H, ECX=1H):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	
		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

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	**HDC_Allow_Block (R/W)** Allow/Block this logical processor for package level HDC control. See Section 14.5.3	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved.	
DB2H	3506	IA32_THREAD_STALL	**Per-Logical_Processor HDC Idle Residency (R/O)**	If CPUID.06H:EAX.[13] = 1
		63:0	**Stall_Cycle_Cnt (R/W)** Stalled cycles due to HDC forced idle on this logical processor. See Section 14.5.4.1	If CPUID.06H:EAX.[13] = 1
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.	
		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

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			
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].

35.4 MSRS IN THE PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 35-6 lists model-specific registers (MSRs) for 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 column "Scope" lists the core/shared/package granularity of sharing in the Silvermont microarchitecture. "Core" means each processor core has a separate MSR, or a bit field not shared with another processor core. "Shared" means the MSR or the bit field is shared by more than one 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. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 35.20, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Shared	See Section 35.20, "MSRs in Pentium Processors."

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 17.14, "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
		63:33		Reserved.
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	Shared	**Processor Hard Power-On Configuration (R/W)** Enables and disables processor features; **(R)** indicates current processor configuration.
		0		Reserved.
		1		**Data Error Checking Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		2		**Response Error Checking Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		3		**AERR# Drive Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		4		**BERR# Enable for initiator bus requests (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved.
		6		Reserved.
		7		**BINIT# Driver Enable (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
		9		**Execute BIST (R/O)** 1 = Enabled; 0 = Disabled
		10		**AERR# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved.
		12		**BINIT# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		13		**Reserved.**
		14		**1 MByte Power on Reset Vector (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		**APIC Cluster ID (R/O)** Always 00B.
		19: 18		Reserved.
		21: 20		**Symmetric Arbitration ID (R/O)** Always 00B.
		26:22		**Integer Bus Frequency Ratio (R/O)**
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.
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. This part of the stack contains pointers to the **source instruction** for one of the last eight branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.12, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
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. This part of the stack contains pointers to the destination instruction for one of the last eight branches, exceptions, or interrupts taken by the processor.
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.
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.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
C1H	193	IA32_PMC0	Core	**Performance counter register** See Table 35-2.
C2H	194	IA32_PMC1	Core	**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 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	Shared	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-sate support) 001b: C1 (Behavior is the same as 000b) 100b: C4 110b: C6 111b: C7 (Silvermont only).
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_ BASE	Shared	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
FEH	254	IA32_MTRRCAP	Core	**Memory Type Range Register (R)** 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.
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.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
		1:0		**AES Configuration (RW-L)** Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES 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**
		16		**USR**
		17		**OS**
		18		**Edge**
		19		**PC**
		20		**INT**
		21		**Reserved**
		22		**EN**

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
		23		**INV**
		31:24		**CMASK**
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Core	See Table 35-2.
198H	408	IA32_PERF_STATUS	Shared	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.
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	Shared	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2.
		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	**Precise Event Based Sampling Unavailable (RO)** See Table 35-2.
		15:13		Reserved.
		16	Shared	**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.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
		23	Shared	**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	Shared	**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	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	Shared	**Offcore Response Event Select Register (R/W)**
1A7H	423	MSR_OFFCORE_RSP_1	Shared	**Offcore Response Event Select Register (R/W)**
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode (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.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		63:32		**Reserved**
1B0H	432	IA32_ENERGY_PERF_BIAS	Core	See Table 35-2.
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 (at 40H).
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.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
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	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.
38EH	910	IA32_PERF_GLOBAL_ STAUS	Core	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	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 on IA32_PMC0. (R/W)
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

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. Counts at the TSC Frequency.
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	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."
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	MSR_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_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	MSR_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
412H	1042	MSR_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	MSR_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	MSR_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	MSR_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."
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."

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
488H	1160	IA32_VMX_CR4_FIXED0	Core	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 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
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 18.13.4, "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

Table 35-6. Common MSRs in Intel Processors Based on the Silvermont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 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).

35.7 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES (BASED ON INTEL® MICROARCHITECTURE CODE NAME WESTMERE)

Intel® Xeon® Processor 5600 Series (based on Intel® microarchitecture code name Westmere) supports the MSR interfaces listed in Table 35-11, Table 35-12, plus additional MSR listed in Table 35-14. These MSRs apply to Intel Core i7, i5 and i3 processor family with CPUID signature DisplayFamily_DisplayModel of 06_25H and 06_2CH, see Table 35-1.

Table 35-15. Additional MSRs Supported by Intel Processors (Based on Intel® Microarchitecture Code Name Westmere)

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.

Table 35-15. Additional MSRs Supported by Intel Processors
(Based on Intel® Microarchitecture Code Name Westmere)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		1:0		**AES Configuration (RW-L)** Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instruction can be mis-configured if a privileged agent unintentionally writes 11b.
		63:2		Reserved.
1A7H	423	MSR_OFFCORE_RSP_1	Thread	**Offcore Response Event Select Register (R/W)**
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.
		63:48		Reserved.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 35-2.

35.8 MSRS IN INTEL® PROCESSOR FAMILY BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Table 35-16 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 35-17.

Table 35-16. 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 35.20, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 35.20, "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 17.14, "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)** 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)**

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy 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.
E4H	228	MSR_PMG_IO_CAPTURE_ BASE	Core	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
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	

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		11	Thread	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Thread	**Precise 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 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (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	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)**
		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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 17.6.2, "Filtering of Last Branch Records."
		0		**CPL_EQ_0**
		1		**CPL_NEQ_0**
		2		**JCC**
		3		**NEAR_REL_CALL**
		4		**NEAR_IND_CALL**
		5		**NEAR_RET**
		6		**NEAR_IND_JMP**
		7		**NEAR_REL_JMP**
		8		**FAR_BRANCH**
		63:9		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
283H	643	IA32_MC3_CTL2	Core	See Table 35-2.
284H	644	MSR_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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_ STAUS		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**
		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**

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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."

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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."

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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."
410H	1040	MSR_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."

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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."
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

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 18.13.4, "Debug Store (DS) Mechanism."
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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		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."

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 17.6.1, "LBR Stack."
681H	1665	MSR_ LASTBRANCH_1_FROM_IP	Thread	**Last Branch Record 1 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_ LASTBRANCH_2_FROM_IP	Thread	**Last Branch Record 2 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_ LASTBRANCH_3_FROM_IP	Thread	**Last Branch Record 3 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_ LASTBRANCH_4_FROM_IP	Thread	**Last Branch Record 4 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_ LASTBRANCH_5_FROM_IP	Thread	**Last Branch Record 5 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_ LASTBRANCH_6_FROM_IP	Thread	**Last Branch Record 6 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_ LASTBRANCH_7_FROM_IP	Thread	**Last Branch Record 7 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_ LASTBRANCH_8_FROM_IP	Thread	**Last Branch Record 8 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_ LASTBRANCH_9_FROM_IP	Thread	**Last Branch Record 9 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_ LASTBRANCH_10_FROM_IP	Thread	**Last Branch Record 10 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_ LASTBRANCH_11_FROM_IP	Thread	**Last Branch Record 11 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_ LASTBRANCH_12_FROM_IP	Thread	**Last Branch Record 12 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_ LASTBRANCH_13_FROM_IP	Thread	**Last Branch Record 13 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_ LASTBRANCH_14_FROM_IP	Thread	**Last Branch Record 14 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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.

Table 35-16. MSRs Supported by Intel® Processors based on Intel® microarchitecture code name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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 17.14.2, "IA32_TSC_AUX Register and RDTSCP Support."

35.8.1 MSRs In 2nd Generation Intel® Core™ Processor Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 35-17 lists 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 35-17. 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

Table 35-17. MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® microarchitecture code name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
		63:32		Reserved.
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU main status
		0		Fixed counter overflowed
		1		An ARB counter overflowed
		2		Reserved
		3		A CBox counter overflowed (on any slice)
		63:4		Reserved.
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore fixed counter control (R/W)
		19:0		Reserved
		20		Enable overflow propagation
		21		Reserved
		22		Enable counting
		63:23		Reserved.

Table 35-17. MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® microarchitecture code name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
395H	917	MSR_UNC_PERF_FIXED_ CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box configuration information (R/O)
		3:0		Encoded number of C-Box, derive value by "-1"
		63:4		Reserved.
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb unit, performance counter 1
3B2H	944	MSR_UNC_ARB_ 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
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.

Table 35-17. MSRs Supported by 2nd Generation Intel® Core™ Processors (Intel® microarchitecture code name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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."
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
720H	1824	MSR_UNC_CBO_2_ PERFEVTSEL0	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1824	MSR_UNC_CBO_2_ PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, performance counter 1
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.

…

35.10.1 MSRs in 4th Generation Intel® Core™ Processor Family (based on Haswell Microarchitecture)

Table 35-25 lists model-specific registers (MSRs) that are specific to 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3CH/06_45H/06_46H, see Table 35-1.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org.
		3:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s
		9:4		Reserved
		10		**I/O MWAIT Redirection Enable (R/W)**
		14:11		Reserved
		15		**CFG Lock (R/WO)**
		24:16		Reserved
		25		**C3 State Auto Demotion Enable (R/W)**
		26		**C1 State Auto Demotion Enable (R/W)**
		27		**Enable C3 Undemotion (R/W)**
		28		**Enable C1 Undemotion (R/W)**
		63:29		Reserved
17DH	390	MSR_SMM_MCA_CAP	THREAD	**Enhanced SMM Capabilities (SMM-RO)** Reports SMM capability Enhancement. Accessible only while in SMM.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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
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.
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.
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
		63:32		Reserved.
392H	914	MSR_UNC_PERF_GLOBAL_ STATUS	Package	Uncore PMU main status
		0		Fixed counter overflowed
		1		An ARB counter overflowed
		2		Reserved
		3		A CBox counter overflowed (on any slice)
		63:4		Reserved.
394H	916	MSR_UNC_PERF_FIXED_ CTRL	Package	Uncore fixed counter control (R/W)
		19:0		Reserved
		20		Enable overflow propagation
		21		Reserved
		22		Enable counting
		63:23		Reserved.
395H	917	MSR_UNC_PERF_FIXED_ CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box configuration information (R/O)
		3:0		Encoded number of C-Box, derive value by "-1"
		63:4		Reserved.
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb unit, performance counter 1
3B2H	944	MSR_UNC_ARB_ 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

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable wake on PMI
		31		Enable Freezing counter when overflow
		63:32		Reserved.
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, counter 1 event select MSR
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	**Enhanced SMM Feature Control (SMM-RW)** Reports SMM capability Enhancement. Accessible only while in SMM.
		0		**Lock (SMM-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 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		N-1:0		**LOG_PROC_STATE (SMM-RO)** Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	**SMM Blocked (SMM-RO)** Reports the blocked state of all logical processors in the package. Available only while in SMM.
		N-1:0		**LOG_PROC_STATE (SMM-RO)** Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)**
		3:0	Package	**Power Units** See Section 14.9.1, "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
613H	1555	MSR_PKG_PERF_STATUS	Package	**PKG Perf Status (R/O)** See Section 14.9.3, "Package RAPL Domain."
640H	1600	MSR_PP1_POWER_LIMIT	Package	**PP1 RAPL Power Limit Control (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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."
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	**Indicator of Frequency Clipping in Processor Cores (R/W)** **(frequency refers to processor core frequency)**
		0		**PROCHOT Status (R0)** When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		**Thermal Status (R0)** When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved.
		4		**Graphics Driver Status (R0)** When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		**Autonomous Utilization-Based Frequency Control Status (R0)** When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		**Core Power Limiting Status (R0)** When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		**Package-Level Power Limiting PL1 Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		**Package-Level PL2 Power Limiting Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		12		**Max Turbo Limit Status (R0)** When set, frequency is reduced below the operating system request due to multi-core turbo limits.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		13		**Turbo Transition Attenuation Status (RO)** When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.
		15:14		**Reserved**
		16		**PROCHOT Log** When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		**Thermal Log** When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		**Graphics Driver Log** When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		**Autonomous Utilization-Based Frequency Control Log** When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		**VR Therm Alert Log** When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		**Electrical Design Point Log** When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		**Core Power Limiting Log** When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		26		**Package-Level PL1 Power Limiting Log** When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		**Package-Level PL2 Power Limiting Log** When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		**Max Turbo Limit Log** When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		**Turbo Transition Attenuation Log** When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved.
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	**Indicator of Frequency Clipping in the Processor Graphics (R/W) (frequency refers to processor graphics frequency)**
		0		**PROCHOT Status (R0)** When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		**Thermal Status (R0)** When set, frequency is reduced below the operating system request due to a thermal event.
		3:2		Reserved.
		4		**Graphics Driver Status (R0)** When set, frequency is reduced below the operating system request due to Processor Graphics driver override.
		5		**Autonomous Utilization-Based Frequency Control Status (R0)** When set, frequency is reduced below the operating system request because the processor has detected that utilization is low
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		**Graphics Power Limiting Status (R0)** When set, frequency is reduced below the operating system request due to domain-level power limiting.
		10		**Package-Level Power Limiting PL1 Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		**Package-Level PL2 Power Limiting Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		**Reserved**
		16		**PROCHOT Log** When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		**Thermal Log** When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		**Graphics Driver Log** When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		**Autonomous Utilization-Based Frequency Control Log** When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		**VR Therm Alert Log** When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		24		**Electrical Design Point Log** When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		25		**Core Power Limiting Log** When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		**Package-Level PL1 Power Limiting Log** When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		**Package-Level PL2 Power Limiting Log** When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		**Max Turbo Limit Log** When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		**Turbo Transition Attenuation Log** When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved.
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	**Indicator of Frequency Clipping in the Ring Interconnect (R/W)** **(frequency refers to ring interconnect in the uncore)**
		0		**PROCHOT Status (R0)** When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		**Thermal Status (R0)** When set, frequency is reduced below the operating system request due to a thermal event.
		5:2		Reserved.
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		9		Reserved.
		10		**Package-Level Power Limiting PL1 Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL1.
		11		**Package-Level PL2 Power Limiting Status (R0)** When set, frequency is reduced below the operating system request due to package-level power limiting PL2.
		15:12		**Reserved**
		16		**PROCHOT Log** When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		17		**Thermal Log** When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		19:18		Reserved.
		20		**Graphics Driver Log** When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		21		**Autonomous Utilization-Based Frequency Control Log** When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		22		**VR Therm Alert Log** When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		23		Reserved.
		24		**Electrical Design Point Log** When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		25		**Core Power Limiting Log** When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		26		**Package-Level PL1 Power Limiting Log** When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		27		**Package-Level PL2 Power Limiting Log** When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		28		**Max Turbo Limit Log** When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		29		**Turbo Transition Attenuation Log** When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.
		63:30		Reserved.
700H	1792	MSR_UNC_CBO_0_ 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
720H	1824	MSR_UNC_CBO_2_ PERFEVTSEL0	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1824	MSR_UNC_CBO_2_ PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR

Table 35-25. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
See Table 35-16, Table 35-17, Table 35-20, Table 35-24 for other MSR definitions applicable to processors with CPUID signatures 063CH, 06_46H				

...

35.13 MSRS IN FUTURE GENERATION INTEL® XEON® PROCESSORS

The MSRs listed in Table 35-31 are available in Intel® Xeon® Processor D Product Family (CPUID DisplayFamily_DisplayModel = 06_56H). It is based on the Broadwell microarchitecture.

Table 35-31 also applies to future Intel Xeon processors based on the Broadwell microarchitecture (CPUID DisplayFamily_DisplayModel = 06_4FH).

...

35.14 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 35-16, Table 35-17, Table 35-20, Table 35-24, Table 35-30, and Table 35-32. For an MSR listed in Table 35-32 that also appears in the model-specific tables of prior generations, Table 35-32 supercede prior generation tables.

Table 35-32. 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)**

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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)**
		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.

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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.**

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
38EH	910	IA32_PERF_GLOBAL_ STAUS		See Table 35-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	**Ovf_PMC0**
		1	Thread	**Ovf_PMC1**
		2	Thread	**Ovf_PMC2**
		3	Thread	**Ovf_PMC3**
		4	Thread	**Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Thread	**Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)**
		6	Thread	**Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Thread	**Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Ovf_FixedCtr0**
		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_STAT US_RESET		See Table 35-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	**Set 1 to clear Ovf_PMC0**
		1	Thread	**Set 1 to clear Ovf_PMC1**
		2	Thread	**Set 1 to clear Ovf_PMC2**
		3	Thread	**Set 1 to clear Ovf_PMC3**
		4	Thread	**Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Thread	**Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)**
		6	Thread	**Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Thread	**Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		32	Thread	**Set 1 to clear Ovf_FixedCtr0**
		33	Thread	**Set 1 to clear Ovf_FixedCtr1**
		34	Thread	**Set 1 to clear Ovf_FixedCtr2**
		54:35		Reserved.
		55	Thread	**Set 1 to clear Trace_ToPA_PMI.**
		57:56		Reserved.
		58	Thread	**Set 1 to clear LBR_Frz.**
		59	Thread	**Set 1 to clear CTR_Frz.**
		60	Thread	**Set 1 to clear ASCI.**
		61	Thread	**Set 1 to clear Ovf_Uncore**
		62	Thread	**Set 1 to clear Ovf_BufDSSAVE**
		63	Thread	**Set 1 to clear CondChgd**
391H	913	IA32_PERF_GLOBAL_STATUS_SET		See Table 35-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	**Set 1 to cause Ovf_PMC0 = 1**
		1	Thread	**Set 1 to cause Ovf_PMC1 = 1**
		2	Thread	**Set 1 to cause Ovf_PMC2 = 1**
		3	Thread	**Set 1 to cause Ovf_PMC3 = 1**
		4	Thread	**Set 1 to cause Ovf_PMC4=1 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Thread	**Set 1 to cause Ovf_PMC5=1 (if CPUID.0AH:EAX[15:8] > 5)**
		6	Thread	**Set 1 to cause Ovf_PMC6=1 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Thread	**Set 1 to cause Ovf_PMC7=1 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Set 1 to cause Ovf_FixedCtr0 = 1**
		33	Thread	**Set 1 to cause Ovf_FixedCtr1 = 1**
		34	Thread	**Set 1 to cause Ovf_FixedCtr2 = 1**
		54:35		Reserved.
		55	Thread	**Set 1 to cause Trace_ToPA_PMI = 1**
		57:56		Reserved.
		58	Thread	**Set 1 to cause LBR_Frz = 1**
		59	Thread	**Set 1 to cause CTR_Frz = 1**
		60	Thread	**Set 1 to cause ASCI = 1**
		61	Thread	**Set 1 to cause Ovf_Uncore**

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		62	Thread	**Set 1 to cause Ovf_BufDSSAVE**
		63		**Reserved**
392H	913	IA32_PERF_GLOBAL_INUSE		See Table 35-2.
3F7H	1015	MSR_PEBS_FRONTEND	Thread	**FrontEnd Precise Event Condition Select (R/W)**
		2:0		**Event Code Select**
		3		Reserved.
		4		**Event Code Select High**
		7:5		Reserved.
		19:8		**IDQ_Bubble_Length Specifier**
		22:20		**IDQ_Bubble_Width Specifier**
		63:23		**Reserved**
500H	1280	IA32_SGX_SVN_STATUS	Thread	**Status and SVN Threshold of SGX Support for ACM (RO).**
		0		**Lock.** See Section 42.12.3, "Interactions with Authenticated Code Modules (ACMs)"
		15:1		Reserved.
		23:16		**SGX_SVN_SINIT.** See Section 42.12.3, "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**

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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	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
64EH	1615	MSR_PPERF	THREAD	Productive Performance Count. (R/O).
		63:0		Hardware's view of workload scalability. See Section 14.4.5.1
652H	1614	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	1615	MSR_CORE_HDC_ RESIDENCY	Core	Core HDC Idle Residency. (R/O).
		63:0		Core_Cx_Duty_Cycle_Cnt.
655H	1617	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).

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		Pkg_C2_Duty_Cycle_Cnt.
656H	1618	MSR_PKG_HDC_DEEP_ RESIDENCY	Package	Package Cx HDC Idle Residency. (R/O).
		63:0		Pkg_Cx_Duty_Cycle_Cnt.
658H	1620	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	1621	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	1622	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	1623	MSR_CORE_GFXE_OVERLAP_C0	Package	Core and Graphics Engine Overlapped C0 Residency. (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.
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 17.9
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.

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
697H	1687	MSR_ LASTBRANCH_23_FROM_IP	Thread	**Last Branch Record 23 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_ LASTBRANCH_24_FROM_IP	Thread	**Last Branch Record 24 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_ LASTBRANCH_25_FROM_IP	Thread	**Last Branch Record 25 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_ LASTBRANCH_26_FROM_IP	Thread	**Last Branch Record 26 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_ LASTBRANCH_27_FROM_IP	Thread	**Last Branch Record 27 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_ LASTBRANCH_28_FROM_IP	Thread	**Last Branch Record 28 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_ LASTBRANCH_29_FROM_IP	Thread	**Last Branch Record 29 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_ LASTBRANCH_30_FROM_IP	Thread	**Last Branch Record 30 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_ LASTBRANCH_31_FROM_IP	Thread	**Last Branch Record 31 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
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 17.9
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.

Table 35-32. Additional MSRs Supported by 6th Generation Intel® Core™ Processors Based on Skylake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6D7H	1751	MSR_ LASTBRANCH_23_TO_IP	Thread	**Last Branch Record 23 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_ LASTBRANCH_24_TO_IP	Thread	**Last Branch Record 24 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_ LASTBRANCH_25_TO_IP	Thread	**Last Branch Record 25 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_ LASTBRANCH_26_TO_IP	Thread	**Last Branch Record 26 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DBH	1755	MSR_ LASTBRANCH_27_TO_IP	Thread	**Last Branch Record 27 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_ LASTBRANCH_28_TO_IP	Thread	**Last Branch Record 28 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_ LASTBRANCH_29_TO_IP	Thread	**Last Branch Record 29 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
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"
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 35-32. 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 17.6.1, "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 35-32. 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.

…

35.15 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 35-33. 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 35-33. 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 35.20, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 35.20, "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 17.14, "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.
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.

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

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	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)**
		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)**

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 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.
		7:0		**Event Select**
		15:8		**UMask**
		16		**USR**
		17		**OS**
		18		**Edge**

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		19		**PC**
		20		**INT**
		21		**AnyThread**
		22		**EN**
		23		**INV**
		31:24		**CMASK**
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 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.
		22:16		**Digital Readout (RO)**
		26:23		Reserved.
		30:27		**Resolution in degrees Celsius (RO)**
		31		**Reading Valid (RO)**
		63:32		Reserved.

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
1A0H	416	IA32_MISC_ENABLE	Thread	**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0		**Fast-Strings Enable**
		2:1		Reserved.
		3		**Automatic Thermal Control Circuit Enable (R/W)**
		6:4		Reserved.
		7		**Performance Monitoring Available (R)**
		10:8		Reserved.
		11		**Branch Trace Storage Unavailable (RO)**
		12		**Precise 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)**
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.

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:8	Package	**Maximum Ratio Limit for Group 0** Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.
		20:16	Package	**Number of Incremental Cores Added to Group 1** Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".
		23:21	Package	**Group Ratio Delta for Group 1** An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.
		28:24	Package	**Number of Incremental Cores Added to Group 2** Group 2, which includes the specified number of additional cores plus all the cores in group 1, operates under the group 2 turbo max ratio limit = "group 1 Max ratio limit" - "group ratio delta for group 2".
		31:29	Package	**Group Ratio Delta for Group 2** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.
		36:32	Package	**Number of Incremental Cores Added to Group 3** Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".
		39:37	Package	**Group Ratio Delta for Group 3** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.
		44:40	Package	**Number of Incremental Cores Added to Group 4** Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".
		47:45	Package	**Group Ratio Delta for Group 4** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.
		52:48	Package	**Number of Incremental Cores Added to Group 5** Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		55:53	Package	**Group Ratio Delta for Group 5** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.
		60:56	Package	**Number of Incremental Cores Added to Group 6** Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".
		63:61	Package	**Group Ratio Delta for Group 6** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.
1B0H	432	IA32_ENERGY_PERF_BIAS	Thread	See Table 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.
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.

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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_A000 0	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_STAUS	Thread	See Table 35-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2.
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)

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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	MSR_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_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	MSR_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	MSR_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	MSR_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.

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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
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

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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."
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."

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 35-20
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O). See Table 35-20
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O). See Table 35-20
64BH	1611	MSR_CONFIG_TDP_ CONTROL	Package	**ConfigTDP Control (R/W)** See Table 35-20
64CH	1612	MSR_TURBO_ACTIVATION_ RATIO	Package	**ConfigTDP Control (R/W)** See Table 35-20
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.
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)

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request register bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request register bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request register bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI register (W/O)

Table 35-33. Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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

25. 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.

...

36.2.3.1 Packet Enable (PacketEn)

When PacketEn is set, the processor is in the mode that Intel PT is monitoring and all packets can be generated to log what is being executed. PacketEn is composed of other states according to this relationship:

```
PacketEn ← TriggerEn AND ContextEn AND FilterEn AND BranchEn
```

These constituent controls are detailed in the following subsections.

PacketEn ultimately determines when the processor is tracing. When PacketEn is set, all control flow packets are enabled. When PacketEn is clear, no control flow packets are generated, though other packets (timing and bookkeeping packets) may still be sent. See Section 36.2.4 for details of PacketEn and packet generation.

Note that, on processors that do not support IP filtering (i.e., CPUID.(EAX=14H, ECX=0):EBX.IPFILT_WRSTPRSV[bit 2] = 0), FilterEn is treated as always set.

...

36.2.3.3 Context Enable (ContextEn)

Context Enable (**ContextEn**) indicates whether the processor is in the state or mode that software configured hardware to trace. For example, if execution with CPL = 0 code is not being traced (IA32_RTIT_CTL.OS = 0), then ContextEn will be 0 when the processor is in CPL0.

Software can discover the current ContextEn value by reading the IA32_RTIT_STATUS.ContextEn bit. ContextEn is defined as follows:

```
ContextEn = !((IA32_RTIT_CTL.OS = 0 AND CPL = 0) OR
(IA32_RTIT_CTL.USER = 0 AND CPL > 0) OR (IS_IN_A_PRODUCTION_ENCLAVE[1]) OR
(IA32_RTIT_CTL.CR3Filter = 1 AND IA32_RTIT_CR3_MATCH does not match CR3)
```

If the clearing of ContextEn causes PacketEn to be cleared, a Packet Generation Disable (TIP.PGD) packet is generated, but its IP payload is suppressed. If the setting of ContextEn causes PacketEn to be set, a Packet Generation Enable (TIP.PGE) packet is generated.

When ContextEn is 0, control flow packets (TNT, FUP, TIP.*, MODE.*) are not generated, and no Linear Instruction Pointers (LIPs) are exposed. However, some packets, such as MTC and PSB (see Section 36.4.2.16 and Section 36.4.2.17), may still be generated while ContextEn is 0. For details of which packets are generated only when ContextEn is set, see Section 36.4.1.

The processor does not update ContextEn when TriggerEn = 0.

The value of ContextEn will toggle only when TriggerEn = 1.

...

36.2.3.5 Filter Enable (FilterEn)

Filter Enable indicates that the Instruction Pointer (IP) is within the range of IPs that Intel PT is configured to watch. Software can get the state of Filter Enable by a RDMSR of IA32_RTIT_STATUS.FilterEn. For details on configuration and use of IP filtering, see Section 36.2.2.3.

On clearing of FilterEn that also clears PacketEn, a Packet Generation Disable (TIP.PGD) will be generated, but unlike the ContextEn case, the IP payload may not be suppressed. For direct, unconditional branches, as well as for indirect branches (including RETs), the PGD generated by leaving the tracing region and clearing FilterEn will contain the target IP. This means that IPs from outside the configured range can be exposed in the trace, as long as they are within context.

When FilterEn is 0, control flow packets are not generated (e.g., TNT, TIP). However, some packets, such as PIP, MTC, and PSB, may still be generated while FilterEn is clear. For details on packet enable dependencies, see Section 36.4.1.

After TraceEn is set, FilterEn is set to 1 at all times if there is no IP filter range configured by software (IA32_RTIT_CTL.ADDRn_CFG != 1, for all n), or if the processor does not support IP filtering (i.e., CPUID.(EAX=14H, ECX=0):EBX.IPFILT_WRSTPRSV[bit 2] = 0). FilterEn will toggle only when TraceEn=1 and ContextEn=1, and when at least one range is configured for IP filtering.

...

36.2.4.2 Table of Physical Addresses (ToPA)

When IA32_RTIT_CTL.ToPA is set and IA32_RTIT_CTL.FabricEn is clear, the ToPA output mechanism is utilized. The ToPA mechanism uses a linked list of tables; see Figure 36-1 for an illustrative example. Each entry in the table contains some attribute bits, a pointer to an output region, and the size of the region. The last entry in the table may hold a pointer to the next table. This pointer can either point to the top of the current table (for circular

1. Trace packets generation is disabled in a production enclave, see Section 36.2.6.3. See *Intel® Software Guard Extensions Programming Reference* about differences between a production enclave and a debug enclave.

array) or to the base of another table. The table size is not fixed, since the link to the next table can exist at any entry.

The processor treats the various output regions referenced by the ToPA table(s) as a unified buffer. This means that a single packet may span the boundary between one output region and the next.

The ToPA mechanism is controlled by three values maintained by the processor:

- **proc_trace_table_base.**
  This is the physical address of the base of the current ToPA table. When tracing is enabled, the processor loads this value from the IA32_RTIT_OUTPUT_BASE MSR. While tracing is enabled, the processor updates the IA32_RTIT_OUTPUT_BASE MSR with changes to proc_trace_table_base, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_base.
- **proc_trace_table_offset.**
  This indicates the entry of the current table that is currently in use. (This entry contains the address of the current output region.) When tracing is enabled, the processor loads this value from bits 31:7 (MaskOrTableOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTableOffset with changes to proc_trace_table_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_offset.
- **proc_trace_output_offset.**
  This a pointer into the current output region and indicates the location of the next write. When tracing is enabled, the processor loads this value from bits 63:32 (OutputOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset with changes to proc_trace_output_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_output_offset.

Figure 36-1 provides an illustration (not to scale) of the table and associated pointers.

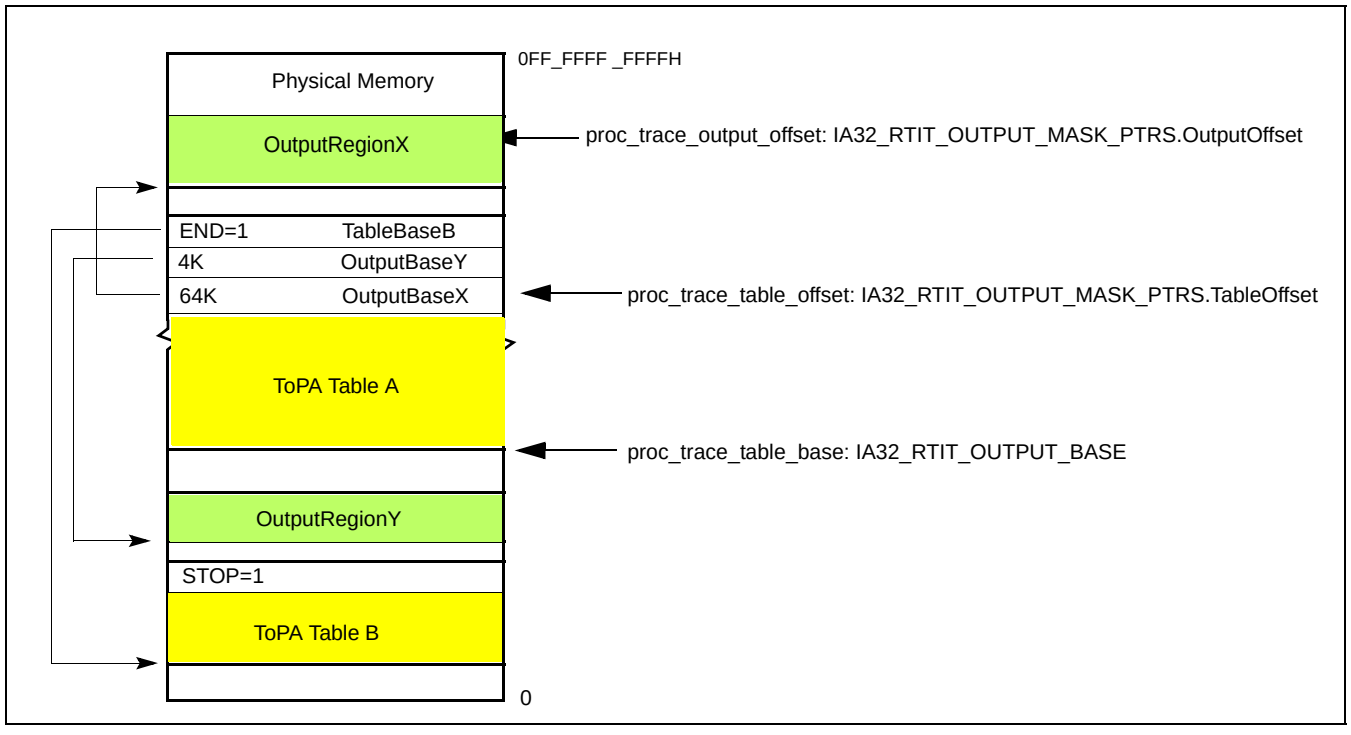


Figure 36-1. ToPA Memory Illustration

With the ToPA mechanism, the processor writes packets to the current output region (identified by proc_trace_table_base and the proc_trace_table_offset). The offset within that region to which the next byte will be written is identified by proc_trace_output_offset. When that region is filled with packet output (thus proc_trace_output_offset = RegionSize–1), proc_trace_table_offset is moved to the next ToPA entry, proc_trace_output_offset is set to 0, and packet writes begin filling the new output region specified by proc_trace_table_offset.

As packets are written out, each store derives its physical address as follows:

```
trace_store_phys_addr ← Base address from current ToPA table entry +
proc_trace_output_offset
```

Eventually, the regions represented by all entries in the table may become full, and the final entry of the table is reached. An entry can be identified as the final entry because it has either the END or STOP attribute. The END attribute indicates that the address in the entry does not point to another output region, but rather to another ToPA table. The STOP attribute indicates that tracing will be disabled once the corresponding region is filled. See Section 36.2.4.2 for details on STOP.

When an END entry is reached, the processor loads proc_trace_table_base with the base address held in this END entry, thereby moving the current table pointer to this new table. The proc_trace_table_offset is reset to 0, as is the proc_trace_output_offset, and packet writes will resume at the base address indicated in the first entry.

If the table has no STOP or END entry, and trace-packet generation remains enabled, eventually the maximum table size will be reached (proc_trace_table_offset = FFFFFFFFH). In this case, the proc_trace_table_offset and proc_trace_output_offset are reset to 0 (wrapping back to the beginning of the current table) once the last output region is filled.

It is important to note that processor updates to the IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs are asynchronous to instruction execution. Thus, reads of these MSRs while Intel PT is enabled may return stale values. Like all IA32_RTIT_* MSRs, the values of these MSRs should not be trusted or saved unless trace packet generation is first disabled by clearing IA32_RTIT_CTL.TraceEn. This ensures that he output MSR values account for all packets generated to that point, after which the output MSR values will be frozen until tracing resumes. [1]

The processor may cache internally any number of entries from the current table or from tables that it references (directly or indirectly). If tracing is enabled, the processor may ignore or delay detection of modifications to these tables. To ensure that table changes are detected by the processor in a predictable manner, software should clear TraceEn before modifying the current table (or tables that it references) and only then re-enable packet generation.

Single Output Region ToPA Implementation

The first processor generation to implement Intel PT supports only ToPA configurations with a single ToPA entry followed by an END entry that points back to the first entry (creating one circular output buffer). Such processors enumerate CPUID.(EAX=14H,ECX=0):ECX.MENTRY[bit 1] = 0 and CPUID.(EAX=14H,ECX=0):ECX.TOPAOUT[bit 0] = 1.

If CPUID.(EAX=14H,ECX=0):ECX.MENTRY[bit 1] = 0, ToPA tables can hold only one output entry, which must be followed by an END=1 entry which points back to the base of the table. Hence only one contiguous block can be used as output.

The lone output entry can have INT or STOP set, but nonetheless must be followed by an END entry as described above. Note that, if INT=1, the PMI will actually be delivered before the region is filled.

ToPA Table Entry Format

The format of ToPA table entries is shown in Figure 36-2. The size of the address field is determined by the processor's physical-address width (MAXPHYADDR) in bits, as reported in CPUID.80000008H:EAX[7:0].

1. Although WRMSR is a serializing instruction, the execution of WRMSR that forces packet writes by clearing TraceEn does not itself cause these writes to be globally observed.

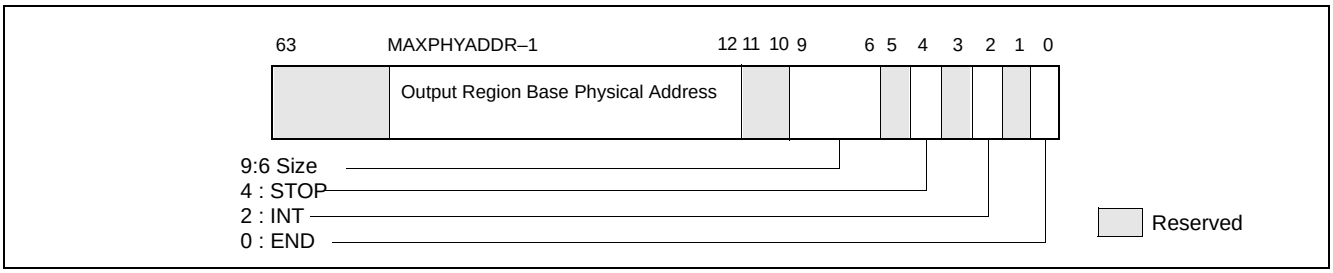


Figure 36-2. Layout of ToPA Table Entry

Table 36-3 describes the details of the ToPA table entry fields. If reserved bits are set to 1, an error is signaled.

Table 36-3. ToPA Table Entry Fields

ToPA Entry Field	Description
Output Region Base Physical Address	If END=0, this is the base physical address of the output region specified by this entry. Note that all regions must be aligned based on their size. Thus a 2M region must have bits 20:12 clear. If the region is not properly aligned, an operational error will be signaled when the entry is reached. If END=1, this is the 4K-aligned base physical address of the next ToPA table (which may be the base of the current table, or the first table in the linked list if a circular buffer is desired). If the processor supports only a single ToPA output region (see above), this address must be the value currently in the IA32_RTIT_OUTPUT_BASE MSR.
Size	Indicates the size of the associated output region. Encodings are: 0: 4K, 1: 8K, 2: 16K, 3: 32K, 4: 64K, 5: 128K, 6: 256K, 7: 512K, 8: 1M, 9: 2M, 10: 4M, 11: 8M, 12: 16M, 13: 32M, 14: 64M, 15: 128M This field is ignored if END=1.
STOP	When the output region indicated by this entry is filled, software should disable packet generation. This will be accomplished by setting IA32_RTIT_STATUS.Stopped, which clears TriggerEn. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
INT	When the output region indicated by this entry is filled, signal Perfmon LVT interrupt. Note that if both INT and STOP are set in the same entry, the STOP will happen before the INT. Thus the interrupt handler should expect that the IA32_RTIT_STATUS.Stopped bit will be set, and will need to be reset before tracing can be resumed. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
END	If set, indicates that this is an END entry, and thus the address field points to a table base rather than an output region base. If END=1, INT and STOP must be set to 0; otherwise it is treated as reserved bit violation (see ToPA Errors). The Size field is ignored in this case. If the processor supports only a single ToPA output region (see above), END must be set in the second table entry.

ToPA STOP

Each ToPA entry has a STOP bit. If this bit is set, the processor will set the IA32_RTIT_STATUS.Stopped bit when the corresponding trace output region is filled. This will clear TriggerEn and thereby cease packet generation. See Section 36.2.5.4 for details on IA32_RTIT_STATUS.Stopped. This sequence is known as “ToPA Stop”.

No TIP.PGD packet will be seen in the output when the ToPA stop occurs, since the disable happens only when the region is already full. When this occurs, output ceases after the last byte of the region is filled, which may mean that a packet is cut off in the middle. Any packets remaining in internal buffers are lost and cannot be recovered.

When ToPA stop occurs, the IA32_RTIT_OUTPUT_BASE MSR will hold the base address of the table whose entry had STOP=1. IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTableOffset will hold the index value for that entry, and the IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset should be set to the size of the region.

Note that this means the offset pointer is pointing to the next byte after the end of the region, a configuration that would produce an operational error if the configuration remained when tracing is re-enabled with IA32_RTIT_STATUS.Stopped cleared.

ToPA PMI

Each ToPA entry has an INT bit. If this bit is set, the processor will signal a performance-monitoring interrupt (PMI) when the corresponding trace output region is filled. This interrupt is not precise, and it is thus likely that writes to the next region will occur by the time the interrupt is taken.

The following steps should be taken to configure this interrupt:

1. Enable PMI via the LVT Performance Monitor register (at MMIO offset 340H in xAPIC mode; via MSR 834H in x2APIC mode). See *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B* for more details on this register. For ToPA PMI, set all fields to 0, save for the interrupt vector, which can be selected by software.
2. Set up an interrupt handler to service the interrupt vector that a ToPA PMI can raise.
3. Set the interrupt flag by executing STI.
4. Set the INT bit in the ToPA entry of interest and enable packet generation, using the ToPA output option. Thus, TraceEn=ToPA=1 in the IA32_RTIT_CTL MSR.

Once the INT region has been filled with packet output data, the interrupt will be signaled. This PMI can be distinguished from others by checking bit 55 (Trace_ToPA_PMI) of the IA32_PERF_GLOBAL_STATUS MSR (MSR 38EH). Once the ToPA PMI handler has serviced the relevant buffer, writing 1 to bit 55 of the MSR at 390H (IA32_GLOBAL_STATUS_RESET) clears IA32_PERF_GLOBAL_STATUS.Trace_ToPA_PMI.

Intel PT is not frozen on PMI, and thus the interrupt handler will be traced (though filtering can prevent this). The IA32_DEBUGCTL.Freeze_Perfmon_on_PMI setting will be applied on ToPA PMI just as on other PMIs, and hence Perfmon counters are frozen.

Assuming the PMI handler wishes to read any buffered packets for persistent output, software should first disable packet generation by clearing TraceEn. This ensures that all buffered packets are written to memory and avoids tracing of the PMI handler. The configuration MSRs can then be used to determine where tracing has stopped. If packet generation is disabled by the handler, it should then be manually re-enabled before the IRET if continued tracing is desired.

ToPA PMI and Single Output Region ToPA Implementation

A processor that supports only a single ToPA output region implementation (such that only one output region is supported; see above) will attempt to signal a ToPA PMI interrupt before the output wraps and overwrites the top of the buffer. To support this functionality, the PMI handler should disable packet generation as soon as possible.

Due to PMI skid, it is possible, in rare cases, that the wrap will have occurred before the PMI is delivered. Software can avoid this by setting the STOP bit in the ToPA entry (see Table 36-3); this will disable tracing once the region is filled, and no wrap will occur. This approach has the downside of disabling packet generation so that some of the instructions that led up to the PMI will not be traced. If the PMI skid is significant enough to cause the region to fill and tracing to be disabled, the PMI handler will need to clear the IA32_RTIT_STATUS.Stopped indication before tracing can resume.

ToPA PMI and XSAVES/XRSTORS State Handling

In some cases the ToPA PMI may be taken after completion of an XSAVES instruction that switches Intel PT state, and in such cases any modification of Intel PT MSRs within the PMI handler will not persist when the saved Intel PT context is later restored with XRSTORS. To account for such a scenario, it is recommended that the Intel PT output configuration be modified by altering the ToPA tables themselves, rather than the Intel PT output MSRs.

Table 36-4 depicts a recommended PMI handler algorithm for managing multi-region ToPA output and handling ToPA PMIs that may arrive between XSAVES and XRSTORS. This algorithm is flexible to allow software to choose between adding entries to the current ToPA table, adding a new ToPA table, or using the current ToPA table as a circular buffer. It assumes that the ToPA entry that triggers the PMI is not the last entry in the table, which is the recommended treatment.

Table 36-4. Algorithm to Manage Intel PT ToPA PMI and XSAVES/XRSTORS

Pseudo Code Flow

```
IF (IA32_PERF_GLOBAL_STATUS.ToPA)
    Save IA32_RTIT_CTL value;
    IF ( IA32_RTIT_CTL.TraceEN )
        Disable Intel PT by clearing TraceEn;
    FI;
    IF ( there is space available to grow the current ToPA table )
        Add one or more ToPA entries after the last entry in the ToPA table;
        Point new ToPA entry address field(s) to new output region base(s);
    ELSE
        Modify an upcoming ToPA entry in the current table to have END=1;
        IF (output should transition to a new ToPA table )
            Point the address of the "END=1" entry of the current table to the new table base;
        ELSE
            /* Continue to use the current ToPA table, make a circular. */
            Point the address of the "END=1"l entry to the base of the current table;
            Modify the ToPA entry address fields for filled output regions to point to new, unused output regions;
            /* Filled regions are those with index in the range of 0 to (IA32_RTIT_MASK_PTRS.MaskOrTableOffset -1). */
        FI;
      FI;
      Restore saved IA32_RTIT_CTL.value;
FI;
```

ToPA Errors

When a malformed ToPA entry is found, an **operation error** results (see Section 36.3.9). A malformed entry can be any of the following:

1. **ToPA entry reserved bit violation**.
   This describes cases where a bit marked as reserved in Section 36.2.4.2 above is set to 1.
2. **ToPA alignment violation**.
   This includes cases where illegal ToPA entry base address bits are set to 1:
   a. ToPA table base address is not 4KB-aligned. The table base can be from a WRMSR to IA32_RTIT_OUTPUT_BASE, or from a ToPA entry with END=1.
   b. ToPA entry base address is not aligned to the ToPA entry size (e.g., a 2MB region with base address[20:12] not equal to 0).
   c. ToPA entry base address sets upper physical address bits not supported by the processor.
3. **Illegal ToPA Output Offset** (if IA32_RTIT_STATUS.Stopped=0).
   IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset is greater than or equal to the size of the current ToPA output region size.
4. **ToPA rules violations**.
   These are similar to ToPA entry reserved bit violations; they are cases when a ToPA entry is encountered with illegal field combinations. They include the following:

a. Setting the STOP or INT bit on an entry with END=1.

b. Setting the END bit in entry 0 of a ToPA table.

c. On processors that support only a single ToPA entry (see above), two additional illegal settings apply:

   i) ToPA table entry 1 with END=0.

   ii) ToPA table entry 1 with base address not matching the table base.

In all cases, the error will be logged by setting IA32_RTIT_STATUS.Error, thereby disabling tracing when the problematic ToPA entry is reached (when proc_trace_table_offset points to the entry containing the error). Any packet bytes that are internally buffered when the error is detected may be lost.

Note that operational errors may also be signaled due to attempts to access restricted memory. See Section 36.2.4.4 for details.

A tracing software have a range of flexibility using ToPA to manage the interaction of Intel PT with application buffers, see Section 36.5.

...

36.2.4.3 Trace Transport Subsystem

When IA32_RTIT_CTL.FabricEn is set, the IA32_RTIT_CTL.ToPA bit is ignored, and trace output is written to the trace transport subsystem. The endpoints of this transport are platform-specific, and details of configuration options should refer to the specific platform documentation. The FabricEn bit is available to be set if CPUID(EAX=14H,ECX=0):EBX[bit 3] = 1.

...

36.2.5.1 General Considerations

Trace packet generation is enabled and configured by a collection of model-specific registers (MSRs), which are detailed below. Some notes on the configuration MSR behavior:

- If Intel Processor Trace is not supported by the processor (see Section 36.3.1), RDMSR or WRMSR of the IA32_RTIT_* MSRs will cause #GP.
- A WRMSR to any of these configuration MSRs that begins and ends with IA32_RTIT_CTL.TraceEn set will #GP fault. Packet generation must be disabled before the configuration MSRs can be changed.

  Note: Software may write the same value back to IA32_RTIT_CTL without #GP, even if TraceEn=1.
- All configuration MSRs for Intel PT are duplicated per logical processor
- For each configuration MSR, any MSR write that attempts to change bits marked reserved, or utilize encodings marked reserved, will cause a #GP fault.
- All configuration MSRs for Intel PT are cleared on a cold RESET.
  - If CPUID.(EAX=14H, ECX=0):EBX.IPFILT_WRSTPRSV[bit 2] = 1, only the TraceEn bit is cleared on warm RESET; though this may have the impact of clearing other bits in IA32_RTIT_STATUS. Other MSR values of the trace configuration MSRs are preserved on warm RESET.
- The semantics of MSR writes to trace configuration MSRs in this chapter generally apply to explicit WRMSR to these registers, using VM-exit or VM-entry MSR load list to these MSRs, XRSTORS with requested feature bit map including XSAVE map component of state_8 (corresponding to IA32_XSS[bit 8]), and the write to IA32_RTIT_CTL.TraceEn by XSAVES (Section 36.3.5.2).

36.2.5.2 IA32_RTIT_CTL MSR

IA32_RTIT_CTL, at address 570H, is the primary enable and control MSR for trace packet generation. Bit positions are listed in Table 36-5.

Table 36-6. IA32_RTIT_CTL MSR

Position	Bit Name	At Reset	Bit Description
0	TraceEn	0	If 1, enables tracing; else tracing is disabled if 0. When this bit transitions from 1 to 0, all buffered packets are flushed out of internal buffers. A further store, fence, or architecturally serializing instruction may be required to ensure that packet data can be observed at the trace endpoint. See Section 36.2.5.3 for details of enabling and disabling packet generation. Note that the processor will clear this bit on #SMI (Section) and warm reset. Other MSR bits of IA32_RTIT_CTL (and other trace configuration MSRs) are not impacted by these events.
1	CYCEn	0	0: Disables CYC Packet (see Section 36.4.2.14). 1: Enables CYC Packet. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX.CPSB_CAM[bit 1] = 0.
2	OS	0	0: Packet generation is disabled when CPL = 0. 1: Packet generation may be enabled when CPL = 0.
3	User	0	0: Packet generation is disabled when CPL > 0. 1: Packet generation may be enabled when CPL > 0.
5:4	Reserved	0	Must be 0.
6	FabricEn	0	0: Trace output is directed to the memory subsystem, mechanism depends on IA32_RTIT_CTL.ToPA. 1: Trace output is directed to the trace transport subsystem, IA32_RTIT_CTL.ToPA is ignored. This bit is reserved if CPUID.(EAX=14H, ECX=0):ECX[bit 3] = 0.
7	CR3Filter	0	0: Disables CR3 filtering. 1: Enables CR3 filtering.
8	ToPA	0	0: Single-range output scheme enabled if CPUID.(EAX=14H, ECX=0):ECX.SNGLRGNOUT[bit 2] = 1 and IA32_RTIT_CTL.FabricEn=0. 1: ToPA output scheme enabled (see Section 36.2.4.2) if CPUID.(EAX=14H, ECX=0):ECX.TOPA[bit 0] = 1, and IA32_RTIT_CTL.FabricEn=0. Note: WRMSR to IA32_RTIT_CTL that sets TraceEn but clears this bit and FabricEn would cause #GP, if CPUID.(EAX=14H, ECX=0):ECX.SNGLRGNOUT[bit 2] = 0. WRMSR to IA32_RTIT_CTL that sets this bit causes #GP, if CPUID.(EAX=14H, ECX=0):ECX.TOPA[bit 0] = 0.
9	MTCEn	0	0: Disables MTC Packet (see Section 36.4.2.16). 1: Enables MTC Packet. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX.MTC[bit 3] = 0.
10	TSCEn	0	0: Disable TSC packets. 1: Enable TSC packets (see Section 36.4.2.11).
11	DisRETC	0	0: Enable RET compression. 1: Disable RET compression (see Section 36.2.1.2).
12	Reserved	0	Must be 0.

Table 36-6. IA32_RTIT_CTL MSR

Position	Bit Name	At Reset	Bit Description
13	BranchEn	0	0: Disable COFI-based packets. 1: Enable COFI-based packets: FUP, TIP, TIP.PGE, TIP.PGD, TNT, MODE.Exec, MODE.TSX. see Section 36.2.4 for details on BranchEn.
17:14	MTCFreq	0	Defines MTC packet Frequency, which is based on the core crystal clock, or Always Running Timer (ART). MTC will be sent each time the selected ART bit toggles. The following Encodings are defined: 0: ART(0), 1: ART(1), 2: ART(2), 3: ART(3), 4: ART(4), 5: ART(5), 6: ART(6), 7: ART(7), 8: ART(8), 9: ART(9), 10: ART(10), 11: ART(11), 12: ART(12), 13: ART(13), 14: ART(14), 15: ART(15) Software must use CPUID to query the supported encodings in the processor, see Section 36.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX.MTC[bit 3] = 0.
18	Reserved	0	Must be 0.
22:19	CycThresh	0	CYC packet threshold, see Section 36.3.6 for details. CYC packets will be sent with the first eligible packet after N cycles have passed since the last CYC packet. If CycThresh is 0 then N=0, otherwise N is defined as $2^{(CycThresh-1)}$. The following Encodings are defined: 0: 0, 1: 1, 2: 2, 3: 4, 4: 8, 5: 16, 6: 32, 7: 64, 8: 128, 9: 256, 10: 512, 11: 1024, 12: 2048, 13: 4096, 14: 8192, 15: 16384 Software must use CPUID to query the supported encodings in the processor, see Section 36.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX.CPSB_CAM[bit 1] = 0.
23	Reserved	0	Must be 0.
27:24	PSBFreq	0	Indicates the frequency of PSB packets. PSB packet frequency is based on the number of Intel PT packet bytes output, so this field allows the user to determine the increment of IA32_IA32_RTIT_STATUS.PacketByteCnt that should cause a PSB to be generated. Note that PSB insertion is not precise, but the average output bytes per PSB should approximate the SW selected period. The following Encodings are defined: 0: 2K, 1: 4K, 2: 8K, 3: 16K, 4: 32K, 5: 64K, 6: 128K, 7: 256K, 8: 512K, 9: 1M, 10: 2M, 11: 4M, 12: 8M, 13: 16M, 14: 32M, 15: 64M Software must use CPUID to query the supported encodings in the processor, see Section 36.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX.CPSB_CAM[bit 1] = 0.
31:28	Reserved	0	Must be 0.
35:32	ADDR0_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR0_A/B based on the following encodings: 0: ADDR0 range unused. 1: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.2.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See 4.2.8 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] >= 0.

Table 36-6. IA32_RTIT_CTL MSR

Position	Bit Name	At Reset	Bit Description
39:36	ADDR1_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR1_A/B based on the following encodings: 0: ADDR1 range unused. 1: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.2.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 36.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 2.
43:40	ADDR2_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR2_A/B based on the following encodings: 0: ADDR2 range unused. 1: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.2.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 36.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 3.
47:44	ADDR3_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR3_A/B based on the following encodings: 0: ADDR3 range unused. 1: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.2.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 36.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 4.
59:48	Reserved	0	Reserved only for future trace content enables, or address filtering configuration enables. Must be 0.
63:60	Reserved	0	Must be 0.

…

36.3.6 Cycle-Accurate Mode

Intel PT can be run in a cycle-accurate mode which enables CYC packets (see Section 36.4.2.14) that provide low-level information in the processor core clock domain. This cycle counter data in CYC packets can be used to compute IPC (Instructions Per Cycle), or to track wall-clock time on a fine-grain level.

To enable cycle-accurate mode packet generation, software should set IA32_RTIT_CTL.CYCEn=1. It is recommended that software also set TSCEn=1 anytime cycle-accurate mode is in use. With this, all CYC-eligible packets will be preceded by a CYC packet, the payload of which indicates the number of core clock cycles since the last CYC packet. In cases where multiple CYC-eligible packets are generated in a single cycle, only a single CYC will be generated before the CYC-eligible packets, otherwise each CYC-eligible packet will be preceded by its own CYC. The CYC-eligible packets are:

- TNT, TIP, TIP.PGE, TIP.PGD, MODE.EXEC, MODE.TSX, PIP, VMCS, OVF, MTC, TSC

TSC packets are generated when there is insufficient information to reconstruct wall-clock time, due to tracing being disabled (TriggerEn=0), or power down scenarios like a transition to a deep-sleep MWAIT C-state. In this case, the CYC that is generated along with the TSC will indicate the number of cycles actively tracing (those powered up, with TriggerEn=1) executed between the last CYC packet and the TSC packet. And hence the amount of time spent while tracing is inactive can be inferred from the difference in time between that expected based on the CYC value, and the actual time indicated by the TSC.

Additional CYC packets may be sent stand-alone, so that the processor can ensure that the decoder is aware of the number of cycles that have passed before the internal hardware counter wraps, or is reset due to other micro-architectural condition. There is no guarantee at what intervals these standalone CYC packets will be sent, except that they will be sent before the wrap occurs. An illustration is given below.

Example 36-1. An Illustrative CYC Packet Example

Time (cycles)	Instruction Snapshot	Generated Packets	Comment
x	call %eax	CYC(?), TIP	?Elapsed cycles from the previous CYC unknown
x + 2	call %ebx	CYC(2), TIP	1 byte CYC packet; 2 cycles elapsed from the previous CYC
x + 8	jnz Foo (not taken)	CYC(6)	1 byte CYC packet
x + 9	ret (compressed)		
x + 12	jnz Bar (taken)		
x + 16	ret (uncompressed)	TNT, CYC(8), TIP	1 byte CYC packet
x + 4111		CYC(4095)	2 byte CYC packet
x + 12305		CYC(8194)	3 byte CYC packet
x + 16332	mov cr3, %ebx	CYC(4027), PIP	2 byte CYC packet

…

Table 36-12. CPUID Leaf 14H, sub-leaf 1H Enumeration of Intel Processor Trace Capabilities

CPUID.(EAX=14H,ECX=1)		Name	Description Behavior
Register	Bits		
EAX	2:0	Number of Address Ranges	A non-zero value specifies the number ADDRn_CFG field supported in IA32_RTIT_CTL and the number of register pair IA32_RTIT_ADDRn_A/ IA32_RTIT_ADDRn_B supported for IP filtering and IP TraceStop. **NOTE:** Currently, no processors support more than 4 address ranges.
	15:3	Reserved	

Table 36-12. CPUID Leaf 14H, sub-leaf 1H Enumeration of Intel Processor Trace Capabilities

CPUID.(EAX=14H,ECX=1)		Name	Description Behavior
Register	Bits		
	31:16	Bitmap of supported MTC Period Encodings	The non-zero bit positions indicate the map of supported encoding values for the IA32_RTIT_CTL.MTCFreq field. This applies only if CPUID.(EAX=14H, ECX=0):EBX.MTC[bit 3] = 1 (MTC Packet generation is supported), otherwise the MTCFreq field is reserved to 0. Each bit position in this field represents 1 encoding value in the 4-bit MTCFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: MTCFreq can be assigned the associated encoding value. 0: MTCFreq cannot be assigned to the associated encoding value. A write to IA32_RTIT_CTLMTCFreq with unsupported encoding will cause #GP fault.
EBX	15:0	Bitmap of supported Cycle Threshold values	The non-zero bit positions indicate the map of supported encoding for the IA32_RTIT_CTL.CycThresh field. This applies only if CPUID.(EAX=14H, ECX=0):EBX.CPSB_CAM[bit 1] = 1 (Cycle-Accurate Mode is Supported), otherwise the CycThresh field is reserved to 0. See Section 36.2.5. Each bit position in this field represents 1 encoding value in the 4-bit CycThresh field (ie, bit 0 is associated with encoding value 0). For each bit: 1: CycThresh can be assigned the associated encoding value. 0: CycThresh cannot be assigned to the associated encoding value. A write to CycThresh with unsupported encoding will cause #GP fault.
	31:16	Bitmap of supported Configurable PSB Frequency encoding	The non-zero bit positions indicate the map of supported encoding for the IA32_RTIT_CTL.PSBFreq field. This applies only if CPUID.(EAX=14H, ECX=0):EBX.CPSB_CAM[bit 1] = 1 (Configurable PSB is supported), otherwise the PSBFreq field is reserved to 0. See Section 36.2.5. Each bit position in this field represents 1 encoding value in the 4-bit PSBFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: PSBFreq can be assigned the associated encoding value. 0: PSBFreq cannot be assigned to the associated encoding value. A write to PSBFreq with unsupported encoding will cause #GP fault.
ECX	31:0	Reserved	
EDX	31:0	Reserved	

...

36.3.1.1 Packet Decoding of RIP versus LIP

FUP, TIP, TIP.PGE, and TIP.PGE packets can contain an instruction pointer (IP) payload. On some processor generations, this payload will be an effective address (RIP), while on others this will be a linear address (LIP). In the former case, the payload is the offset from the current CS base address, while in the latter it is the sum of the offset and the CS base address (Note that in real mode, the CS base address is the value of CS<<4, while in protected mode the CS base address is the base linear address of the segment indicated by the CS register.). Which IP type is in use is indicated by enumeration (see CPUID.(EAX=14H, ECX=0):ECX.LIP[bit 31] in Table 36-10).

For software that executes while the CS base address is 0 (including all software executing in 64-bit mode), the difference is indistinguishable. A trace decoder must account for cases where the CS base address is not 0 and the resolved LIP will not be evident in a trace generated on a CPU that enumerates use of RIP. This is likely to cause problems when attempting to link the trace with the associated binaries.

Note that IP comparison logic, for IP filtering and TraceStop range calculation, is based on the same IP type as these IP packets. For processors that output RIP, the IP comparison mechanism is also based on RIP, and hence on those processors RIP values should be written to IA32_RTIT_ADDRn_[AB] MSRs. This can produce differing behavior if the same trace configuration setting is run on processors reporting different IP types, i.e. CPUID.(EAX=14H, ECX=0):ECX.LIP[bit 31]. Care should be taken to check CPUID when configuring IP filters.

...

36.3.8 Internal Buffer Overflow

In the rare circumstances when new packets need to be generated but the processor's dedicated internal buffers are all full, an "internal buffer overflow" occurs. On such an overflow packet generation ceases (as packets would need to enter the processor's internal buffer) until the overflow resolves. Once resolved, packet generation resumes.

When the buffer overflow is cleared, an OVF packet (Section 36.4.2.16) is generated, and the processor ensures that packets which follow the OVF are not compressed (IP compression or RET compression) against packets that were lost.

If IA32_RTIT_CTL.BranchEn = 1, the OVF packet will be followed by a FUP if the overflow resolves while PacketEn=1. If the overflow resolves while PacketEn = 0 no packet is generated, but a TIP.PGE will naturally be generated later, once PacketEn = 1. The payload of the FUP or TIP.PGE will be the Current IP of the first instruction upon which tracing resumes after the overflow is cleared. Between the OVF and following FUP or TIP.PGE, there may be packets that do not depend on PacketEn, such as timing packets. If the overflow resolves while PacketEn=0, other packets that are not dependent on PacketEn may come before the TIP.PGE.

...

36.4.2.6 Flow Update (FUP) Packet

Table 36-24. FUP Packet Definition

<table>
<tr><td>Name</td><td colspan="5">Flow Update (FUP) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="5">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td colspan="3">IPBytes</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td colspan="8">IP[7:0]</td></tr>
<tr><td>2</td><td colspan="8">IP[15:8]</td></tr>
<tr><td>3</td><td colspan="8">IP[23:16]</td></tr>
<tr><td>4</td><td colspan="8">IP[31:24]</td></tr>
<tr><td>5</td><td colspan="8">IP[39:32]</td></tr>
<tr><td>6</td><td colspan="8">IP[47:40]</td></tr>
<tr><td>7</td><td colspan="8">IP[55:48]</td></tr>
<tr><td>8</td><td colspan="8">IP[63:56]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn & ContextEn. (Typically depends on BranchEn and FilterEn as well, see Section 36.2.2 for details.)</td><td>Generation Scenario</td><td colspan="3">Asynchronous Events (interrupts, exceptions, INIT, SIPI, SMI, VM exit[1], #MC), XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, (EENTRY, EEXIT, ERESUME, EEE, AEX,)[2], INT 0, INT 3, INT n, a WRMSR that disables packet generation.</td></tr>
<tr><td>Description</td><td colspan="5">Provides the source address for asynchronous events, and some other instructions. Is never sent alone, always sent with an associated TIP or MODE packet, and potentially others.</td></tr>
<tr><td>Application</td><td colspan="5">FUP packets provide the IP to which they bind. However, they are never standalone, but are coupled with other packets.
In TSX cases, the FUP is immediately preceded by a MODE.TSX, which binds to the same IP. A TIP will follow only in the case of TSX aborts, see Section 36.4.2.8 for details.
Otherwise, FUPs are part of compound packet events (see Section 36.4.1). In these compound cases, the FUP provides the source IP for an instruction or event, while a following TIP (or TIP.PGD) uop will provide any destination IP. Other packets may be included in the compound event between the FUP and TIP.</td></tr>
</table>

NOTES:
1. If IA32_VMX_MISC[bit 14] reports 1.
2. If Intel Software Guard Extensions is supported.

FUP IP Payload

Flow Update Packet gives the source address of an instruction when it is needed. In general, branch instructions do not need a FUP, because the source address is clear from the disassembly. For asynchronous events, however, the source address cannot be inferred from the source, and hence a FUP will be sent. Table 36-24 illustrates cases where FUPs are sent, and which IP can be expected in those cases.

Table 36-24. FUP Cases and IP Payload

Event	Flow Update IP	Comment
External Interrupt, NMI/SMI, Traps, Machine Check (trap-like), INIT/SIPI	Address of next instruction (Next IP) that would have been executed	Functionally, this matches the LBR FROM field value and also the EIP value which is saved onto the stack.

Table 36-24. FUP Cases and IP Payload

Event	Flow Update IP	Comment
Exceptions/Faults, Machine check (fault-like)	Address of the instruction which took the exception/fault (Current IP)	This matches the similar functionality of LBR FROM field value and also the EIP value which is saved onto the stack.
Software Interrupt	Address of the software interrupt instruction (Current IP)	This matches the similar functionality of LBR FROM field value, but does not match the EIP value which is saved onto the stack (Next Linear Instruction Pointer - NLIP).
EENTER, EEXIT, ERESUME, Enclave Exiting Event (EEE), AEX[1]	Current IP of the instruction	This matches the LBR FROM field value and also the EIP value which is saved onto the stack.
XACQUIRE	Address of the X* instruction	
XRELEASE, XBEGIN, XEND, XABORT, other transactional abort	Current IP	
#SMI	IP that is saved into SMRAM	
WRMSR that clears TraceEn	Current IP	

NOTES:

1. Information on EENTER, EEXIT, ERESUME, EEE, Asynchronous Enclave eXit (AEX) can be found in *Intel® Software Guard Extensions Programming Reference*.

On a canonical fault due to sequentially fetching an instruction in non-canonical space (as opposed to jumping to non-canonical space), the IP of the fault (and thus the payload of the FUP) will be a non-canonical address. This is consistent with what is pushed on the stack for such faulting cases.

If there are post-commit task switch faults, the IP value of the FUP will be the original IP when the task switch started. This is the same value as would be seen in the LBR_FROM field. But it is a different value as is saved on the stack or VMCS.

...

36.4.2.12 Mini Time Counter (MTC) Packet

Table 36-33. MTC Packet Definition

<table>
<tr><td>Name</td><td colspan="3">Mini time Counter (MTC) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3"><table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td colspan="8">CTC[N+7:N]</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.MTCEn && TriggerEn</td><td>Generation Scenario</td><td>Periodic, based on the core crystal clock, or Always Running Timer (ART).</td></tr>
<tr><td>Description</td><td colspan="3">When enabled by software, an MTC packet provides a periodic indication of wall-clock time. The 8-bit CTC (Common Timestamp Copy) payload value is set to (ART >> N) & 0xFF. The frequency of the ART is related to the Maximum Non-Turbo frequency, and the ratio can be determined from CPUID leaf 15H, as described in Section 36.8.3.
Software can select the threshold N, which determines the MTC frequency by setting the IA32_RTIT_CTL.MTCFreq field (see Section 36.2.5.2) to a supported value using the lookup enumerated by CPUID (see Section 36.3.1).
See Section 36.8.3 for details on how to use the MTC payload to track TSC time.
MTC provides 8 bits from the ART, starting with the bit selected by MTCFreq to dictate the frequency of the packet. Whenever that 8-bit range being watched changes, an MTC packet will be sent out with the new value of that 8-bit range. This allows the decoder to keep track of how much wall-clock time has elapsed since the last TSC packet was sent, by keeping track of how many MTC packets were sent and what their value was. The decoder can infer the truncated bits, CTC[N-1:0], are 0 at the time of the MTC packet.
There are cases in which MTC packet can be dropped, due to overflow or other micro-architectural conditions. The decoder should be able to recover from such cases by checking the 8-bit payload of the next MTC packet, to determine how many MTC packets were dropped. It is not expected that >256 consecutive MTC packets should ever be dropped.</td></tr>
<tr><td>Application</td><td colspan="3">MTC does not precisely indicate the time of any other packet, nor does it bind to any IP. However, all preceding packets represent instructions or events that executed before the indicated ART time, and all subsequent packets represent instructions that executed after, or at the same time as, the ART time.</td></tr>
</table>

…

36.4.2.19 Maintenance (MNT) Packet

Table 36-40. MNT Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Maintenance (MNT) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>3</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>4</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>5</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>6</td><td colspan="8">Payload[31:24]</td></tr>
<tr><td>7</td><td colspan="8">Payload[39:32]</td></tr>
<tr><td>8</td><td colspan="8">Payload[47:40]</td></tr>
<tr><td>9</td><td colspan="8">Payload[55:48]</td></tr>
<tr><td>10</td><td colspan="8">Payload[63:56]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">Implementation specific.</td></tr>
<tr><td>Description</td><td colspan="4">This packet is generated by hardware, the payload meaning is model-specific.</td></tr>
<tr><td>Application</td><td colspan="4">Unless a decoder has been extended for a particular family/model/stepping to interpret MNT packet payloads, this packet should simply be ignored. It does not bind to any IP.</td></tr>
</table>

…

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 in its default setting 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 36-43.

Table 36-44. 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.
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 no-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 non-Root operation and the absence of TSC adjustments in TSC packets generated in Root operation. The VMM can supply this information to the decoder.

...

36.5.2.5 Emulation of Intel PT Traced State

If a VMM emulates an element of processor state by taking a VM exit on reads and/or writes to that piece of state, and the state element impacts Intel PT packet generation or values, it may be incumbent upon the VMM to insert or modify the output trace data.

If a VM exit is taken on a guest write to CR3 (including "MOV CR3" as well as task switches), the PIP packet normally generated on the CR3 write will be missing.

To avoid decoder confusion when the guest trace is decoded, the VMM should emulate the missing PIP by writing it into the guest output buffer. If the guest CR3 value is manipulated, the VMM may also need to manipulate the IA32_RTIT_CR3_MATCH value, in order to ensure the trace behavior matches the guest's expectation.

Similarly, if a VMM emulates the TSC value by taking a VM exit on RDTSC, the TSC packets generated in the trace may mismatch the TSC values returned by the VMM on RDTSC. To ensure that the trace can be properly aligned

with software logs based on RDTSC, the VMM should either make corresponding modifications to the TSC packet values in the guest trace, or use mechanisms such as TSC offsetting or TSC scaling in place of exiting.

36.5.2.6 TSC Scaling

When TSC scaling is enabled for a guest using Intel PT, the VMM should ensure that the value of Maximum Non-Turbo Ratio[15:8] in MSR_PLATFORM_INFO (MSR 0CEH) and the TSC/"core crystal clock" ratio (EBX/EAX) in CPUID leaf 15H are set in a manner consistent with the resulting TSC rate that will be visible to the VM. This will allow the decoder to properly apply TSC packets, MTC packets (based on the core crystal clock or ART, whose frequency is indicated by CPUID leaf 15H), and CBR packets (which indicate the ratio of the processor frequency to the Max Non-Turbo frequency). Absent this, or separate indication of the scaling factor, the decoder will be unable to properly track time in the trace. See Section 36.8.3 for details on tracking time within an Intel PT trace.

...

36.8.3.1 Time Domain Relationships

The three domains are related by the following formula:

```
TimeStampValue = (CoreCrystalClockValue * P) + AdjustedProcessorCycles + Software_Offset;
```

The CoreCrystalClockValue can provide the coarse-grained component of the TSC value. P, or the TSC/"core crystal clock" ratio, can be derived from CPUID leaf 15H, as described in Section 36.8.3.

The AdjustedProcessorCycles component provides the fine-grained distance from the rising edge of the last core crystal clock. Specifically, it is a cycle count in the same frequency as the timestamp counter from the last crystal clock rising edge. The value is adjusted based on the ratio of the processor core clock frequency to the Maximum Non-Turbo (or P1) frequency.

The Software_Offsets component includes software offsets that are factored into the timestamp value, such as IA32_TSC_ADJUST.

36.8.3.2 Estimating TSC within Intel PT

For many usages, it may be useful to have an estimated timestamp value for all points in the trace. The formula provided in Section 36.8.3.1 above provides the framework for how such an estimate can be calculated from the various timing packets present in the trace.

The TSC packet provides the precise timestamp value at the time it is generated; however, TSC packets are infrequent, and estimates of the current timestamp value based purely on TSC packets are likely to be very inaccurate for this reason. In order to get more precise timing information between TSC packets, CYC packets and/or MTC packets should be enabled.

MTC packets provide incremental updates of the CoreCrystalClockValue. On processors that support CPUID leaf 15H, the frequency of the timestamp counter and the core crystal clock is fixed, thus MTC packets provide a means to update the running timestamp estimate. Between two MTC packets A and B, the number of crystal clock cycles passed is calculated from the 8-bit payloads of respective MTC packets:

(CTC_B - CTC_A), where CTC_i = MTC_i[15:8] << IA32_RTIT_CTL.MTCFreq and i = A, B.

The time from a TSC packet to the subsequent MTC packet can be calculated using the TMA packet that follows the TSC packet. The TMA packet provides both the crystal clock value (lower 16 bits, in the CTC field) and the AdjustedProcessorCycles value (in the FastCounter field) that can be used in the calculation of the corresponding core crystal clock value of the TSC packet.

When the next MTC after a pair of TSC/TMA is seen, the number of crystal clocks passed since the TSC packet can be calculated by subtracting the TMA.CTC value from the time indicated by the MTC_{Next} packet by

CTC_{Delta}[15:0] = (CTC_{Next}[15:0] - TMA.CTC[15:0]), where CTC_{Next} = $MTC_{Payload}$ << IA32_RTIT_CTL.MTCFreq.

The TMA.FastCounter field provides the fractional component of the TSC packet into the next crystal clock cycle.

CYC packets can provide further precision of an estimated timestamp value to many non-timing packets, by providing an indication of the time passed between other timing packets (MTCs or TSCs).

When enabled, CYC packets are sent preceding each CYC-eligible packet, and provide the number of processor core clock cycles that have passed since the last CYC packet. Thus between MTCs and TSCs, the accumulated CYC values can be used to estimate the adjusted_processor_cycles component of the timestamp value. The accumulated CPU cycles will have to be adjusted to account for the difference in frequency between the processor core clock and the P1 frequency. The necessary adjustment can be estimated using the core:bus ratio value given in the CBR packet, by multiplying the accumulated cycle count value by $P1/CBR_{payload}$.

A greater level of precision may be achieved by calculating the CPU clock frequency, see Section 36.8.3.4 below for a method to do so using Intel PT packets.

CYCs can be used to estimate time between TSCs even without MTCs, though this will likely result in a reduction in estimated TSC precision.

36.8.3.3 VMX TSC Manipulation

When software executes in non-Root operation, additional offset and scaling factors may be applied to the TSC value. These are optional, but may be enabled via VMCS controls on a per-VM basis. See Chapter 25, "VMX Non-Root Operation" for details on VMX TSC offsetting and TSC scaling.

Like the value returned by RDTSC, TSC packets will include these adjustments, but other timing packets (such as MTC, CYC, and CBR) are not impacted. In order to use the algorithm above to estimate the TSC value when TSC scaling is in use, it will be necessary for software to account for the scaling factor. See Section 36.5.2.6 for details.

...

26. New Chapter 37, New Volume 3D

A new chapter, Chapter 37, has been added to the new volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

CHAPTER 37
INTRODUCTION TO INTEL® SOFTWARE GUARD EXTENSIONS

37.1 OVERVIEW

This document describes the Intel® Software Guard Extensions (Intel® SGX), a set of instructions and mechanisms for memory accesses added to future Intel® Architecture processors. Intel SGX can encompass two collections of instruction extensions, referred to as SGX1 and SGX2, see Table 37-4. The SGX1 extensions allow an application to instantiate a protected container, referred to as an enclave. An enclave is a protected area in the application's address space (see Figure 37-1), which provides confidentiality and integrity even in the presence of privileged malware. Accesses to the enclave memory area from any software not resident in the enclave are prevented. The SGX2 extensions allow additional flexibility in runtime management of enclave resources and thread execution within an enclave.

Chapter 38 covers main concepts, objects and data structure formats that interact within the Intel SGX architecture. Chapter 39 covers operational aspects ranging from preparing an enclave, transferring control to enclave code, and programming considerations for the enclave code and system software providing support for enclave execution. Chapter 40 describes the behavior of Asynchronous Enclave Exit (AEX) caused by events while executing enclave code. Chapter 41 covers the syntax and operational details of the instruction and as-

sociated leaf functions available in Intel SGX. Chapter 42 describes interaction of various aspects of IA32 and Intel® 64 architectures with Intel SGX. Chapter 43 covers Intel SGX support for application debug, profiling and performance monitoring.

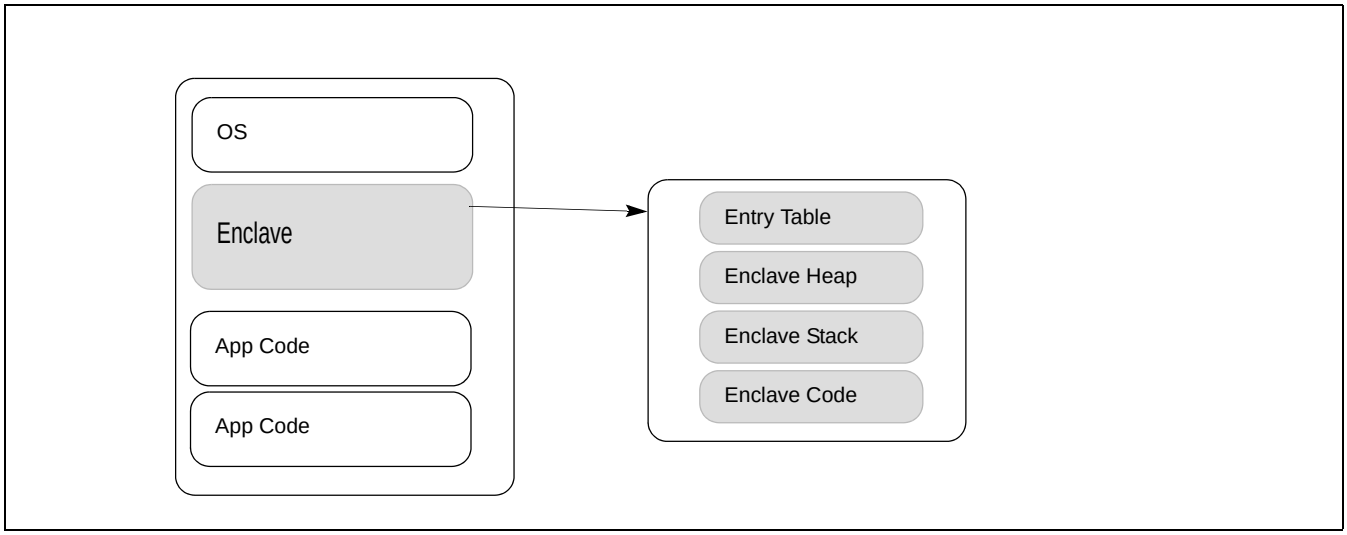


Figure 37-1. An Enclave Within the Application's Virtual Address Space

37.2 ENCLAVE INTERACTION AND PROTECTION

Intel SGX allows the protected portion of an application to be distributed in the clear. Before the enclave is built, the enclave code and data are free for inspection and analysis. The protected portion is loaded into an enclave where its code and data is measured. Once the application's protected portion of the code and data are loaded into an enclave, it is protected against external software access. An enclave can prove its identity to a remote party and provide the necessary building-blocks for secure provisioning of keys and credentials. The application can also request an enclave-specific and platform-specific key that it can use to protect keys and data that it wishes to store outside the enclave.[1]

Intel SGX introduces two significant capabilities to the Intel Architecture. First is the change in enclave memory access semantics. The second is protection of the address mappings of the application.

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 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.

Proper memory management procedure for enclave memory access or non-enclave memory access are required throughout the life cycle of an enclave: from creation, use, to destruction.

1. For additional information, see white papers on Intel SGX at http://software.intel.com/en-us/intel-isa-extensions.

During enclave creation, code and data for an enclave are loaded from a clear-text source, i.e. from non-enclave memory.

Un-trusted application code starts using an initialized enclave typically by using the Intel SGX EENTER instruction to transfer control to the enclave code residing in protected enclave page cache (EPC). The enclave code returns to the caller via the EEXIT instruction. 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 the stack pointer then executes the EEXIT instruction.

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 also be done using the EEXIT instruction. EEXIT and a software convention between the trusted section and the un-trusted section.

An active enclave consumes available resource from the EPC. Intel SGX provides the EREMOVE instruction that an EPC manager can use to reclaim resources committed to an enclave no longer in use. The EPC manager uses EREMOVE on every page. After execution of EREMOVE the 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) and the Thread Control Structure (TCS).

There is one SECS for each enclave. The SECS contains meta-data which is used by the hardware to protect the enclave. Included in the SECS is a field which 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. The SECS cannot be accessed by software.

Every enclave contains one or more TCSs. 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, which may be accessed by software.

The SECS is created at the time 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 a secure storage used by the processor to store enclave pages when they are a part of an executing enclave.

The EPC is divided into chunks of 4KB pages. An EPC page is always aligned on a 4KB boundary.

EPC is used to hold pages belonging to instance of enclaves. 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 one EPC page holding its SECS. The security metadata for each EPC page are held in an internal micro-architecture structure called Enclave Page Cache Map (EPCM).

The EPC is a platform asset and as such must be managed by privileged software. Intel SGX provides a set of instructions for adding and removing content to and from the EPC. The EPC is typically configured by BIOS at boot time. On implementations in which EPC is part of system DRAM, the contents of the EPC are protected by an encryption engine.

37.5.1 Enclave Page Cache Map (EPCM)

The EPCM is a secure structure used by the processor to track the contents of the EPC. The EPCM holds one entry for each page in the EPC. The format of the EPCM is micro-architectural, and consequently is implementation dependent. However, the EPCM contains the following architectural information to hardware:

- The status of EPC page with respect to validity and accessibility.
- The enclave instance that owns the page. SECS identifier of the enclave to which the page belongs.
- The type of page: regular, SECS, TCS or VA.
- The linear address through which the enclave is allowed to access the page.
- The specified read/write/execute permissions on that page.

The EPCM structure is used by the CPU in the address-translation flow to enforce access-control on the enclave pages loaded into the EPC. The EPCM structure is described in Table 38-27, and the conceptual access-control flow is described in Section 38.5.

The EPCM entries are managed by the processor as part of various instruction flows.

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 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 also used interchangeably in this document for brevity.

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 a page	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 by debugger	ENCLU[EREPORT]	Create a cryptographic report
ENCLS[EDBGWR]	Write data 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 as blocked		
ENCLS[ELDU]	Load an EPC page as unblocked		
ENCLS[EPA]	Add version array		
ENCLS[EREMOVE]	Remove a page from EPC		
ENCLS[ETRACK]	Activate EBLOCK checks		
ENCLS[EWB]	Write back/invalidate an EPC page		

Table 37-2. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX2

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EAUG]	Allocate page to an existing enclave	ENCLU[EACCEPT]	Accept changes to a page
ENCLS[EMODPR]	Restrict page permissions	ENCLU[EMODPE]	Enhance access rights
ENCLS[EMODT]	Make page TCS	ENCLU[EACCEPTCOPY]	Copy page to a new location

37.7 DISCOVERING SUPPORT FOR INTEL® SGX AND ENABLING ENCLAVE INSTRUCTIONS

Detection of support of Intel SGX and enumeration of available and enabled Intel SGX resources are queried using the CPUID instruction. The enumeration interface comprises the following:

- Processor support of Intel SGX is enumerated by a feature flag in CPUID leaf 07H: CPUID.(EAX=07H, ECX=0H):EBX.SGX[bit 2]. If CPUID.(EAX=07H, ECX=0H):EBX.SGX = 1, the processor has support for Intel SGX, and requires opt-in enabling by BIOS via IA32_FEATURE_CONTROL MSR.

  If CPUID.(EAX=07H, ECX=0H):EBX.SGX = 1, CPUID will report via the available sub-leaves of CPUID.(EAX=12H) on available and/or configured Intel SGX resources.
- The available and configured Intel SGX resources enumerated by the sub-leaves of CPUID.(EAX=12H) depend on the state of opt-in configuration by BIOS.

37.7.1 Intel® SGX Opt-In Configuration

On processors that support Intel SGX, IA32_FEATURE_CONTROL provides the SGX_ENABLE field (bit 18). Before system software can configure and enable Intel SGX resources, BIOS is required to set IA32_FEATURE_CONTROL.SGX_ENABLE = 1 to opt-in the use of Intel SGX by system software.

The semantics of setting SGX_ENABLE follows the rules of IA32_FEATURE_CONTROL.LOCK (bit 0). Software is considered to have opted into Intel SGX if and only if IA32_FEATURE_CONTROL.SGX_ENABLE and IA32_FEATURE_CONTROL.LOCK are set to 1. The setting of IA32_FEATURE_CONTROL.SGX_ENABLE (bit 18) is not reflected by CPUID.

Table 37-3. Intel® SGX Opt-in and Enabling Behavior

CPUID.(07H,0H):EBX.SGX	CPUID.(12H)	FEATURE_CONTROL.LOCK	FEATURE_CONTROL.SGX_ENABLE	Enclave Instruction
0	Invalid	X	X	#UD
1	Valid*	X	X	#UD
1	Valid*	0	X	#GP
1	Valid*	1	0	#GP
1	Valid*	1	1	Available (see Table 37-4 for details of SGX1 and SGX2).
* Leaf 12H enumeration results are dependent on enablement.				

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 opcodes of SGX1 instruction listed in Table 37-1 are supported.
	1	SGX2: If 1, indicates opcodes 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 enclave size is 2^(EDX[7:0]) byes when not in 64-bit mode.
	15:8	MaxEnclaveSize_64: the maximum enclave size is 2^(EDX[15:8]) byes 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 bit fields of bits [31:0] of SECS.ATTRIBUTES that software can set with ECREATE.
EBX	31:0	Report the valid bit fields of bits [63:32] of SECS.ATTRIBUTES that software can set with ECREATE.
ECX	31:0	Report the valid bit fields of bits [95:64] of SECS.ATTRIBUTES that software can set with ECREATE.
EDX	31:0	Report the valid bit fields of bits [127:96] of SECS.ATTRIBUTES that software can set with ECREATE.

CPUID leaf 12H sub-leaves 2 and higher report physical memory resources available for use with Intel SGX. These physical memory sections are typically configured 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, EBX:EAX and EDX:ECX report 0. 0001b: This sub-leaf provides information on the Enclave Page Cache (EPC) in EBX:EAX and EDX:ECX. All other encoding are reserved.
	11:4	Reserved (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 (0).
ECX	3: 0	0000b: Not valid. 0001b: The EPC section is confidentiality, integrity and replay protected. All other encoding are reserved.
	11:4	Reserved (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 (0).

...

27. New Chapter 38, New Volume 3D

A new chapter, Chapter 38, has been added to the new volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

CHAPTER 38
ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

38.1 OVERVIEW OF ENCLAVE EXECUTION ENVIRONMENT

An enclave comprises a contiguous range in the linear address space. An enclave must run from a special area of physical memory called Enclave Page Cache (EPC), which is protected from "non-enclave" memory accesses. An enclave need not be physically contiguous within the EPC. It is up to the EPC manager to allocate EPC pages to various enclaves. Enclaves 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, and the hardware guarantees that these pages will be mapped to EPC (any failure generates an exception).

38.2 TERMINOLOGY

A memory access that is initiated by internal enclave code to a linear address inside that enclave is called a Direct Enclave Access (Direct EA).

Memory accesses initiated by certain Intel® SGX instruction leaf functions such as ECREATE, EADD, EDBGRD, EDBGWR, ELDU/ELDB, EWB, EREMOVE, EENTER, and ERESUME that need to access EPC data by a non-enclave, managing context are called Indirect Enclave Accesses (Indirect EA). Table 38-1 lists additional details of the indirect EA of SGX1 and SGX2 extensions.

Direct EAs and Indirect EAs together are called Enclave Accesses (EAs). Intel SGX instruction leaves with indirect EA are listed in Table 38-1.

Any memory access that is not an Enclave Access is called a non-enclave access.

38.3 ACCESS-CONTROL REQUIREMENTS

Enclave accesses have the following access-control requirements:

- All memory accesses must conform to segmentation and paging policies set by the OS/VMM.
- Enclave entry/exit must happen through specific enclave instructions or events:
  - — ENCLU[EENTER], ENCLU[ERESUME].
  - — ENCLU[EEXIT], Asynchronous Enclave Exit (AEX).
- Direct jumps from outside an enclave to any linear address that maps to an enclave page do not enable enclave mode and result in abort page semantics and undefined behavior.
- 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 abort page semantics.
- Hardware detects and prevents enclave accesses using logical-to-linear address translations which are different than the original direct EA used to allocate the page. Detection of modified translation results in #GP(0).
- Direct EAs to any EPC pages must conform to the currently defined security attributes for that page. These attributes may be defined at enclave creation time (EADD) or when the enclave redefines them using SGX2 instructions:
  - — Target must belong to the same enclave.
  - — RWX attributes of the access must be compatible with the current RWX permissions.
  - — Target must not have a restricted page type (PT_SECS, PT_TCS or PT_VA, PT_TRIM).
  - — The EPC page must not be BLOCKED.
  - — The EPC page must not be PENDING.
  - — The EPC page must not be MODIFIED.

NOTE

For read accesses with abort-page semantics, see Section 6.5, "Exception Classifications," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. Write accesses with abort-page semantics are ignored.

38.4 SEGMENT-BASED ACCESS CONTROL

Intel SGX architecture does not modify the segment checks performed by a logical processor. All memory accesses arising from a logical processor in protected mode (including one that is inside an enclave) are subject to segmentation checks with the appropriate segment register.

To ensure that outside entities do not modify the enclave's logical-to-linear address translation in an unexpected fashion, ENCLU[EENTER] and ENCLU[ERESUME] check that CS, DS, ES, and SS, if usable (i.e., not null), have segment base value of zero. A non-zero segment base value for these registers results in a #GP(0).

On enclave entry either via EENTER or ERESUME, the processor saves the contents of the FS and GS registers, and modifies these registers to enable the enclave's use of these registers for accessing the thread-local storage inside the enclave. FS and GS are loaded from values stored in the TCS at build time. On enclave exit, the contents at time of entry are restored. The details of these modifications can be found in the descriptions of EENTER, ERESUME, EEXIT, and AEX flows.

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 afford enclaves a protected execution environment. Intel SGX provides page-granular access-control for enclave pages that are loaded into an EPC. Enclave pages loaded into an EPC are only accessible from inside the same enclave, or through certain Intel SGX instructions.

38.5.2 Memory Accesses that Split across ELRANGE

Memory data accesses are allowed to split across ELRANGE (i.e., a part of the access is inside ELRANGE and a part of the access is outside ELRANGE) while the processor is inside an enclave. If an access splits across ELRANGE, the processor splits the access into two sub-accesses (one inside ELRANGE and the other outside ELRANGE), and each access is evaluated. A code-fetch access that splits across ELRANGE results in a #GP due to the portion that lies outside of the ELRANGE.

38.5.3 Implicit vs. Explicit Accesses

Memory accesses originating from Intel SGX instruction leaf functions are categorized as either explicit accesses or implicit accesses. Table 38-1 lists the implicit and explicit memory accesses made by Intel SGX leaf functions.

38.5.3.1 Explicit Accesses

Accesses to memory locations provided as explicit operands to Intel SGX instruction leaf functions, or their linked data structures are called explicit accesses.

Explicit accesses are always made using logical addresses. These accesses are subject to segmentation, paging, extended paging, and APIC-virtualization checks, and trigger any faults/exit associated with these checks when the access is made.

The interaction of explicit memory accesses with data breakpoints is leaf-function-specific, and is documented in Section 43.3.5.

38.5.3.2 Implicit Accesses

Accesses to data structures whose physical addresses are cached by the processor are called implicit accesses. These accesses are not passed as operands of the instruction but are implied by use of the instruction.

Implicit accesses are made using physical addresses that are cached by the processor. 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]. This binding is severed when the corresponding page is removed from the EPC via ENCLS[EREMOVE]. Physical addresses of TCS and SSA memory 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 used for implicit accesses are derived from logical (or linear) addresses using ordinary address translation. Before caching such a physical address the original logical (or linear) address is subject to ordinary 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	Explicit 1	Explicit 2	Explicit 3	Implicit
ECREATE	PAGEINFO and linked structures	EPCPAGE		
EADD	PAGEINFO and linked structures	EPCPAGE		
EEXTEND	EPCPAGE			SECS
EINIT	SIGSTRUCT	SECS	EINITTOKEN	
EBLOCK	EPCPAGE			SECS
ETRACK	EPCPAGE			
ELDB/ELDU	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	
EWB	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	SECS
EREMOVE	EPCPAGE			SECS
EDBGRD	EPCADDR	Destination		SECS
EDBGWR	EPCADDR	Source		SECS
EENTER	TCS and linked SSA			SECS
ERESUME	TCS and linked SSA			SECS
EGETKEY	KEYREQUEST	KEY		SECS
EREPORT	TARGETINFO	REPORTDATA	OUTPUTDATA	SECS
EEXIT				SECS, TCS
EPA	EPCADDR			
EAUG	PAGEINFO and linked structures	EPCPAGE		SECS
EMODPE	SECINFO	EPCPAGE		
EMODPR	SECINFO	EPCPAGE		SECS
EMODT	SECINFO	EPCPAGE		SECS
EACCEPT	SECINFO	EPCPAGE		SECS

Table 38-1. List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions

Instr. Leaf	Explicit 1	Explicit 2	Explicit 3	Implicit
EACCEPTCOPY	SECINFO	EPCPAGE (Src)	EPCPAGE (Dst)	
Asynchronous Enclave Exit*				SECS, TCS, SSA
*Details of Asynchronous Enclave Exit (AEX) is described in Section 40.4				

38.6 INTEL® SGX DATA STRUCTURES OVERVIEW

Enclave operation is managed via a collection of data structures, many of the top-level data structures contain sub-structures. The top-level data structures relate to parameters that may be used in enclave setup/maintenance, by Intel SGX instructions, or AEX event. The top-level data structures are:

- SGX Enclave Control Structure (SECS)
- Thread Control Structure (TCS)
- State Save Area (SSA)
- Page Information (PageInfo)
- Security Information (SECINFO)
- Paging Crypto MetaData (PCMD)
- Enclave Signature Structure (SIGSTRUCT)
- EINIT Token Structure (EINITTOKEN)
- Report Structure (REPORT)
- Report Target Info (TARGETINFO)
- Key Request (KEYREQUEST)
- Version Array (VA)
- Enclave Page Cache Map (EPCM)

Details of the top-level data structures and associated sub-structures are listed in Section 38.7 through Section 38.19.

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 conditionally MISC).
MISCSELECT	20	4	Bit vector specifying which extended features are saved to the MISC region 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 proper format.
RESERVED	96	32	
MRSIGNER	128	32	Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for proper format.
RESERVED	160	96	
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
EID	Implementation dependent	8	Enclave Identifier.
PADDING	Implementation dependent	352	Padding pattern from the Signature (used for key derivation strings).
RESERVED	260	3836	Includes EID, other non-zero reserved field and must-be-zero fields.

38.7.1 ATTRIBUTES

The ATTRIBUTES data structure is comprised of bit-granular fields that are used in the SECS, CPUID enumeration, the REPORT and the KEYREQUEST structures.

Table 38-3. Layout of ATTRIBUTES Structure

Field	Bit Position	Description
RESERVED	0	
DEBUG	1	If 1, the enclave permit debugger to read and write data to enclave.
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

If CPUID.(EAX=12H, ECX=0):EBX[31:0] != 0, the processor can save extended information into the MISC region of SSA when an AEX occurs. An enclave writer can specify in the SECS.MISCSELECT field with the bit vector representing which extended information are 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 page fault and general protection exception info 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
RESERVED	0	8	
FLAGS	8	8	The thread's execution flags.
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.
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 beginning of the enclave.
RESERVED	40	8	
OFSBASGX	48	8	When added to the base address of the enclave, produces the base address FS segment inside the enclave. Must be page aligned.
OGSBASGX	56	8	When added to the base address of the enclave, produces the base address 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.

38.8.1 TCS.FLAGS

Table 38-6. Layout of TCS.FLAGS Field

Field	Bit Position	Description
DBGOPTIN	0	If set, enables debugging features (TF, breakpoints, etc.) while executing in the enclave on this TCS. Hardware clears this bit. A debugger may later modify it.
RESERVED	63:1	

38.8.2 State Save Area Offset (OSSA)

The OSSA points to a stack of state save area frames used to save the processor state when an interrupt or exception occurs while executing in the enclave. Each frame in the stack consists of the XSAVE region starting at the

base of a state save area frame. The GPRSGX region is top-aligned to the end of the frame. Each frame must be 4KBytes aligned and multiples of 4KBytes in size. Enclave writer can choose the pad size between the XSAVE region and the MISC region. A MISC region contains additional information written by the processor is next below the GPRSGX region inside the frame.

38.8.3 Number of State Save Area Frames (NSSA)

NSSA specifies the number of SSA frames available for this TCS. There must be at least one available SSA frame when EENTER-ing the enclave or the EENTER will fail.

38.8.4 Current State Save Area Frame (CSSA)

CSSA is the index of the current SSA frame that will be used by the processor to determine where to save the processor state on an interrupt or exception that occurs while executing in the enclave. It is an index into the array of frames addressed by OSSA. CSSA is incremented on an AEX and decremented on an ERESUME.

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.
- The GPRSGX region. 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 GPRSGX region is flush-aligned within the end of an SSA frame.
- 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 are pads that software can use. If the MISC region is present, the region between the MISC and XSAVE regions are pads that software can use.

  One or more components of extended information may be written to the MISC region if one or more bits in CPUID.(EAX=12H, ECX=0):EBX[31:0] are set. The component written to the MISC region is determined by the set bits in SECS.MISCSELECT.

Table 38-7. Top-to-Bottom Layout of an SSA Frame

Region	Offset (Byte)	Size (Bytes)	Description
GPRSGX	SSAFRAMESIZE -1 77	176	See Table 38-8 for layout of the GPRSGX region.
MISC	base of GPRSGX -sizeof(MISC)	Calculate from highest set bit of SECS.MISCSELECT	See Section 38.9.2.
Pad	End of XSAVE region	Chosen by enclave writer	Ensure the end of GPRSGX region is aligned to the end of a 4KB page.
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.

38.9.1 GPRSGX Region

The layout of the GPRSGX region is shown in Table 38-8.

Table 38-8. Layout of GPRSGX Portion of the State Save Area

Field	OFFSET (Bytes)	Size (Bytes)	Description
RAX	0	8	
RCX	8	8	
RDX	16	8	
RBX	24	8	
RSP	32	8	
RBP	40	8	
RSI	48	8	
RDI	56	8	
R8	64	8	
R9	72	8	
R10	80	8	
R11	88	8	
R12	96	8	
R13	104	8	
R14	112	8	
R15	120	8	
RFLAGS	128	8	Flag register.
RIP	136	8	Instruction pointer.
URSP	144	8	Untrusted (outside) stack pointer. Saved by EENTER, restored on AEX.
URBP	152	8	Untrusted (outside) RBP pointer. Saved by EENTER, restored on AEX.
EXITINFO	160	4	Contains information about exceptions that cause AEXs, which might be needed by enclave software.
RESERVED	164	4	Padding to 8-byte alignment.
FSBASE	168	8	FS BASE.
GSBASE	176	8	GS BASE.

38.9.1.1 EXITINFO

EXITINFO contains the information used to report exit reasons to software inside the enclave. It is a 4 byte field laid out as in Table 38-9. The VALID bit is set only for the exceptions conditions which are reported inside an enclave. See Table 38-10 for which exceptions are reported inside the enclave. If the exception condition is not one reported inside the enclave then VECTOR and EXIT_TYPE are cleared.

Table 38-9. Layout of EXITINFO Field

Field	Bit Position	Description
VECTOR	7:0	Exception number of exceptions reported inside enclave.
EXIT_TYPE	10:8	011b: Hardware exceptions. 110b: Software exceptions. Other values: Reserved.
RESERVED	30:11	Reserved as zero.
VALID	31	0: unsupported exceptions. 1: Supported exceptions. Includes two categories: • Unconditionally supported exceptions: #DE, #DB, #BP, #BR, #UD, #MF, #AC, #XM. • Conditionally supported exception: — #PF, #GP if SECS.MISCSELECT.EXINFO = 1.

38.9.1.2 VECTOR Field Definition

Table 38-10 contains the VECTOR field. This field contains information about some exceptions which occur inside the enclave. These vector values are the same as the values that would be used when vectoring into regular exception handlers. All values not shown are not reported inside an enclave.

Table 38-10. Exception Vectors

Name	Vector #	Description
#DE	0	DIV and IDIV instructions.
#DB	1	For Intel use only.
#BP	3	INT 3 instruction.
#BR	5	BOUND instruction.
#UD	6	UD2 instruction and reserved opcodes.
#GP	13	General protection violation. Only reported if SECS.MISCSELECT.EXINFO = 1.
#PF	14	Page fault. Only reported if SECS.MISCSELECT.EXINFO = 1.
#MF	16	x87 FPU floating-point or WAIT/FWAIT instruction.
#AC	17	Any data reference in memory.
#XM	19	Any SIMD floating-point exceptions.

38.9.2 MISC Region

The layout of the MISC region is shown in Table 38-11. The number of components available in the MISC region corresponds to the set bits of CPUID.(EAX=12H, ECX=0):EBX[31:0]. 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 must consult both CPUID.(EAX=12H, ECX=0):EBX[31:0], SECS.MISCSELECT, and Offset/Size information of Table 38-11 to determine the size of the MISC region. The first component, EXINFO, starts below the base of the GPRSGX region. Additional components in the MISC region grow downward within the MISC 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 the highest bit set in SECS.MISCSELECT in conjunction with the offset and size information defined in Table 38-11. For example, if the highest bit set in SECS.MISCSELECT is bit 0, the MISC region size is OFFSET(EXINFO) + Sizeof(EXINFO).

Table 38-11. Layout of MISC region of the State Save Area

MISC Components	OFFSET (Bytes)	Size (Bytes)	Description
EXINFO	base(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)

38.9.2.1 EXINFO Structure

Table 38-12 contains the layout of the EXINFO structure that provides additional information.

Table 38-12. Layout of EXINFO Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MADDR	0	8	Page fault address (unused for #GP).
ERRCD	8	4	Exception error code for either #GP or #PF.
RESERVED	12	4	

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 in Intel Architecture.
W/R	1	Same as non-SGX page fault exception W/R flag.
U/S	2	Always set to 1 (user mode reference).
RSVD	3	Reserved.
I/D	4	Same as non-SGX page fault exception I/D flag.
RSVD	31:5	Reserved.

38.10 PAGE INFORMATION (PAGEINFO)

PAGEINFO is an architectural data structure that is used as a parameter to the EPC-management instructions. It requires 32-Byte alignment.

Table 38-14. Layout of PAGEINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
LINADDR	0	8	Enclave linear address.
SRCPGE	8	8	Effective address of the page where page contents are located.
SECINFO/PCMD	16	8	Effective address of the SECINFO or PCMD (for ELDU, ELDB, EWB) structure for the page.
SECS	24	8	Effective address of EPC slot that currently contains a copy of the SECS.

38.11 SECURITY INFORMATION (SECINFO)

The SECINFO data structure holds meta-data about an enclave page.

Table 38-15. Layout of SECINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
FLAGS	0	8	Flags describing the state of the enclave page; R/W by software.
RESERVED	8	56	Must be zero.

38.11.1 SECINFO.FLAGS

The SECINFO.FLAGS are a set of bits describing the properties of an enclave page.

Table 38-16. Layout of SECINFO.FLAGS Field

Field	Bit Position	Description
R	0	If 1 indicates that the page can be read from inside the enclave; otherwise the page cannot be read from inside the enclave.
W	1	If 1 indicates that the page can be written from inside the enclave; otherwise the page cannot be written from inside the enclave.
X	2	If 1 indicates that the page can be executed from inside the enclave; otherwise the page cannot be executed from inside the enclave.
PENDING	3	If 1 indicates that the page is in the PENDING state; otherwise the page is not in the PENDING state.
MODIFIED	4	If 1 indicates that the page is in the MODIFIED state; otherwise the page is not in the MODIFIED state.
RESERVED	7:5	Must be zero.
PAGE_TYPE	15:8	The type of page that the SECINFO is associated with.
RESERVED	63:16	Must be zero.

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.
PT_REG	2	Page is a normal 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 writes out the reserved field and MAC values. 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	ENCLAVEID.
RESERVED	72	40	Must be zero.
MAC	112	16	MAC for the page, page meta-data and reserved field.

38.13 ENCLAVE SIGNATURE STRUCTURE (SIGSTRUCT)

SIGSTRUCT contains information about the enclave from the enclave signer, and must be 4K-Bytes aligned.

SIGSTRUCT includes ENCLAVEHASH as SHA256 digests as defined in FIPS PUB 180-4. The digests are byte strings of length 32 with the most significant byte of each of the 8 HASH dwords at the left most byte position.

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.

Table 38-19. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
HEADER	0	16	Must be byte stream 06000000E10000000000010000000000H.	Y
VENDOR	16	4	Intel Enclave: 00008086H Non-Intel Enclave: 00000000H.	Y
DATE	20	4	Build date is yyyymmdd in hex: yyyy=4 digit year, mm=1-12, dd=1-31.	Y
HEADER2	24	16	Must be byte stream 01010000600000006000000001000000H.	Y
SWDEFINED	40	4	Available for software use.	Y
RESERVED	44	84	Must be zero.	Y
MODULUS	128	384	Module Public Key (keylength=3072 bits).	N
EXPONENT	512	4	RSA Exponent = 3.	N
SIGNATURE	516	384	Signature over Header and Body.	N
MISCSELECT*	900	4	Bit vector specifying Extended SSA frame feature set to be used.	Y
MISCMASK*	904	4	Bit vector mask of MISCSELECT to enforce.	Y
RESERVED	908	20	Must be zero.	Y
ATTRIBUTES	928	16	Enclave Attributes that must be set.	Y
ATTRIBUTEMASK	944	16	Mask of Attributes to enforce.	Y
ENCLAVEHASH	960	32	MRENCLAVE of enclave this structure applies to.	Y
RESERVED	992	32	Must be zero.	Y
ISVPRODID	1024	2	ISV assigned Product ID.	Y
ISVSVN	1026	2	ISV assigned SVN (security version number).	Y
RESERVED	1028	12	Must be zero.	N
Q1	1040	384	Q1 value for RSA Signature Verification.	N
Q2	1424	384	Q2 value for RSA Signature Verification.	N
* If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISCSELECT must be 0. If CPUID.(EAX=12H, ECX=0):EBX[31:0] !=0, enclave writers must specify MISCSELECT such that each cleared bit in MISCMASK must also specify the corresponding bit as 0 in MISCSELECT.				

38.14 EINIT TOKEN STRUCTURE (EINITTOKEN)

The EINIT token is used by EINIT to verify that the enclave is permitted to launch.

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	Bits 0: 1: Valid; 0: Debug. All other bits reserved.
RESERVED	4	44	Y	Must be zero.

Table 38-20. Layout of EINIT Token (EINITTOKEN)

Field	OFFSET (Bytes)	Size (Bytes)	MACed	Description
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		MASKEDMISCSELECT of Launch Enclave. This should be set to the LE's MASKEDMISCSELECT masked with MISCMASK of the LE's KEYREQUEST.
MASKEDATTRIBUTESLE	240	16	N	MASKEDATTRIBUTES of Launch Enclave. 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	A cryptographic MAC on EINITTOKEN using Launch key.

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	SSA Frame specified extended feature set bit vector.
RESERVED	20	28	Must be zero.
ATTRIBUTES	48	16	The values of the attributes flags for the enclave. See Section 38.7.1 (ATTRIBUTES Bits) for the definitions of these flags.
MRENCLAVE	64	32	The value of SECS.MRENCLAVE.
RESERVED	96	32	Reserved.
MRSIGNER	128	32	The value of SECS.MRSIGNER.
RESERVED	160	96	Zero.
ISVPRODID	256	02	Enclave PRODUCT ID.
ISVSVN	258	02	The security version number of the Enclave.
RESERVED	260	60	Zero.
REPORTDATA	320	64	A set of data used for communication between the enclave and the target enclave.This value is provided by the EREPORT call in RCX.
KEYID	384	32	Value for key wear-out protection.
MAC	416	16	The CMAC on the report using report key.

38.15.1 REPORTDATA

The REPORTDATA structure is specifies the address of a 64-Byte input buffer that the EREPORT instruction will use to generate cryptographic report. It requires 128-Byte alignment.

38.16 REPORT TARGET INFO (TARGETINFO)

This structure is an input parameter to the EREPORT instruction leaf. The address of TARGETINFO is specified as an effective address in RBX. It is used to identify the enclave which will be able to cryptographically verify the REPORT structure returned by EREPORT. A TARGETINFO requires 512-Byte alignment.

Table 38-22. Layout of TARGETINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MEASUREMENT	0	32	The MRENCLAVE of the target enclave.
ATTRIBUTES	32	16	The ATTRIBUTES field of the target enclave.
RESERVED	48	4	
MISCSELECT	52	4	SSA Frame extended feature set bit vector.
RESERVED	56	456	

38.17 KEY REQUEST (KEYREQUEST)

This structure is an input parameter to the EGETKEY instruction. It is passed in as an effective address in RBX and requires 512-Byte alignment. It is used for selecting the appropriate key and any additional parameters required in the derivation of that key.

Table 38-23. Layout of KEYREQUEST Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	02	Identifies the Key Required.
KEYPOLICY	02	02	Identifies which inputs are required to be used in the key derivation.
ISVSVN	04	02	The ISV security version number used in the key derivation.
RESERVED	06	02	Must be zero.
CPUSVN	08	16	The security version number of the processor used in the key derivation.
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in the derivation of the Seal Key.
KEYID	40	32	Value for key wear-out protection.
MISCMASK	72	4	A mask defining which MISCSELECT bits will be included in the derivation of the Seal Key.
RESERVED	76	436	

38.17.1 KEY REQUEST KeyNames

Table 38-24. Supported KEY Name Values

Key Name	Value	Description
LAUNCH_KEY	0	Launch key.
PROVISION_KEY	1	Provisioning Key.
PROVISION_SEAL_KEY	2	Provisioning Seal Key.
REPORT_KEY	3	Report Key.
SEAL_KEY	4	Report Key.
	All other	Reserved.

38.17.2 Key Request Policy Structure

Table 38-25. Layout of KEYPOLICY Field

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register.
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register.
RESERVED	15:2	Must be zero.

38.18 VERSION ARRAY (VA)

In order to securely store the versions of evicted EPC pages, Intel SGX defines a special EPC page type called a Version Array (VA). Each VA page contains 512 slots, each of which can contain an 8-byte version number for a page evicted from the EPC. When an EPC page is evicted, software chooses an empty slot in a VA page; this slot receives the unique version number of the page being evicted. When the EPC page is reloaded, a VA slot must hold the version of the page. If the page is successfully reloaded, the version in the VA slot is cleared.

VA pages can be evicted, just like any other EPC page. When evicting a VA page, a version slot in some other VA page must be used to receive the version for the VA being evicted. A Version Array Page must be 4K-Bytes aligned.

Table 38-26. Layout of Version Array Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
Slot 0	0	08	Version Slot 0
Slot 1	8	08	Version Slot 1
…			
Slot 511	4088	08	Version Slot 511

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 are allowed for the EPC page.
W	Write access; indicates whether enclave accesses are allowed for the EPC page.
X	Execute access; indicates whether enclave accesses are allowed for the EPC page.
PT	EPCM page type (PT_SECS, PT_TCS, PT_REG, PT_VA, PT_TRIM).
ENCLAVESECS	SECS identifier of the enclave to which the page belongs.
ENCLAVEADDRESS	Linear enclave address of the page.
BLOCKED	Indicates whether the page is in the blocked state.
PENDING	Indicates whether the page is in the pending state.
MODIFIED	Indicates whether the page is in the modified state.

...

28. New Chapter 39, New Volume 3D

A new chapter, Chapter 39, has been added to the new volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

CHAPTER 39
ENCLAVE OPERATION

The following aspects of enclave operation are described in this chapter:

- Enclave creation: Includes loading code and data from outside of enclave into the EPC and establishing the enclave entity.
- Adding pages and measuring the enclave.
- Initialization of an enclave: Finalizes the cryptographic log and establishes the enclave identity and sealing identity.
- Enclave entry and exiting including
  - — Synchronous 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.

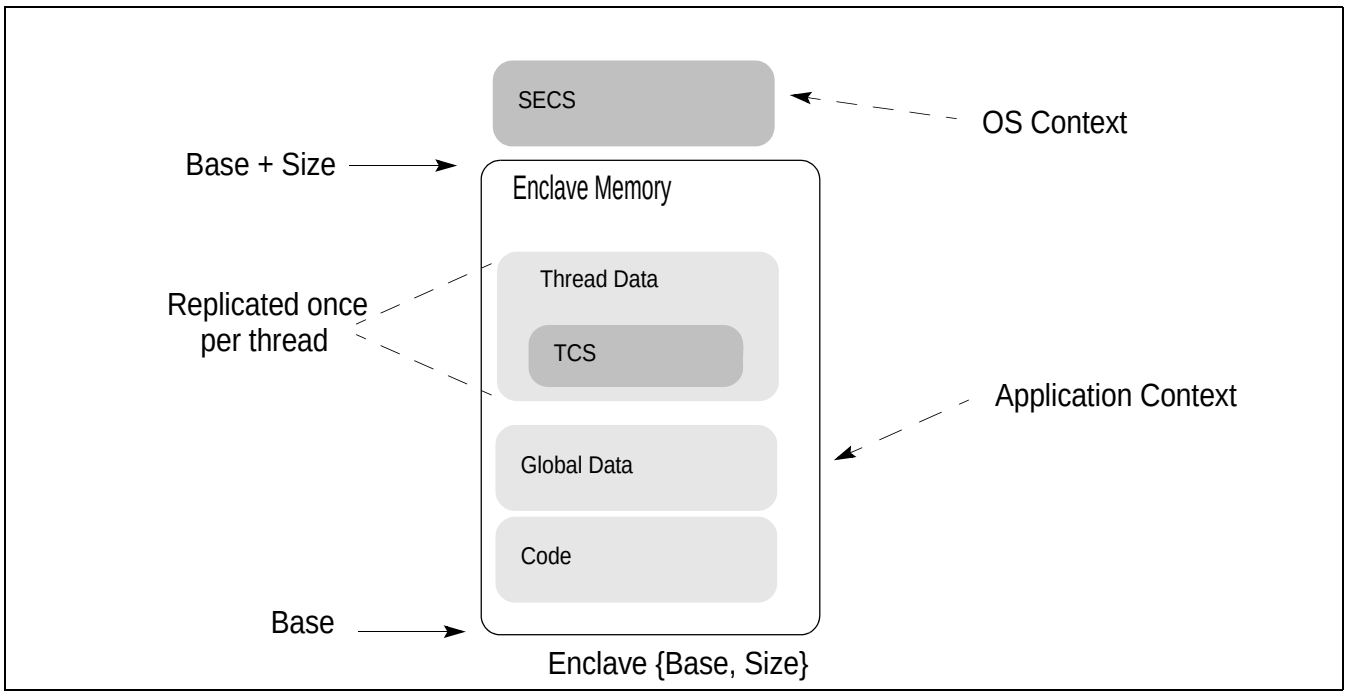


Figure 39-1. Enclave Memory Layout

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:

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 ring-0.
2. Use the ECREATE leaf 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. Use the EADD instruction leaf to commit EPC pages to the enclave, and use EEXTEND to measure the committed memory content of the enclave. For each additional page to be added to the enclave:
   - Use EADD to add the new page to the enclave.
   - If the enclave developer requires measurement, use EEXTEND to add a measurement for 256 bytes of the page. Repeat this operation until the entire page is measured.
4. Use the EINIT instruction leaf 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).

39.1.1 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 initialize the EPCM entry to add the type of page (PT_REG or PT_TCS), the linear address used by the enclave to access the page, and the enclave RWX 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 a cryptographic log stored in the SECS and copy 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.

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 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 un-trusted system software.

Examples of incorrect construction includes adding multiple pages with the same enclave linear address resulting in an alias, loading modified contents into an enclave page, or not measuring all of the enclave.

39.1.2 EINIT Interaction

Once system software has completed the process of adding and measuring pages, the enclave needs to be initialized by the EINIT instruction. Initializing an enclave prevents the addition or measurement of enclave pages and enables enclave entry. 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. Correct construction results in the cryptographic log matching the one built by the enclave owner in SIGSTRUCT. It can be verified by a remote party.

The enclave is initialized by the EINIT instruction. The EINIT instruction checks the ENIT 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 then EINIT will fail. See the EINIT instruction for details on the error reporting.

The enclave identity is a cryptographic hash that reflects the content of the enclave, the order in which it was built, the addresses it occupies in memory, and the security attributes of each page. The Enclave Identity is established by EINIT.

The sealing identity is managed by a sealing authority represented by the hash of a public key used to sign a structure processed by EINIT. The sealing authority assigns a product ID and security version number to a particular enclave identity comprising the attributes of the enclave and the measurement of the enclave.

EINIT establishes the sealing identity using the following steps:

1. Verifies that SIGSTRUCT is 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 are compatible with those specified in SIGSTRUCT.

4. Finalizes the measurement of the enclave and records the sealing identity and enclave identity (the sealing authority, product id and security version number) in the SECS.

39.2 ENCLAVE ENTRY AND EXITING

39.2.1 Synchronous Entry and Exit

The EENTER instruction 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 area inside the enclave an AEX should store the register state.

When a logical processor enters an enclave, the TCS is considered busy until the logical processors exits the enclave. Intel® SGX allows an enclave builder to define multiple TCSs, thereby providing support for multi-threaded enclaves.

EENTER also defines the Asynchronous Exit Pointer (AEP) parameter. AEP is an address external to the enclave which is used to transition back into the enclave after an AEX. The AEP is the address 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 TLB entries for enclave linear addresses in the enclave's ELRANGE.
2. Change the mode of operation to be in enclave mode.
3. Save the RSP, RBP for later restore on AEX.
4. Save XCR0 and replace it with the XFRM value for the enclave.
5. Check if the enclave is debuggable and the software wishes to debug. If not then set hardware so the enclave appears as a single instruction.
6. If the enclave is debuggable and the software wishes to debug, then set hardware to allow traps, breakpoints, and single steps inside the enclave.
7. Set the TCS as busy.
8. Transfer control from outside enclave to predetermined location inside the enclave specified by the TCS.

The EEXIT instruction is the method of leaving the enclave under program control, it performs the following operations:

1. Clear enclave mode and TLB entries for enclave addresses.
2. Mark TCS as not busy.
3. Transfer control from inside the enclave to a location on the outside specified by the register, RBX.

It is the responsibility of enclave software to erase any secret from the registers prior to invoking EEXIT.

39.2.2 Asynchronous Enclave Exit (AEX)

Asynchronous and synchronous events, such as exceptions, interrupts, SMIs, and VM exits may occur while executing inside an enclave. These events are referred to as Enclave Exiting Events (EEE). Upon an EEE, the processor state is securely saved inside the enclave (in the thread's current SSA frame) and then replaced by a synthetic state to prevent leakage of secrets. The process of securely saving state and establishing the synthetic state is called an Asynchronous Enclave Exit (AEX).

As part of most EEEs, the AEP is pushed onto the stack as the location of the eventing address. This is the location where control will return to after executing the IRET. The ERESUME can be executed from that point to reenter the enclave and resume execution from the interrupted point.

After AEX has completed, the logical processor is no longer in enclave mode and the exiting event is processed normally. Any new events that occur after the AEX has completed are treated as having occurred outside the enclave (e.g. a #PF in dispatching to an interrupt handler).

39.2.3 Resuming Execution after AEX

After system software has serviced the event that caused the logical processor to exit an enclave, the logical processor can re-start execution using ERESUME. ERESUME restores registers and returns control to where execution was interrupted.

If the cause of the exit was an exception or a fault and was not resolved, the event will be triggered again if the enclave is re-entered using ERESUME. For example, if an enclave performs a divide by 0 operation, executing ERESUME will cause the enclave to attempt to re-execute the faulting instruction and result in another divide by 0 exception. In order to handle an exception that occurred inside the enclave, software can enter the enclave at a different location and invoke the exception handler within the enclave by executing the EENTER instruction. The exception handler within the enclave can attempt to resolve the faulting condition or simply return and indicate to software that the enclave should be terminated (e.g. using EEXIT).

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 plus REX.W = 1.

39.3 CALLING ENCLAVE PROCEDURES

39.3.1 Calling Convention

In standard call conventions subroutine parameters are generally pushed onto the stack. The called routine, being aware of its own stack layout, knows how to find parameters based on compile-time-computable offsets from the SP or BP register (depending on runtime conventions used by the compiler).

Because of the stack switch when calling an enclave, stack-located parameters cannot be found in this manner. Entering the enclave requires a modified parameter passing convention.

For example, the caller might push parameters onto the untrusted stack and then pass a pointer to those parameters in RAX to the enclave software. The exact choice of calling conventions is up to the writer of the edge routines; be those routines hand-coded or compiler generated.

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 to temporarily contain secrets.

39.3.3 Returning to Caller

No registers are modified during EEXIT. It is the responsibility of software to remove secrets in registers before executing EEXIT.

39.4 INTEL® SGX KEY AND ATTESTATION

To provide cryptographic separation between platforms, Intel SGX provides individual keys to each platform.

Each processor is provisioned with a unique key as the root of the key hierarchy. This is done at manufacturing time. This key is the basis for all keys derived in the EGETKEY instruction. Figure 39-2 shows the hierarchy used to generate keys on the platform.

Each enclave requests keys using the EGETKEY instruction. The key is based on enclave parameters such as measurement or the enclave signing key plus the key derived from the device key and various security version numbers (SVNs). See the EGETKEY instruction for more details.

In order for a remote party to understand the security level of a remote platform, security version numbers are designed into the Intel SGX architecture. Some of the version numbers indicate the patch level of the relevant phases of the processor boot up and system operations that affect the identity of the Intel SGX instructions.

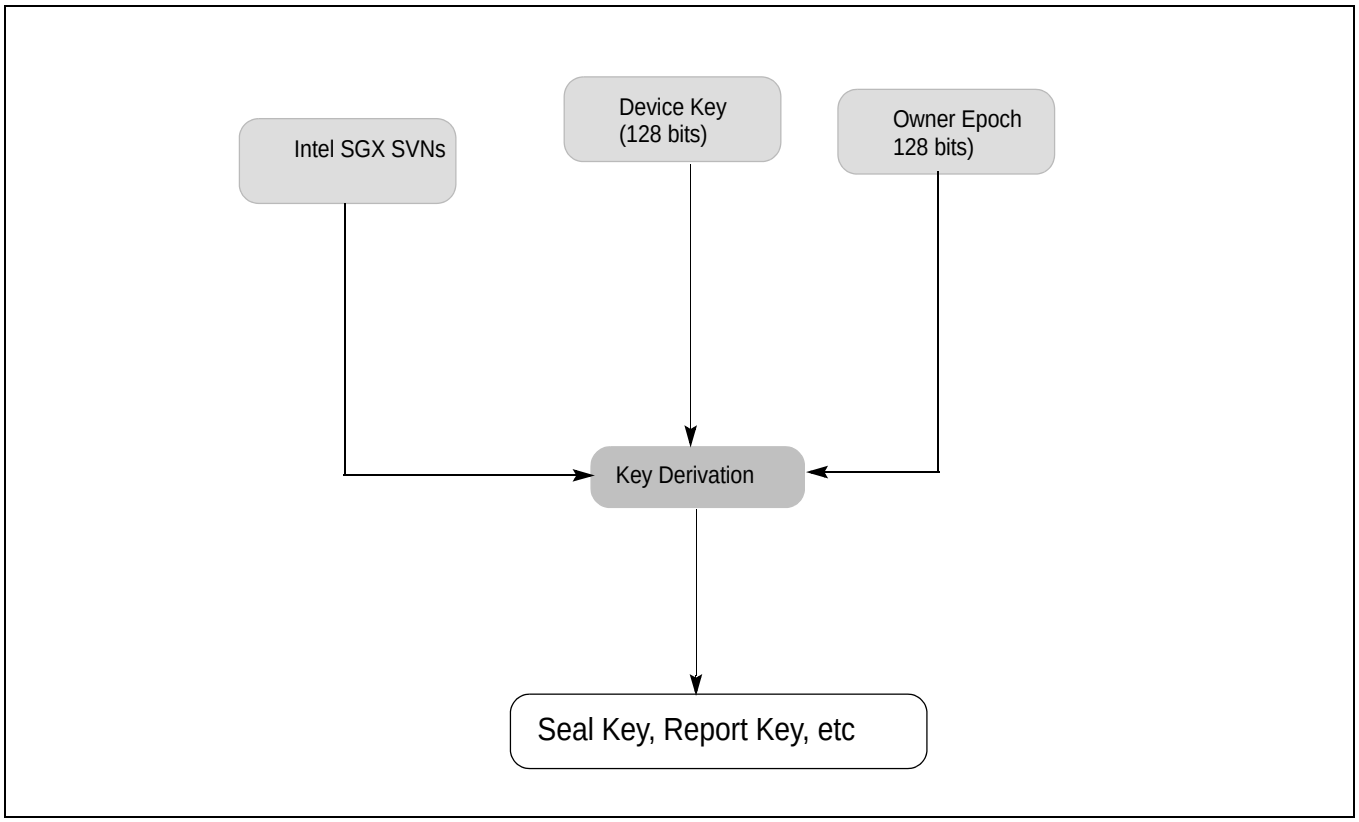


Figure 39-2. Intel® SGX Key Overview

The SVN values are reported to the remote user as part of the attestation process. They are part of the EREPORT instruction output.

Owner Epoch is a 128 bit value which is loaded into the SGXOWNEREPOCH0 and SGXOWNEREPOCH1 MSRs when Intel SGX is booted. These registers provide a user with the ability to add personal entropy into the key derivation process.

NOTE: Owner Epoch must be kept the same in order to decrypt data using the EGETKEY instruction. A different Owner Epoch will result in the failure to decrypt files sealed by EGETKEY in a previous boot.

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

One example of EPC implementation is 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. To oversubscribe the EPC the EPC manager must keep track of all EPC entries, type and state, context affiliation, SECS affiliation, so that it could manage this resource and properly swap pages out of and into the EPC.

On processors that support Intel SGX with Intel SGX instruction opcode support of SGX1 (i.e. CPUID.(EAX=12H, ECX=0):EAX.SGX1 = 1 but CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 0), Intel SGX instructions provide a number of primitives for managing memory resources used by an enclave.

Intel SGX includes the EWB instruction for securely evicting pages out of the EPC. EWB encrypts a page in the EPC and writes it to unprotected memory. In addition, EWB also creates a cryptographic 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.

Intel SGX 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 VMM. When a VMM reloads an evicted page, it needs to restore 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.

39.5.2.1 Enhancement to Managing EPC Pages

On processors with Intel SGX instruction opcode supporting SGX2 (i.e. CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 1), the EPC manager can manage EPC resources (while enclave is running) with more flexibility provided by the SGX2 instructions. The additional flexibility is described in Section 39.5.7 through Section 39.5.11

39.5.3 Eviction of Enclave Pages

Intel SGX paging is optimized to allow the OS to page out multiple EPC pages under a single synchronization.

1. For each enclave page to be evicted:
    a. Select a slot in a Version Array page.

- If no VA page exists with an empty slot, create a new PT_VA page using the EPA instruction.

b. Remove mapping from the page table (OS removes from system page table, VMM removes from EPT).

c. Execute EBLOCK for the target page. This sets the target page state to BLOCKED. At this point no new mappings of the page will be created. Accesses which do not have mapping cached in the TLB will generate a #PF.

2. For each enclave containing pages selected in step 1:

   — Execute an ETRACK on that enclave.

3. For all hardware threads 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 hardware threads to exit any enclaves they might be in, and as a result flush all TLB entries that might hold stale translations to blocked pages.

4. After enclaves exit, allow h/w threads to execute normally.

5. For each page to be evicted:

   — Evict the page using the EWB command. Parameters include the EPC page linear address (the OS or VMM needs to use its own, private page mapping for this because of step 1.c), the VA slot, a 4k byte buffer to hold the encrypted page contents, and a 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 an encrypted copy of each page data and page metadata, both 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. If the VA page or the parent SECS are not already in the EPC, they must be reloaded first.

1. Execute ELDB/ELDU, passing as parameters: the EPC page linear address (again, using a private mapping), the VA slot, the encrypted page, and the page metadata.
2. Create a mapping in the page table 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.

39.5.5 Eviction of an SECS Page

The eviction of an SECS page is similar to the eviction of an enclave page. The only difference is that an SECS page cannot be evicted until all other pages belonging to the enclave have been evicted. Since all other pages have been evicted, there will be no threads executing inside the enclave. When reloading an enclave, the SECS page must be reloaded before all other constituent pages.

1. Ensure all pages are evicted from enclave.
2. Select a slot in a Version Array page.

   — If no VA page exists with an empty slot, create a new one using EPA.

3. Evict the page using the EWB command. Parameters include the EPC page effective address, the VA slot, a 4k byte buffer to hold the encrypted page contents and a buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

39.5.6 Eviction of a Version Array Page

VA pages do not belong to any enclave. When evicting the VA page, a slot in a different VA page must be specified in order to provide versioning of the evicted VA page.

1. Select a slot in a Version Array page other than the page being evicted.
   — If no VA page exists with an empty slot, create a new one using EPA.
2. Evict the page using the EWB command. Parameters include the EPC page linear address, the VA slot, a 4k byte buffer to hold the encrypted page contents, and a buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

39.5.7 Allocating a Regular Page

On processors that support SGX2, allocating a new page is accomplished by invoking the EAUG instruction. 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 instruction to accept the new page into the enclave. Page allocation operations may be batched to improve efficiency.

The typical process for allocating a page is as follows:

1. Enclave requests additional memory from OS when the current allocation becomes insufficient.
2. The OS calls EAUG to add a new memory page to the enclave.
   a. EAUG may only be called on an invalid 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.
3. The enclave issues an EACCEPT instruction, which clears the pending bit. 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 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 OS may 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.

The procedure for allocating a TCS page is as follows:

1. Enclave requests an additional page from the OS.
2. The OS calls EAUG to add a new regular memory page to the enclave.
   a. EAUG may only be called on an invalid page.
   b. Successful completion of the EAUG instruction places the target page in the VALID and PENDING state.
3. The enclave issues an EACCEPT instruction, at which point the page becomes accessible for normal enclave use.
4. The enclave initializes the contents of the new page.
5. The enclave requests that the OS convert the page from type PT_REG to PT_TCS.
6. 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 the RWX bits to 000 because TCS pages may not be accessed by enclave code.

7. 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 removal of an enclave page as a special case of EMODT. 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. 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 protocol 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 calls EMODT 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 pages.
3. OS performs an ETRACK instruction to remove the TLB addresses from all the processors.
4. Enclave issues an EACCEPT instruction.
5. The OS may now permanently remove it (by calling EREMOVE).

39.5.10 Restricting the EPCM Permissions of a Page

On processors that support SGX2, restricting the EPCM permissions associated with a page is accomplished using the EMODPR instruction. 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 protocol for restricting the permissions of a page is as follows:

1. Enclave requests that the OS restrict the permissions of an EPC page.
2. OS performs permission restriction, TLB flushing, and page-table modifications.
   a. Invokes EMODPR to restrict permissions.
   b. Performs ETRACK.
   c. Updates page tables to match the new EPCM permissions.
   d. Sends IPIs to trigger enclave thread exit and TLB shootdown.
3. Enclave calls EACCEPT.
   a. Enclave may access page throughout the entire process.
   b. Successful call to EACCEPT guarantees that no stale TLB mappings are present.

39.5.11 Extending the EPCM Permissions of a Page

On processors that support SGX2, extending the EPCM permissions associated with a page is performed directly be the enclave using EMODPE. After performing the EPCM permission extension, the enclave requests that the OS update the page table permissions to match. Permission extension does not require enclave threads to leave the enclave---TLBs with stale references to the more restrictive permissions will be flushed on demand.

1. Enclave invokes EMODPE to extend the EPCM permissions associated with am EPC page.

2. Enclave requests that OS update the page tables to match the new EPCM permissions.
3. Enclave code resumes.
    a. If TLB mappings are present to the more restrictive permissions, the enclave thread will page fault. The OS sees 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 TLB mappings are not present, access to the page with the new permissions will succeed without an enclave exit.

39.6 CHANGES TO INSTRUCTION BEHAVIOR INSIDE AN ENCLAVE

This section covers instructions whose behavior changes when executed in enclave mode.

39.6.1 Illegal Instructions

The instructions listed in Table 39-1 are ring 3 instructions which become illegal when executed inside an enclave. Executing these instructions inside an enclave will generate a #UD fault.

The first row of Table 39-1 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 39-1 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 39-1 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 39-1 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.
LAR, VERR, VERW	#UD	Might provide access to kernel information.
ENCLU[EENTER], ENCLU[ERESUME]	#GP	Cannot enter an enclave from within an enclave.

RDTSC and RDTSCP instructions are legal instructions inside an enclave.

RDTSC and RDTSCP instructions can be disabled by setting CR4. TSD when inside an enclave.

RDTSC and RDTSCP instructions may cause a VM exit when inside an 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.

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.

39.6.2 RDRAND and RDSEED Instructions

These instructions may cause a VM exit if the "RDRAND exiting" VM-execution control is 1. Unlike other instructions that can cause VM exits, these instructions are legal inside an enclave. As noted in Section 6.5.5, any VM exit originating on an instruction boundary inside an enclave sets bit 27 of the exit-reason field of the VMCS. If a VMM receives a VM exit due to an attempt to execute either of these instructions determines (by that bit) that the execution was inside an enclave, it can do either of two things. It can clear the "RDRAND exiting" VM-execution control and execute VMRESUME; this will result in the enclave executing RDRAND or RDSEED again, and this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

NOTE

It is expected that VMMs that virtualize Intel SGX will not set "RDRAND exiting" to 1.

39.6.3 PAUSE Instruction

The PAUSE instruction may cause a VM exit if either of the "PAUSE exiting" and "PAUSE-loop exiting" VM-execution controls is 1. Unlike other instructions that can cause VM exits, the PAUSE instruction is legal inside an enclave.

If a VMM receives a VM exit due to the 1-setting of "PAUSE-loop exiting", it may take action to prevent recurrence of the PAUSE loop (e.g., by scheduling another virtual CPU of this virtual machine) and then execute VMRESUME; this will result in the enclave executing PAUSE again, but this time the PAUSE loop (and resulting VM exit) will not occur.

If a VMM receives a VM exit due to the 1-setting of "PAUSE exiting", it can do either of two things. It can clear the "PAUSE exiting" VM-execution control and execute VMRESUME; this will result in the enclave executing PAUSE again, but this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

NOTE

It is expected that VMMs that virtualize Intel SGX will not set "PAUSE exiting" to 1.

39.6.4 INT 3 Behavior Inside an Enclave

INT3 is legal inside an enclave, however, the behavior inside an enclave is different from its behavior outside an enclave. See Section 43.4.1 for details.

39.6.5 INVD Handling when Enclaves Are Enabled

Once processor reserved memory protections are activated (see Section 39.5), any execution of INVD will result in a #GP(0).

...

29. New Chapter 40, New Volume 3D

A new chapter, Chapter 40, has been added to the new volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

CHAPTER 40
ENCLAVE EXITING EVENTS

Certain events, such as exceptions and interrupts, incident to (but asynchronous with) enclave execution may cause control to transition to an address outside the enclave. (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

Asynchronous enclave exits push information onto the appropriate stack in a form expected by the operating system. To accomplish this, an address to trampoline code is pushed onto the exiting stack as the RIP. This trampoline code eventually returns to the enclave by means of an ENCLU(ERESUME) instruction.

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.

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 is placed into RAX, the TCS pointer is placed in RBX and the AEP (see below) is placed into RCX for later use when resuming the enclave after the exit.

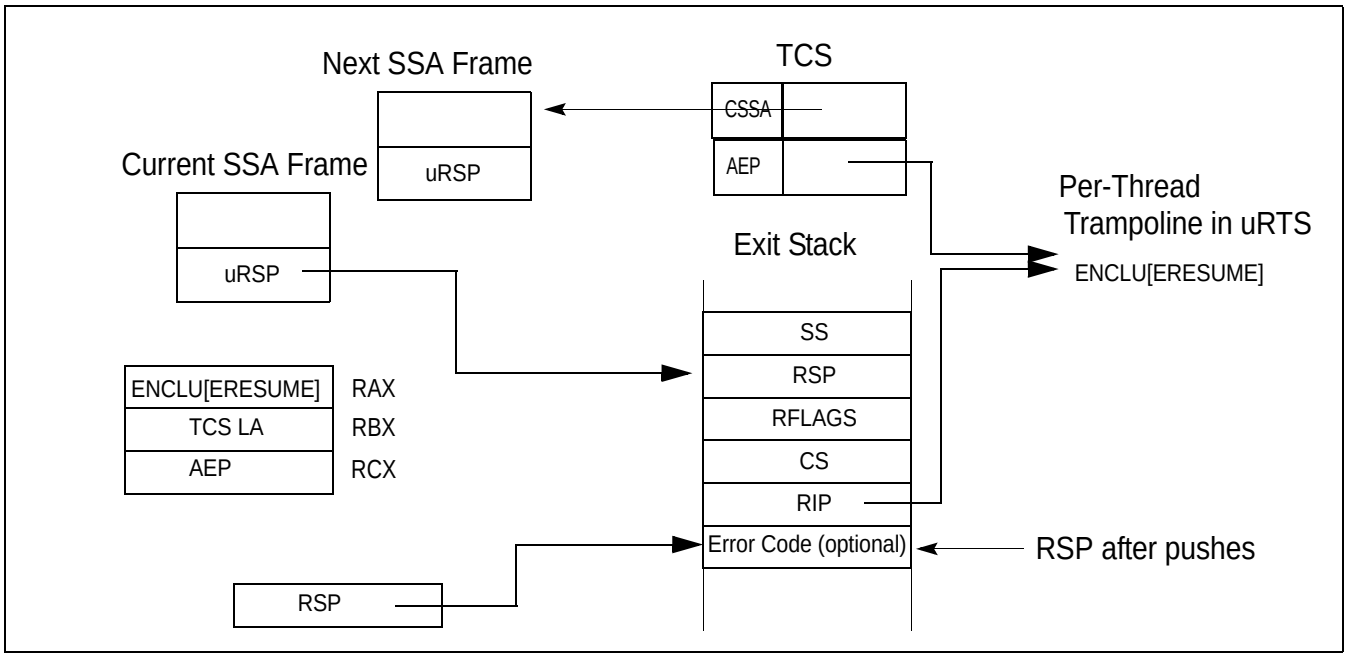


Figure 40-1. Exit Stack Just After Interrupt with Stack Switch

Upon an AEX, the AEP (Asynchronous Exit Pointer) is pushed onto the exit stack as the return 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.

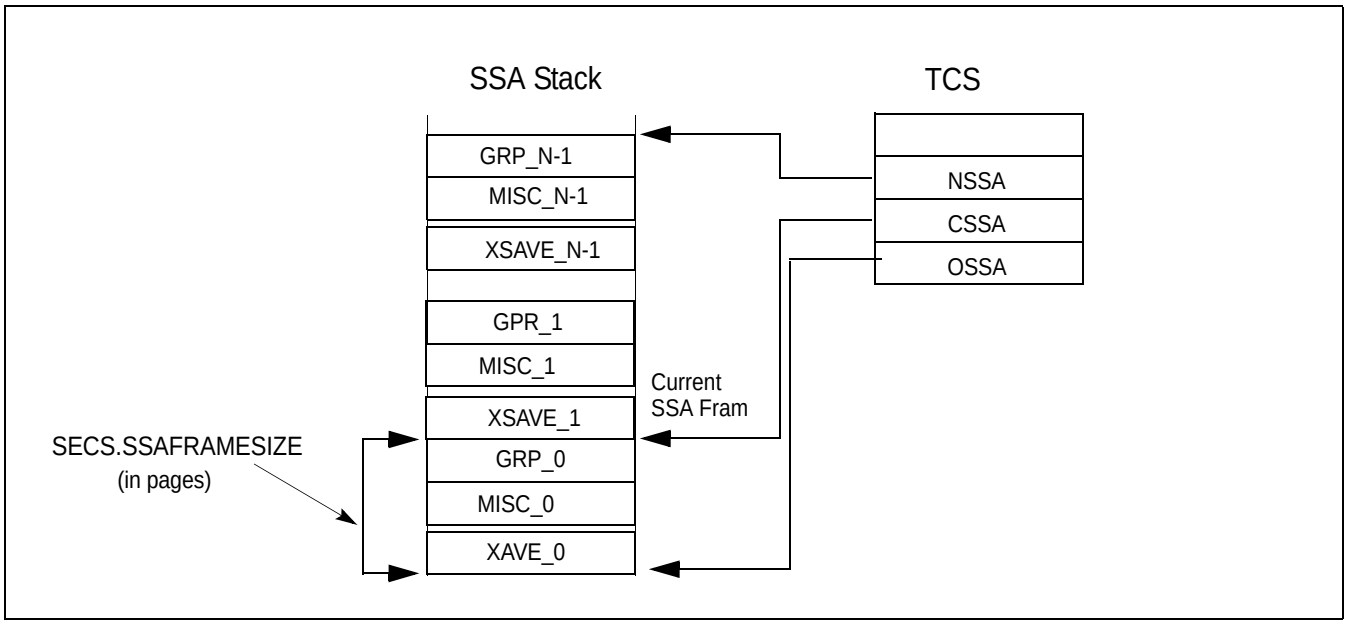


Figure 40-2. The SSA Stack

The location of the SSA frames to be used is controlled by three variables in the TCS:

Number of State Save Area Slots (NSSA). Defines the total number of slots (frames) in the State Save Area stack.

Current State Save Area Slot (CSSA). Defines the current slot to use on the next exit.

State Save Area (OSSA). Address of a set of save area slots large enough to hold the GPR state and the XSAVE state.

When an AEX occurs while executing on a thread inside the enclave, hardware selects the SSA frame to use by examining TCS.CSSA. Processor state (as described in Section 40.3.1) is saved and loaded with a synthetic state (to avoid leaking secrets) 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 inside the enclave.

The format of the XSAVE section of SSA is identical to the format used by the XSAVE/XRSTOR instructions.

Note: On EENTER, CSSA must be less than NSSA, ensuring that there is at least one Save Area available for exits.

Multiple SSA frames are defined to allow for a variety of behavior. When an AEX occurs the SSA frame is loaded and the pointer incremented. An ERESUME restores the processor state and frees the SSA frame. If after the AEX an EENTER is executed then the next SSA frame is reserved to hold state for another AEX. If there is no free SSA frame when executing EENTER, the entry will fail.

40.3 SYNTHETIC STATE ON ASYNCHRONOUS ENCLAVE EXIT

40.3.1 Processor Synthetic State on Asynchronous Enclave Exit

Table 40-1 shows the synthetic state loaded on AEX. The values written 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	TCS pointer of interrupted enclave thread.
RCX	AEP of interrupted enclave thread.
RDX, RSI, RDI	0.
RSP	Loaded from SSA.uRSP.
RBP	Loaded 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. Remaining 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 instruction mask and XCR0, and both clear in XSTATE_BV the XSAVE header.
FCW	On #MF exception: 037EH. On all other exits: 037FH.
FSW	On #MF exception: 8081H. On all other exits: 0H.
MXCSR	On #XM exception:1F01H. On all other exits: 0H.
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 it is the AEX that clears those bits, and EEE delivery occurs after AEX. Also, the address of an access causing a #PF during EEE delivery reveals no enclave secrets.
FS, GS	Including hidden portion. Restored to values as of most recent EENTER/ERESUME.

40.3.2 Synthetic State for Extended Features

When CR4.OSXSAVE = 1, extended features (those controlled by XCR0[63:2]) are set to their respective INIT states when this corresponding bit of SECS.XFRM is set. The INIT state is the state that would be loaded by the XRSTOR instruction had the instruction mask and the XSTATE_BV field of the XSAVE header each contained the value XFRM. (When the AEX occurs in 32-bit mode, those features that do not exist in 32-bit mode are unchanged.)

40.3.3 VMCS Synthetic State on Asynchronous Enclave Exit

All processor registers saved in the VMCS have the same synthetic values listed above. Additional VMCS fields that are treated specially on VM exit are listed in Table 40-2.

Table 40-2. VMCS Synthetic States on Asynchronous Enclave Exit

Field	Value
ENCLAVE_INTERRUPTION	A new configuration bit (bit 4 in the "Guest Interruptibility State" field) and an indicator (bit 27 in Basic VM-exit information field) for exit reasons. Set to 1 if exit occurred in enclave mode.
Guest-linear address	If the event that caused the AEX is an EPT violation that sets bit 7 of the Exit-Qualification field, the low 12 bits are cleared. Note: If the EPT violation occurs during delivery of an event that caused the AEX (e.g., an EPT violation that occurs during IDT vectoring), then the low 12 bits are NOT cleared.
Guest-physical address	If the event that caused the AEX is an EPT violation or mis-configured EPT, then the low 12 bits are cleared. Note: If the EPT violation or misconfiguration occurs during delivery of an event that caused the AEX (e.g., an EPT violation or misconfiguration that occurs during IDT vectoring), then the low 12 bits are NOT cleared.
Exit-Qualification	On page-fault that causes an AEX: low 12 bits are cleared. On APIC-access that causes an AEX: low 12 bits are cleared. Note: If either the page-fault or APIC-access occurs during delivery of an event that caused the AEX, the low 12 bits are NOT cleared.
VM-exit instruction length	Cleared.
VM-exit instruction information	This field is defined only for VM exits due to or during the execution of specific instructions (i.e. should be reported properly). Most of these instructions do not cause VM exits when executed inside an enclave. Exceptions are MOV DR, INVEPT, INVVPID, RDTSC, RDTSCP, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON. Normally, this field is defined for VM exits due to INT3 (or exceptions encountered while delivering INT3). This is not true for INT3 in an enclave, as the instruction becomes fault-like. INT3 Interruption types are reported as hardware exception when invoked inside enclave instead of 6 respectively when invoked outside enclave. This field is cleared for all other VM exits.
I/O RCX	Cleared.
I/O RSI	Cleared.
I/O RDI	Cleared.
I/O RIP	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 state of those miscellaneous features 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 the 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. In addition, on VM exit the VMCS or SMRAM state is initialized as described in Table 40-2.
3. In the current SSA frame, the cause of the AEX is saved in the EXITINFO field. See Table 38-9 for details and values of the various fields.
4. Any code and data breakpoints that were suppressed at the time of enclave entry are unsuppressed when exiting the enclave.
5. RFLAGS.TF is set to the value that it had at the time of the most recent enclave entry (an exception is made if that entry was opt-in; see Section 43.2). In the SSA, RFLAGS.TF is set to 0. However, due to the way TF is handled on enclave entry, this value is irrelevant (see EENTER and ERESUME instructions).
6. 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.
7. Any performance monitoring activity (including PEBS) on the exiting thread that was suppressed due to the enclave entry on that thread is unsuppressed. Any counting that had been demoted to MyThread (on other threads) is promoted back to AnyThread.

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
        SSA.RFLAGS.TF ← 0;
    ELSE    (* TMP_MODE64 = 1 *)
        Save RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8-R15, RFLAGS, RIP into SSA using CR_GPR_PA
        SSA.RFLAGS.TF ← 0;
FI;
Save FS and GS BASE into SSA using CR_GPR_PA;

(* Use a special version of XSAVE that takes a list of physical addresses of logically sequential pages to
    perform the save. TMP_MODE64 specifies whether to use the 32-bit or 64-bit layout.
    SECS.ATTRIBUTES.XFRM selects the features to be saved.
    CR_XSAVE_PAGE_n specifies a list of 1 or more physical addresses of pages that contain the XSAVE area.
*)
XSAVE(TMP_MODE64, SECS.ATTRIBUTES.XFRM, CR_XSAVE_PAGE_n);

(* 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 outside RSP and RBP from the current SSA frame.
    This is where they had been stored on most recent EENTER *)
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;

(* Update 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 (VMCS.MTF = 1)
   THEN Pend MTF VM Exit at the end of exit; FI;

(* Clear low 12 bits of CR2 on #PF *)
IF (Exception is #PF)
   THEN CR2 ← CR2 & ~0xFFF; FI;

(* end_of_flow *)

(* Execution continues with normal event processing. *)

...
```

30. New Chapter 41, New Volume 3D

A new chapter, Chapter 41, has been added to the new volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

CHAPTER 41
SGX INSTRUCTION REFERENCE

Supervisor and user level instructions provided by Intel® Software Guard Extensions are described in this chapter. In general, a various functionalities are encoded as leaf functions within the ENCLS (supervisor) and ENCLU (user) instruction mnemonics. Different leaf functions are encoded by specifying an input value in the EAX register of the respective instruction mnemonic.

41.1 INTEL® SGX INSTRUCTION SYNTAX AND OPERATION

ENCLS and ENCLU instruction mnemonics for all leaf functions are covered in this section.

For all instructions, the value of CS.D is ignored; addresses and operands are 64 bits in 64-bit mode and are otherwise 32 bits. Aside from EAX specifying the leaf number as input, each instruction leaf may require all or some subset of the RBX/RCX/RDX as input parameters. Some leaf functions may return data or status information in one or more of the general purpose registers.

41.1.1 ENCLS Register Usage Summary

Table 41-1 summarizes the implicit register usage of supervisor mode enclave instructions.

Table 41-1. Register Usage of Privileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
ECREATE	00H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EADD	01H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EINIT	02H (In)	SIGSTRUCT (In, EA)	SECS (In, EA)	EINITTOKEN (In, EA)
EREMOVE	03H (In)		EPCPAGE (In, EA)	
EDBGRD	04H (In)	Result Data (Out)	EPCPAGE (In, EA)	
EDBGWR	05H (In)	Source Data (In)	EPCPAGE (In, EA)	
EEXTEND	06H (In)		EPCPAGE (In, EA)	
ELDB	07H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDU	08H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
EBLOCK	09H (In)		EPCPAGE (In, EA)	
EPA	0AH (In)	PT_VA (In)	EPCPAGE (In, EA)	
EWB	0BH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ETRACK	0CH (In)		EPCPAGE (In, EA)	
EAUG	0DH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	LINADDR
EMODPR	0EH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODT	0FH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EA: Effective Address				

41.1.2 ENCLU Register Usage Summary

Table 41-2 Summarized the implicit register usage of user mode enclave instructions.

Table 41-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
EREPORT	00H (In)	TARGETINFO (In, EA)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)
EGETKEY	01H (In)	KEYREQUEST (In, EA)	KEY (In, EA)	
EENTER	02H (In)	TCS (In, EA)	AEP (In, EA)	
	RBX.CSSA (Out)		Return (Out, EA)	

Table 41-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
ERESUME	03H (In)	TCS (In, EA)	AEP (In, EA)	
EEXIT	04H (In)	Target (In, EA)	Current AEP (Out)	
EACCEPT	05H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODPE	06H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EACCEPTCOPY	07H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	EPCPAGE (In, EA)
EA: Effective Address				

41.1.3 Information and Error Codes

Information and error codes are reported by various instruction leaf functions to show an abnormal termination of the instruction or provide information which may be useful to the developer. Table 41-3 shows the various codes and the instruction which generated the code. Details of the meaning of the code is provided in the individual instruction.

Table 41-3. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
No Error	0	
SGX_INVALID_SIG_STRUCT	1	EINIT
SGX_INVALID_ATTRIBUTE	2	EINIT, EGETKEY
SGX_BLSTATE	3	EBLOCK
SGX_INVALID_MEASUREMENT	4	EINIT
SGX_NOTBLOCKABLE	5	EBLOCK
SGX_PG_INVLD	6	EBLOCK
SGX_LOCKFAIL	7	EBLOCK, EMODPR, EMODT
SGX_INVALID_SIGNATURE	8	EINIT
SGX_MAC_COMPARE_FAIL	9	ELDB, ELDU
SGX_PAGE_NOT_BLOCKED	10	EWB
SGX_NOT_TRACKED	11	EWB, EACCEPT
SGX_VA_SLOT_OCCUPIED	12	EWB
SGX_CHILD_PRESENT	13	EWB, EREMOVE
SGX_ENCLAVE_ACT	14	EREMOVE
SGX_ENTRYEPOCH_LOCKED	15	EBLOCK
SGX_INVALID_EINIT_TOKEN	16	EINIT
SGX_PREV_TRK_INCMPL	17	ETRACK
SGX_PG_IS_SECS	18	EBLOCK
SGX_PAGE_ATTRIBUTES_MISMATCH	19	EACCEPT, EACCEPTCOPY
SGX_PAGE_NOT_MODIFIABLE	20	EMODPR, EMODT
SGX_INVALID_CPUSVN	32	EINIT, EGETKEY
SGX_INVALID_ISVSVN	64	EGETKEY

Table 41-3. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
SGX_UNMASKED_EVENT	128	EINIT
SGX_INVALID_KEYNAME	256	EGETKEY

41.1.4 Internal CREGs

The CREGs as shown in Table 5-4 are hardware specific registers used in this document to indicate values kept by the processor. These values are used while executing in enclave mode or while executing an Intel SGX instruction. These registers are not software visible and are implementation specific. The values in Table 41-4 appear at various places in the pseudo-code of this document. They are used to enhance understanding of the operations.

Table 41-4. List of Internal CREG

Name	Size (Bits)	Scope
CR_ENCLAVE_MODE	1	LP
CR_DBGOPTIN	1	LP
CR_TCS_LA	64	LP
CR_TCS_PH	64	LP
CR_ACTIVE_SECS	64	LP
CR_ELRANGE	128	LP
CR_SAVE_TF	1	LP
CR_SAVE_FS	64	LP
CR_GPR_PA	64	LP
CR_XSAVE_PAGE_n	64	LP
CR_SAVE_DR7	64	LP
CR_SAVE_PERF_GLOBAL_CTRL	64	LP
CR_SAVE_DEBUGCTL	64	LP
CR_SAVE_PEBS_ENABLE	64	LP
CR_CPUSVN	128	PACKAGE
CSR_SGX_OWNEREPOCH	128	PACKAGE
CSR_INTELPUBKEYHASH	32	PACKAGE
CR_SAVE_XCR0	64	LP
CR_SGX_ATTRIBUTES_MASK	128	LP
CR_PAGING_VERSION	64	PACKAGE
CR_VERSION_THRESHOLD	64	PACKAGE
CR_NEXT_EID	64	PACKAGE
CR_BASE_PK	128	PACKAGE
CR_SEAL_FUSES	128	PACKAGE

41.1.5 Concurrent Operation Restrictions

To protect the integrity of Intel SGX data structures, under certain conditions, Intel SGX disallows certain leaf functions from operating concurrently. Listed below are some examples of concurrency that are not allowed.

- For example, Intel SGX disallows the following leafs to concurrently operate on the same EPC page.
  - ECREATE, EADD, and EREMOVE are not allowed to operate on the same EPC page concurrently with themselves or any other Intel SGX leaf function.
  - EADD, EEXTEND, and EINIT leafs are not allowed to operate on the same SECS concurrently.
- Intel SGX disallows the EREMOVE leaf from removing pages from an enclave that is in use.
- Intel SGX disallows entry (EENTER and ERESUME) to an enclave while a page from that enclave is being removed.

When disallowed operation is detected, a leaf function causes an exception. To prevent such exceptions, software must serialize leaf functions or prevent these leaf functions from accessing the same resource.

41.1.5.1 Concurrency Table of Intel® SGX Instructions

Summary tables of concurrency describing whether a given Intel SGX instruction leaf is allowed to execute while another leaf function is executing or owns common resource. 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.

The concurrency restriction depends on the type of EPC page and the parameter of the two concurrent instructions each Intel SGX instruction leaf attempts to operate on. The spectrum concurrency behavior of the instruction leaf 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.
- 'Y': The instruction leaf listed in a given row may execute concurrently with the instruction leaf shown in the respective column.
- '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 concurrent execution of two leaf functions that is architecturally impossible or restricted, e.g. executing an ENCLU and an ENCLS leaf on the same logical processor, or executing two leaves with incompatible EPCM state requirements. Concurrent execution of two such leaf instructions may result in a page fault in one of the leaf instructions.

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.

41.2 INTEL® SGX INSTRUCTION REFERENCE

ENCLS—Execute an Enclave System Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
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 instruction also results in a #UD if CR0.PE is 0 or RFLAGS.VM is 1, or if it is executed from inside SMM. Additionally, any attempt to execute this instruction when current privilege level is not 0 results in #UD.

Any attempt to invoke an undefined leaf function results in #GP(0).

If CR0.PG is 0, any attempt to execute ENCLS results in #GP(0).

In VMX non-root operation, execution of ENCLS is unconditionally allowed if the "Enable ENCLS exiting" VM-execution control is cleared. If the "Enable ENCLS exiting" VM-execution control is set, execution of individual leaf function of ENCLS is governed by the "ENCLS-exiting bitmap". Each bit position of "ENCLS-exiting bitmap" corresponds to the index (EAX) of an ENCLS leaf function.

Software in VMX root mode of operation can intercept the invocation of various ENCLS leaf functions from VMX non-root mode by setting the Enable_ENCLS_EXITING control and writing the desired bit patterns into the "ENCLS-exiting bitmap" (accessed via encoding pair 0202EH/0202FH). A processor implements the Enable_ENCLS_EXITING VM-execution control field if IA32_VMX_PROCBASED_CTLS2[15] is read as 1.

The DS segment is used to create linear addresses.

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.

Segment prefix override is ignored. Address size prefix (67H) override is ignored.

REX prefix is ignored 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 (CPL > 0)
    Then #UD; FI;

IF ( (in VMX non-root operation) and ( Enable_ENCLS_EXITING = 1) )
```

```
    Then
        IF ( ((EAX < 63) and (ENCLS_EXITING_Bitmap[EAX] = 1)) or (EAX> 62 and ENCLS_EXITING_Bitmap[63] = 1) )
            Then
            Set VMCS.EXIT_REASON = ENCLS;
            Deliver 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;

IN_64BIT_MODE ← IA32_EFER.LMA AND CS.L ? 1 : 0;

IF (IN_64BIT_MODE = 0 and (DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1)
    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 that are used for operating the enclaves.

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 instruction also results in a #UD if CR0.PE is 0 or RFLAGS.VM is 1, or if it is executed from inside SMM. Additionally, any attempt to execute this instruction when current privilege level is not 3 results in #UD.

Any attempt to invoke an undefined leaf function results in #GP(0).

Any attempt to execute ENCLU instruction when paging is disabled or in MS-DOS compatible mode results in #GP.

The DS segment is used to create linear addresses.

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.

Segment prefix override is ignored. Address size prefix (67H) override is ignored.

REX prefix is ignored 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 (IN_64BIT_MODE = 0 and ((CS.D = 0) or (DS.B = 0) )
    Then #GP(0); FI;

IF (CR_ENCLAVE_MODE = 1 and ((EAX = EENTER) or (EAX = ERESUME) ) )
    Then #GP(0); FI;

IF (CR_ENCLAVE_MODE = 0 and ((EAX = EGETKEY) or (EAX = EREPORT) or (EAX = EEXIT) or (EAX = EACCEPT) or
    (EAX = EACCEPTCOPY) or (EAX = EMODPE) ) )
    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.

41.3 INTEL® SGX SYSTEM LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLS instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EADD—Add a Page to an Uninitialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLS[EADD]	IR	V/V	SGX1	This leaf function adds a page to an uninitialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EADD (In)	Address of a PAGEINFO (In)	Address of the destination EPC page (In)

Description

This leaf function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. As part of the association, the enclave offset and the security attributes are measured and extended into the SECS.MRENCLAVE. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of EADD leaf function.

EADD Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EADD Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	If security attributes specifies a TCS and the source page specifies unsupported TCS values or fields.
The SECS has been initialized.	The specified enclave offset is outside of the enclave address space.

Concurrency Restrictions

Table 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
	Type	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		
	Type	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] ← 000000044444145H; // "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(fault 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(fault 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
	Type	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		
	Type	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;

(* 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(fault 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(fault 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:

EBLOCK Error Codes

0 (No Error)	EBLOCK successful
SGX_BLKSTATE	Page already blocked. This value is used to indicate that the page was already EBLOCKed and thus will need to be restored to this state when it is eventually reloaded (using ELDB).
SGX_ENTRYEPOCH_LOCKED	This value indicates that an ETRACK is currently executing on the SECS. The EBLOCK should be re-attempted.
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 ECREATE, ELDU/ELDB, or EWB.

Concurrency Restrictions

Table 41-9. 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
	Type	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			C	Y	C		C	Y	N	C		N
	SECS			Y	C	Y	Y	Y			Y			Y		Y		Y	Y			Y	

Table 41-10. Concurrency Restrictions of EBLOCK with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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		C		N	C	C	N				Y	C	N	C			C			
	SECS	Y	Y		Y	C	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;

(* Check concurrency with other Intel SGX instructions *)
IF (ETRACK executed concurrently)
    Then
        RAX← SGX_ENTRYEPOCH_LOCKED;
        RFLAGS.ZF ← 1;
        goto Done;
    ELSIF (Other Intel SGX instructions reading or writing EPCM)
        RAX← SGX_LOCKFAIL;
        RFLAGS.ZF ← 1;
        goto Done;
    FI;
FI;

IF (EPCM(DS:RCX). VALID = 0)
    Then
        RFLAGS.ZF ← 1;
        RAX← SGX_PG_INVLD;
        goto Done;
FI;

IF ( (EPCM(DS:RCX).PT != PT_REG) and (EPCM(DS:RCX).PT != PT_TCS) and (EPCM(DS:RCX).PT != PT_TRIM) )
    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(fault 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(fault 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 41-11. 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
	Type	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 41-12. Concurrency Restrictions of ECREATE with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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 and 0FFFFFFFF00000000H) )
    Then #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE and 0FFFFFFE000000000H) )
    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).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(fault 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(fault 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 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 41-13. 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
	Type	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	

Table 41-14. Concurrency Restrictions of EDBGRD with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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			

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) & 0xFFF >= 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(fault 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(fault 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.	

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 EDBGWR with Other Intel® SGX Operations 1 of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Type	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	

Table 41-16. Concurrency Restrictions of EDBGWR with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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			

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) & 0xFF8H != offset_of_FLAGS & 0FF8H) )
    Then #GP(0); FI;

(* Locate the SECS for the enclave to which the DS:RCX page belongs *)
TMP_SECS ← GET_SECS_PHYS_ADDRESS(EPCM(DS:RCX).ENCLAVESECS);

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
    Then #GP(0); FI;
```

```
IF ( (TMP_MODE64 = 1) )
    Then (DS:RCX)[63:0] ← RBX[63:0];
    ELSE (DS:RCX)[31:0] ← EBX[31:0];
FI;
```

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(fault 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(fault 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	RCX
IR	EEXTEND (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.

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

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 41-17. 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
	Type	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					N	Y	Y			Y			N		N			N			Y	

Table 41-18. Concurrency Restrictions of EEXTEND with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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	Y		Y		N				N		N						

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: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();

(* 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 RCS 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(fault code)	If a page fault occurs in accessing memory operands.
	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 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(fault code)	If a page fault occurs in accessing memory operands.
	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.

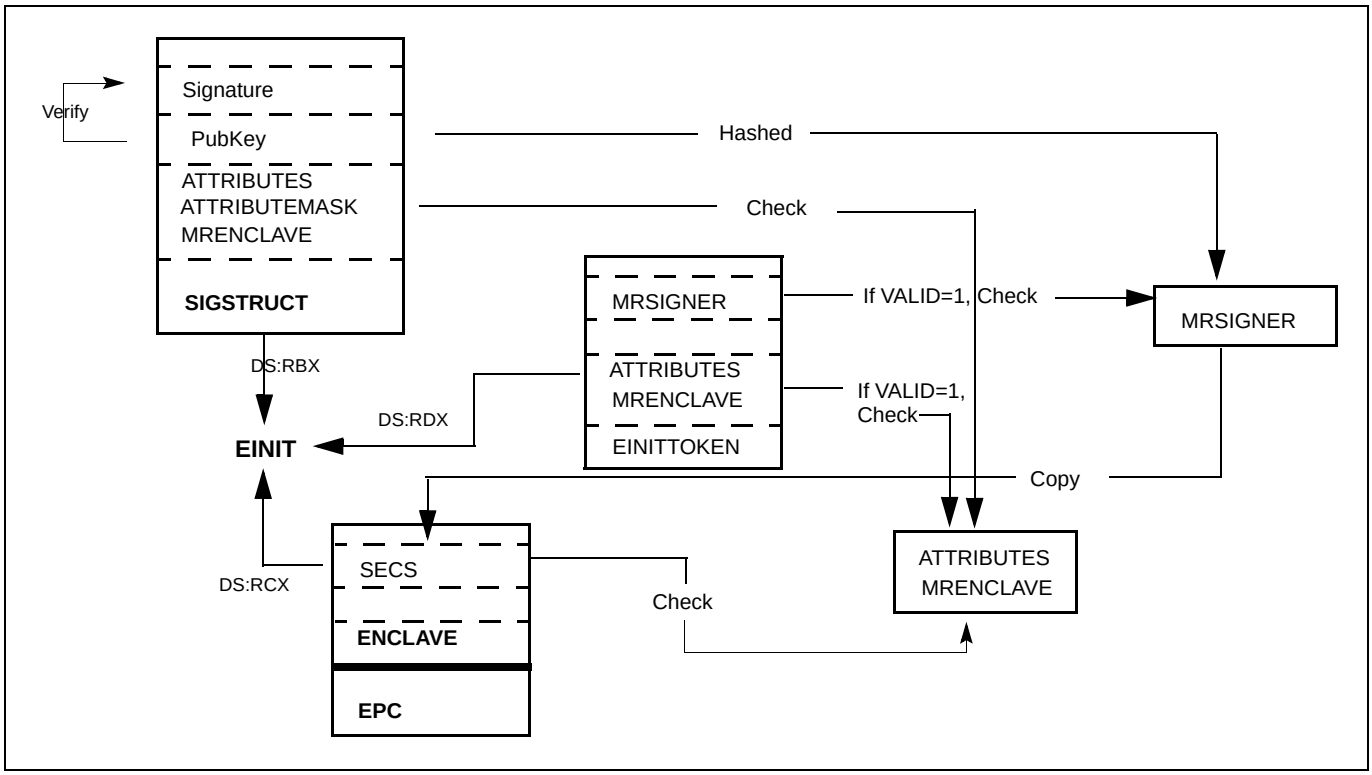


Figure 41-1. Relationships Between SECS, SIGSTRUCT and EINITTOKEN

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 Intel-only bits are set in SIGSTRUCT.ATTRIBUTES unless SIGSTRUCT was signed by Intel.

Checks that SIGSTRUCT.ATTRIBUTES equals the result of logically and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.

If EINITTOKEN.VALID is 0, checks that SIGSTRUCT is signed by Intel.

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 are INTR, NMI, SMI, INIT, VMX_TIMER, MCAKIND, MCE_SMI, and CMCI_SMI. EINIT does not fail if the pending event is inhibited (e.g., INTR could be inhibited due to MOV/POP SS blocking and STI blocking).

RFLAGS.{CF,PF,AF,OF,SF} are set to 0. 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.

Concurrency Restrictions

Table 41-19. 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
	Type	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 41-20. Concurrency Restrictions of EINIT with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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.
INTEL_ONLY_MASK	ATTRIBUTES	16Bytes	Constant mask of all ATTRIBUTE bits that can only be set for Intel enclaves.
CSR_INTELPUBKEYHASH		32Bytes	Constant with the SHA256 of the Intel Public key used to sign Architectural 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 INTEL_ONLY ATTRIBUTES are set, SIGSTRUCT must be signed using the Intel Key *)
INTEL_ONLY_MASK ← 0000000000000020H;
IF ( ( (DS:RCX.ATTRIBUTES & INTEL_ONLY_MASK) != 0) and (TMP_MRSIGNER != CSR_INTELPUBKEYHASH) )
    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 Intel *)
IF (TMP_TOKEN.VALID[0] = 0)
    IF (TMP_MRSIGNER != CSR_INTELPUBKEYHASH)
        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 ← LAUNCH_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 ← 0;
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(fault 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(fault 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:

ELDB/ELDU Error Codes

0 (No Error)	ELDB/ELDU successful
SGX_MAC_COMPARE_FAIL	If the MAC check fails.

Concurrency Restrictions

Table 41-21. Concurrency Restrictions of ELDB/ELDU with Intel® SGX Instructions - 1of 2

Operation		EEXIT			EADD		EBLOCK		ECREATE	EDBGRD/WR		EENTER/ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA
	Type	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 41-22. Concurrency Restrictions of ELDB/ELDU with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
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						

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).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(fault 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(fault 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 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.	

Concurrency Restrictions

Table 41-23. 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
	Type	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					N					N			N
	SECS			Y		N		Y			Y			Y		N		Y	N			Y	

Table 41-24. Concurrency Restrictions of EMODPR with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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					N			N		C		C		C		C			C	Y	Y
	SECS	Y	Y		Y	N	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;

(* Check concurrency with ETRACK *)
IF (ETRACK executed concurrently)
    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;

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(fault 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(fault 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 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.	

Concurrency Restrictions

Table 41-25. 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
	Type	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		N	N	N		N	N		C			N				N	N	N		N
	SECS			Y		N	Y	Y			Y			Y		N		Y	N			Y	

Table 41-26. Concurrency Restrictions of EMODT with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	Targ	SECS	Param	SECS	SECS	SRC	VA	SECS	Targ	SECS	Targ	SECINFO	Targ	SECS	Targ	SECS	Targ	SECINFO	SECS	Targ	SRC	SECINFO
EMODT	Targ	N					N	N		N	N	C		C		C		C			C	Y	Y
	SECS	Y	Y		Y	C	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).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(fault 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(fault 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 41-27. 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
	Type	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 41-28. Concurrency Restrictions of EPA with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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).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(fault 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(fault 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:

EREMOVE Error Codes

0 (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 41-29. 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
	Type	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	N	C	C	C	N	N	N	N	N
	SECS			C		Y	Y	Y			Y			C		Y		C	Y			Y	

Table 41-30. Concurrency Restrictions of EREMOVE with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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			N	C	N	C			C			
	SECS	Y	Y	Y	C	Y	Y		Y		Y				Y		Y			C			

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 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;

SCRATCH_SECINFO ← DS:RBX.TMP_SECINFO;

(* 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(fault 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(fault 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:

ETRACK Error Codes

0 (No Error)	ETRACK successful.
SGX_PREV_TRK_INCMPL	All logical processors on the platform did not complete the previous tracking cycle.

Concurrency Restrictions

Table 41-31. 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
	Type	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		N	N	N	Y			Y		Y		Y	Y	N		Y	N

Table 41-32. Concurrency Restrictions of ETRACK with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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				N		N			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(fault 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(fault code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

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

Concurrency Restrictions

Table 41-33. 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
	Type	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	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	

Table 41-34. Concurrency Restrictions of EWB with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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			N	C	N	C			C			
	VA	N					N	Y		N						N							
	SECS	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 #P(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 & ~0xFFF).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;

(* 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.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(fault 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(fault 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/Write 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.

Concurrency Restrictions

Table 41-35. 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
	Type	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	

Table 41-36. Concurrency Restrictions of EACCEPT with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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	Y
	SECINFO			U								Y	Y					Y	Y			U	Y
	SECS	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).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;

IF (DS:RCX is not 512Byte 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 & 0xFFF != 0xFFF) or (DS:RCX.GSLIMIT & 0xFFF != 0xFFF)) )
    Then #GP(0); FI;

(* Clear PENDING/MODIFIED flags to mark accept operation complete *)
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 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(fault 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(fault 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.

Concurrency Restrictions

Table 41-37. 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
	Type	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 41-38. Concurrency Restrictions of EACCEPTCOPY with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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		
	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).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 ) 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(fault 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(fault 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 != 0x3.
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 43.2.6).
  - — 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 43.2.4):
  - — 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 41-39. 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
	Type	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			U				U						
	SECS			Y		N	Y	Y			Y			Y		N		Y	N			Y	

Table 41-40. Concurrency Restrictions of EENTER with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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			

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 & & 0xFFFFFFFFFFFFFFFE) != 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;
        Clear_all_pending_debug_exceptions;
        Clear_pending_MTF_VM_exit;
    ELSE
        IF (RFLAGS.TF = 1)
            Then Pend_Single-Step_#DB_at_the_end_of_ENTER; FI;
        IF (VMCS.MTF = 1)
            Then Pend_MTF_VM_exit_at_the_end_of_ENTER; 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(fault 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.
#NM	If CR0.TS is set.

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(fault 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.
#NM	If CR0.TS is set.

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 41-41. 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
	Type	Targ	VA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA
EEXIT	TCS	N	N	N			Y	N		Y	N	N		N	N	N		N					
	SSA		U	N			Y	N		Y	N		U	N	N	N	U	N					
	SECS			Y		N	Y	Y			Y			Y		N		Y	N		N	Y	

Table 41-42. Concurrency Restrictions of EEXIT with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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	N		N		Y		N						N	Y	N	Y		N			
	SSA	Y	N	U	N		Y		N			Y	U	Y	N	Y	N	Y	U	N		U	U
	SECS	Y	Y		Y	Y	Y		Y		Y				Y		Y		N	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)
    Pend Single-Step #DB at the end of EEXIT;
FI;

IF (VMCS.MTF = 1)
    Pend 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(fault 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(fault 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 41-43
- 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 41-43) 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 41-43 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 41-43. 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;					
Launch	Yes	Request	Yes	Request	Request	Yes	No	No	Request
Report	Yes	Yes	Yes	Yes	No	No	Yes	No	Request
Seal	Yes	Request	Yes	Request	Request	Yes	Request	Request	Request
Provisioning	Yes	Request	Yes	Request	Request	Yes	No	Yes	Yes
Provisioning Seal	Yes	Request	Yes	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.

Some keys are derived based on a hardcode PKCS padding constant (352 byte string):

HARDCODED_PKCS1_5_PADDING[15:0] ß 0100H;

HARDCODED_PKCS1_5_PADDING[2655:16] ß SignExtend330Byte(-1); // 330 bytes of 0FFH

HARDCODED_PKCS1_5_PADDING[2815:2656] ß 2004000501020403650148866009060D30313000H;

The error codes are:

EGETKEY Error Codes

0 (No Error)	EGETKEY successful.
SGX_INVALID_ATTRIBUTE	The KEYREQUEST contains a KEYNAME for which the enclave is not authorized.
SGX_INVALID_CPUSVN	If KEYREQUEST.CPUSVN is beyond 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 41-44. 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
	Type	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	

Table 41-45. Concurrency Restrictions of EGETKEY with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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			

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 ← 0;
        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(fault 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(fault 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 41-46. 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
	Type	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 41-47. Concurrency Restrictions of EMODPE with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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(fault 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(fault 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	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.

An effective address not properly aligned. May page fault.	An memory address does not resolve in an EPC page.

Concurrency Restrictions

Table 41-48. 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
	Type	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	

Table 41-49. Concurrency Restrictions of EREPORT with Other Intel® SGX Operations 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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			

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_REPORTKEY[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(fault 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(fault 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 != 0x3.
CR4.OSFXSR != 1	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
Offsets 520-535 of the XSAVE area not 0	The bit vector stored at offset 512 of the XSAVE area must be a subset of SECS.ATTRIBUTES.XFRM.
The SSA frame is not valid or in use	

If CR0.TS is set, ERESUME generates a #NM exception.

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 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 43.2.6).

— On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 43.2.4).

- 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 43.2.4):
  — 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 41-50. 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
	Type	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			U				U						
	SECS			Y		N	Y	Y			Y			Y		N		Y	N			Y	

Table 41-51. Concurrency Restrictions of ERESUME with Intel® SGX Instructions - 2 of 2

Operation		EREMOVE		EREPORT		ETRACK	EWB			EAUG		EMODPE		EMODPR		EMODT		EACCEPT			EACCEPTCOPY		
	Type	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 & & 0xFFFFFFFFFFFFFFFE) != 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_monitor_trap_flag for the source of the execution of the enclave;
        Clear_all_pending_debug_exceptions;
        Clear_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(fault 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.
#NM	If CR0.TS is set.

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(fault 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.
#NM	If CR0.TS is set.

...

31. New Chapter 42, New Volume 3D

A new chapter, Chapter 42, has been added to the new volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

CHAPTER 42
INTEL® SGX INTERACTIONS WITH IA32 AND INTEL® 64 ARCHITECTURE

Intel® SGX provides Intel® Architecture with a collection of enclave instructions for creating protected execution environments on processors supporting IA32 and Intel® 64 architectures. These Intel SGX instructions are designed to work with legacy software and the various IA32 and Intel 64 modes of operation.

42.1 INTEL® SGX AVAILABILITY IN VARIOUS PROCESSOR MODES

The Intel SGX extensions (see Table 37-1) are available only when the processor is executing in protected mode of operation. Additionally, the extensions are not available in System Management Mode (SMM) of operation or in Virtual 8086 (VM86) mode of operation. Finally, all leaf functions of ENCLU and ENCLS require CR0.PG enabled.

The exact details of exceptions resulting from illegal modes and their priority are listed in the reference pages of ENCLS and ENCLU.

42.2 IA32_FEATURE_CONTROL

A new bit in IA32_FEATURE_CONTROL MSR (bit 18) is provided to 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.

42.3 INTERACTIONS WITH SEGMENTATION

42.3.1 Scope of Interaction

Intel SGX extensions are available only when the processor is executing in a protected mode operation (see Section 42.1 for Intel SGX availability in various processor modes). Enclaves abide by all the segmentation policies set up by the OS.

Intel SGX interacts with segmentation at two levels:

- The Intel SGX instruction (see the enclave instruction in Table 37-1).
- logical-processor execution inside an enclave (legacy instructions and enclave instructions permitted inside an enclave).

42.3.2 Interactions of Intel® SGX Instructions with Instruction Prefixes and Addressing

All the memory operands used by the Intel SGX instructions are interpreted as offsets within the data segment (DS). The segment-override prefix on Intel SGX instructions is ignored.

Operand size is fixed for each enclave instruction. The operand-size prefix is reserved, and results in a #UD exception.

All address sizes are determined by the operating mode of the processor. The address-size prefix is ignored. This implies that while operating in 64-bit mode of operation, the address size is always 64 bits, and while operating in 32-bit mode of operation, the address size is always 32 bits. Additionally, when operating in 16-bit addressing, memory operands used by enclave instructions use 32 bit addressing; the value of CS.D is ignored.

42.3.3 Interaction of Intel® SGX Instructions with Segmentation

The Intel SGX instructions used for entering the enclave (ENCLU[EENTER] and ENCLU[ERESUME]) ensure that all usable segment registers (i.e., the segment registers that have "Segment Unusable" bit in "Access Rights" field, a.k.a., "null" bit, set to 0) except for FS and GS have a zero base.

Additionally they save the existing contents of the FS/GS segment registers (including the hidden portion) in the processor, and load those registers with new values. The instructions also ensure that the linear ranges and

access rights available under the newly-loaded FS and GS are subsets of the linear-address range/access rights available under DS. See EENTER Leaf and ERESUME Leaf in Chapter 41 for exact details of this computation.

Any exit from the enclave either via ENCLU[EEXIT] or via an AEX restores the saved values of FS/GS segment registers.

The enclave-entry instructions also ensure that the CS segment mode (64-bit vs 32 bit) is 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.

Finally, 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.

42.3.4 Interactions of Enclave Execution with Segmentation

During the course of execution, enclave code abides by all segmentation policies as dictated by legacy 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 (via MOV seg register, POP seg register, LDS, far jump, etc.) results in the generation of a #UD.

Execution of WRFSBASE and WRGSBASE from inside a 64-bit enclave does not generate the #UD exception. If the software running inside an enclave modifies the segment-base values for these registers using the WRFSBASE and WRGSBASE instructions, the new values are saved into the current SSA frame on an asynchronous enclave exit (AEX) and restored 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 in paged mode of operation. Any attempt to execute these leaf functions in non-paged mode of operation results in delivery of #UD to the system software (OS or VMM).

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the data segments, 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[EEXIT] 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 of 3), enclave execution is always subject to paging-based access control. The Intel SGX access control itself is implemented as an extension to the traditional IA-32 and Intel 64 paging state machine. See Section 38.5 for details.

It should be noted that Intel SGX instructions may set the A and D bit on non-faulting EPC pages, even if the instruction may eventually fault due to 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 or VMX-non-root mode, as long as:

- The software is not running in SMM mode of operation.
- The software is using a legal mode of operation (see Section 42.1).

A VMM has the flexibility to configure the VMCS to permit a guest to use the entirety of the ENCLS leaf functions or any sub-set of the ENCLS leaf functions at the granularity of individual leaf function. Availability of the ENCLU leaf functions in VMX non-root operation has the same requirement as ENCLU leaf functions outside of a virtualized environment.

Enhancement in the VMCS to allow configurability for Intel SGX in a guest is enumerated by VMX capability MSRs. A summary of the enumerated capability is listed in Table 42-1.

Table 42-1. Summary of VMX Capability Enumeration MSRS for Processors Supporting Intel® SGX

Interface	Description
IA32_VMX_PROCBASED_CTLS2[bit 15]	Mirrors the value of CPUID.(EAX=07H, ECX=0).EBX.SGX.
IA32_VMX_MISC[bit 30]	If 1, VM entry checks that the VM-entry instruction length is in the range 0-15. See Section 42.5.3.

Details of the VMCS control to allow VMM to configure support of Intel SGX in guest operation is described in Section 42.5.1

42.5.1 VMM Controls to Configure Guest Support of Intel® SGX

The Intel SGX capability is primarily exposed to the software via CPUID instruction. VMMs can virtualize CPUID instruction to expose/hide this capability to/from guests.

Next, the various parameters related to Intel SGX resources (such as EPC size, EPC location, etc.) are exposed/controlled via model-specific registers. VMMs can virtualize these MSRs for the guests using standard RDMSR/WRMSR hooks.

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 EPTs or shadow page tables.

The VMM can hook into the ENCLS instruction by setting the new VM-exiting control called "enable ENCLS exiting" (bit 15 in the secondary processor-based VM-execution controls). Support for the 1-setting of this control will be enumerated in the VMX capability MSRs (see Section 42.5.1.1).

If the "enable ENCLS exiting" control is 0 on a VM entry, 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 called "ENCLS-exiting bitmap" (encoding pair 0202EH/0202FH).

When bits in the "ENCLS-exiting bitmap" are set, execution of the corresponding ENCLS leaf functions in the guest results in a VM exit.

The priority of "ENCLS-exiting bitmap" check is immediately below the CPL check. This field exists only on processors that support the 1-setting of "enable ENCLS exiting".

Processors that do not support Intel SGX, i.e. CPUID.(EAX=07H, ECX=0):EBX.SGX = 0, the following items hold:

- VMX capability MSRS enumerate the 1-setting of "enable ENCLS exiting" is not supported.
- VM entries with "enable ENCLS exiting" field set to 1 will fail.
- VMREAD/VMWRITE of the "ENCLS-exiting bitmap" will fail due to access to an unsupported VMCS field.

42.5.1.1 Guest State Area - Guest Non-Register State

Table 42-2. Guest Interruptibility State

Position	Field	Value
0	Blocking by STI	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*
1	Blocking by MOV SS	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*
3	Blocking by SMI	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*
4	Blocking by NMI	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*
5	ENCLAVE_INTERRUPTION	See Section 42.5.5.

42.5.1.2 VM-Execution Controls

VM-Execution controls related to Intel SGX include a ENCLS-exiting bitmap (accessed via VMCS encoding pair 0202EH/0202FH) and the "Enable ENCLS exiting" control at bit 15 of the secondary processor based VM execution controls. The ENCLS-exiting bitmap provides bit fields for VMM to permit individual ENCLS leaf functions to execute without causing a VM exit in a guest, see "ENCLS—Execute an Enclave System Function of Specified Leaf Number". If bit 31 of the primary processor-based VM execution controls is 0, the processor functions as if the Enable ENCLS Exiting bit was set to 0.

Table 42-3. Secondary Processor Based VM Execution Controls

Position	Field	Value
14:0	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*	
15	Enable ENCLS exiting	Enable ENCLS-exiting bitmap for ENCLS leaf functions.
31:16	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*	

42.5.1.3 Basic VM-Exit Information

The VM-exit information fields adds bit 27 to provide information on VM exits due to the interaction between enclave and asynchronous events.

Table 42-4. Format of Exit Reason

Bit Position	Value
15:0	Basic exit reason: See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*
26:16	Reserved: See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*
27	ENCLAVE_INTERRUPTION: see Section 42.5.2.
31:28	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*

The encodings of Basic Exit Reason can indicate if the VM exit is related to executing ENCLS leaf functions.

Table 42-5. Basic Exit Reasons

Basic Exit Reason	Value
0 through 59	See Appendix C of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*
60	ENCLS.

42.5.2 VM Exits While Inside an Enclave

All VM exits that originate on an instruction boundary inside an enclave set a new bit called the "Enclave Interruption" bit (bit position 4) in the VMCS Guest Interruptibility State field (field encoding 4824H, Table 42-2) and in the EXIT_REASON field (bit 27) of the VMCS before delivering the VM exit to the VMM. Any VM exit (except for failed VM-entry VM exit) that sets the ENCLAVE_INTERRUPTION bit in GUEST_INTERRUPTIBILITY state, also sets Bit 27 in the EXIT_REASON field. These VM exit conditions include:

- Direct VM exits caused by exceptions, interrupts, and NMIs that happen while the logical processor is executing inside an enclave.
- Indirect VM exits triggered by interrupts, exceptions, and NMIs that happen while the logical processor is executing inside an enclave.
  - This includes VM exits encountered during vectoring due to EPT violations, task switch, etc.
- Parallel VM exits caused by SMI that is received while the logical processor is executing inside an enclave.
- All other VM exits that happen on an instruction boundary that is inside an enclave.

IA32/Intel 64 Architectures define very strict priority ordering between classes of events that are received on the same instruction boundary, and such ordering requires careful attention to cross-interactions between events. See Section 42.6 for details of interactions of architecturally visible events with Intel SGX architecture.

All processor states saved in the VMCS on VM exits from an enclave contain synthetic state. See Table 40-2 for details of the state saved into the VMCS.

A failed VM-entry VM exit will not set the ENCLAVE_INTERRUPTION bit in EXIT_REASON but since it will not save the GUEST_INTERRUPTIBILITY_STATE, the original value of the ENCLAVE_INTERRUPTION bit will remain untouched in GUEST_INTERRUPTIBILITY_STATE.

42.5.3 VM Entry Consistency Checks and Intel® SGX

A VM entry will perform consistency checks according to those described in Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

Additionally, VM entry may allow the VM-entry instruction-length field having a value 0 if the following items all hold true:

- IA32_VMX_MISC[30] as 1.
- The valid bit (bit 31) of the VM-entry interruption-information field in the current VMCS is 1.
- the interruption type (bits 10:8)of the VM-entry interruption-information field has value 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

42.5.4 VM Execution Control Setting Checks

A VM entry will perform consistency checks according to those described in Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. Additional consistency check on VM-execution control fields includes:

- If CPUID.(EAX=07H, ECX=0):EBX.SGX = 0, and if the "ENCLS Exiting" control (bit 15 in the secondary processor-based VM-execution controls) is set, then the VM entry fails, which sets RFLAGS.ZF=1 and error code=7 (VM entry with invalid control field).

42.5.5 Guest Interruptibility State Checks

If the ENCLAVE_INTERRUPTION bit in VM-entry control field is set and if CPUID.(EAX=07H, ECX=0):EBX.SGX = 0, VM entry will fail.

If both the MOV-SS blocking and ENCLAVE_INTERRUPTION bits are set in the interruptibility-state field in the guest-state area of the VMCS, VM entry leads to a Failed VMENTRY/VMEXIT, error code 33. Note that, since the MOV SS and POP SS instructions are illegal inside an enclave, no VM exit will set the interruptibility-state field with both bits set.

If the ENCLAVE_INTERRUPTION bit is set in the interruptibility-state field of the VMCS, and a VM entry leads to a VMEXIT during event injection, then the VM exit sets the ENCLAVE_INTERRUPTION bit. Such a transition does not include an asynchronous enclave exit and consequently, neither the processor's architectural state, nor the state saved in the guest-state area of the VMCS is synthesized as is done during asynchronous enclave exits (for example: there is no clearing of the GPRs or of VMCS fields such as the VM-exit instruction length or the low 12 bits in certain address fields in the VMCS).

42.5.6 Interaction of Intel® SGX with Various VMMs

If IA32_VMX_MISC.[bit 30] = 0, permitted VM entry instruction lengths are 1-15 bytes. If IA32_VMX_MISC.[bit 30] = 1, permitted VM entry instruction lengths allow 0 as a legal value for interruption type 4(software interrupt), 5 (privileged software exception), or 6 (software exception).

42.5.7 Interactions with EPTs

Intel SGX instructions are fully compatible with Extended Page Tables.

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the data segments, and the linear addresses generated by combining these offsets with DS segment register are subject to paging and EPT-based access control.

The Intel SGX access control itself is implemented as an extension to the traditional IA-32 paging/EPT state machine. See Section 38.5 for details of this extension.

Intel SGX instructions may set A and D bit on non-faulting EPC pages, even if the instruction may eventually fault due to some other reason, in IA page tables and EPT page tables when enabled.

42.5.8 Interactions with APIC Virtualization

The Intel SGX architecture interacts with APIC virtualization due to its interactions with the APIC access page as well as Virtual APIC Page. See Section 42.11.2 for interactions of the Intel SGX architecture with the Virtual APIC Page, and to Section 42.11.4 for the interactions of Intel SGX architecture with the APIC Access Page.

42.5.9 Interactions with Monitor Trap Flag

The interactions of Intel SGX with the Monitor Trap Flag are documented in Section 43.2.

42.5.10 Interactions with Interrupt-Virtualization Features and Events

If software is executing in an enclave and a VM exit would occur that would report "interrupt window" as basic exit reason (due to the 1-setting of the "interrupt window exiting" VM-execution control), an AEX occurs before the VM exit is delivered.

If software is executing in an enclave and a virtual interrupt would be delivered through the IDT (due to the 1-setting of the "virtual interrupt delivery" VM-execution control), an AEX occurs before delivery of the virtual interrupt.

If software is executing in an enclave and an external interrupt arrives that would cause a VM exit (due to the 1-setting of the "external interrupt exiting" VM-execution control), an AEX occurs before the VM exit is delivered.

If software is executing in an enclave and an external interrupt arrives that would cause virtual interrupts to be posted to the virtual-IRR field in the virtual-APIC page (due to the 1-setting of the "process posted interrupts" VM-execution control), an AEX may or may not occur before the posting of the virtual interrupts. This behavior is implementation specific.

42.6 INTEL® SGX INTERACTIONS WITH ARCHITECTURALLY-VISIBLE EVENTS

All architecturally visible vectored events (IA32 exceptions, interrupts, SMI, NMI, INIT, VM exit) that are detected while inside an enclave cause an asynchronous enclave exit. Additionally, INT3, entry/redirection, and the SignalTXTMsg[SENTER] events also cause asynchronous enclave exits. Note that SignalTXTMsg[SEXIT] 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 INTERACTIONS WITH THE XSAVE/XRSTOR PROCESSOR EXTENDED STATES

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 in the state-save area (SSA), and clear the secrets in the processor extended state, if the extended state is being used by an enclave.
- Intel SGX architecture must ensure that erroneous XCR0 and/or XBV_HEADER settings by system software do not result in SSA overflow.
- Enclave software should be able to discover only those processor extended states for which such protection is enabled.
- 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:

— Certain processor extended state (e.g., Memory Protection Extensions, see Chapter 9 of *Intel® Architecture Instruction Set Extensions Programming Reference*) 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.

— 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 entry, after certain consistency checks, the value in the XCR0 is saved in a micro-architectural location, 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 ENCLU[EGETKEY]).

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 ENCLU[ERESUME], the saved state for processor extended states enabled by XFRM is restored.

42.7.2 Relevant Fields in Various Data Structures

42.7.2.1 SECS.ATTRIBUTES.XFRM

The ATTRIBUTES field of the SECS data structure (see Section 38.7) contains a sub-field called X-Feature Request Mask (XFRM). Software populates this field at the time of enclave creation indicating the processor extended state configuration required by the enclave.

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 and the XSAVE area (x87 and XMM states are stored in the latter area). The specified size must be large enough to hold all the general-purpose registers, additional Intel

SGX specific information, plus the state size of set of processor extended states specified by SECS.ATTRIBUTES.XFRM (see Section 38.9 for the layout of SSA). 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 returns 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.

42.7.2.3 XSAVE Area in SSA

The XSAVE area of an SSA frame begins at offset 0 of the frame.

42.7.3 Processor Extended States and ENCLS[ECREATE]

The ECREATE leaf of the ENCLS instruction enforces a number of consistency checks described earlier. The execution of ENCLS[ECREATE] instruction results in a #GP(0) exception 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.
  - SSAFRAMESIZE*4096 < 168 + X, where X is the value that would be returned in EBX if CPUID were executed with EAX=0DH, ECX=0, and XCR0 was loaded with the value of XFRM.

1. It is the responsibility of the enclave to actually allocate this memory.

42.7.4 Processor Extended States and ENCLU[EENTER]

42.7.4.1 Fault Checking

The EENTER leaf of ENCLU instruction enforces a number of consistency requirements described earlier. Specifically, the ENCLU[EENTER] instruction results in a #GP(0) exception 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.

42.7.4.2 State Loading

If ENCLU[EENTER] is successful, it saves the current value of XCR0 in a micro-architectural location and sets XCR0 to SECS.ATTRIBUTES.XFRM.

42.7.5 Processor Extended States and AEX

42.7.5.1 State Saving

On an AEX, processor extended states are saved into the XSAVE area of the SSA frame as if the XSAVE instruction was executed with EDX:EAX = SECS.ATTRIBUTES.XFRM, with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if REX.W=1. The XSTATE_BV part of the XSAVE header is saved with 0 for every bit that is 0 in XFRM. Other bits may be saved as 0 if the state saved is initialized.

Note that enclave entry ensures that if CR4.OSXSAVE is set to 0, then SECS.ATTRIBUTES.XFRM is set to 3. It should also be noted that it is not possible to enter an enclave with FXSAVE disabled. While AEX is defined to save data as XSAVE would, implementations may use FXSAVE flows if CR4.OSXSAVE=0. In this case, the implementation ensures that the non-state data is consistent with the XSAVE format, and not the FXSAVE format (e.g., the XSAVE header).

42.7.5.2 State Synthesis

After saving state, AEXs restore XCR0 to the value it held at the time of the most recent enclave entry.

The state of features corresponding to bits set in XFRM is synthesized. In general, these states are initialized. Details of state synthesis on AEX are documented in Section 40.3.1.

42.7.6 Processor Extended States and ENCLU[ERESUME]

42.7.6.1 Fault Checking

The ERESUME leaf of ENCLU instruction enforces a number of consistency requirements described earlier. Specifically, the ENCLU[ERESUME] instruction results in a #GP(0) exception 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] instruction to fault. Examples include 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. In this case, it is the responsibility of the processor to generate faults that are caused by XRSTOR and not by FXRSTOR.

42.7.6.2 State Loading

If ENCLU[ERESUME] is successful, it saves the current value of XCR0 microarchitecturally and sets XCR0 to XFRM.

State is loaded from the XSAVE area of the SSA frame as if the XRSTOR instruction were executed with XCR0=XFRM, EDX:EAX = XFRM, with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if REX.W=1. The XSTATE_BV part of the XSAVE header is saved with 0 for every bit that is 0 in XFRM, as a non-compacted buffer. Other bits may be saved as 0 if the state saved is initialized.

ENCLU[ERESUME] ensures that a subsequent execution of XSAVEOPT inside the enclave will operate properly (e.g., by marking all state as modified).

42.7.7 Processor Extended States and ENCLU[EEXIT]

The ENCLU[EEXIT] instruction does not perform any X-feature specific consistency checks. However, successful execution of the ENCLU[EEXIT] instruction restores XCR0 to the value it held at the time of the most recent enclave entry.

42.8 INTERACTIONS WITH SMM

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 a #UD exception.

42.8.2 SMI while Inside an Enclave

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 42-4) and in the Guest Interruptibility State field (see Table 42-2) of the SMM transfer VMCS.

An SMI received immediately after ERESUME results in an asynchronous exit. The asynchronous exit does not set a pending MTF indication, and consequently, no pending MTF indication is saved inside the SMRAM. After RSM the processor will re-establish the MTF VMCS execution control.

42.8.3 SMRAM Synthetic State of AEX Triggered by SMI

All processor registers saved in the SMRAM have the same synthetic values listed in Section 40.3. Additional SMRAM fields that are treated specially on SMI are:

Table 42-6. SMRAM Synthetic States on Asynchronous Enclave Exit

Position	Field	Value
SMRAM Offset 07EE0H.Bit 1	ENCLAVE_INTERRUPTION	Set to 1 if exit occurred in enclave mode.

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 (WFS) state.

INIT received inside an enclave, while the logical processor (LP) is in VMX-root operation, is blocked until either the LP exits VMX operation (via VMXOFF) or enters VMX-non-root operation (via VMLAUNCH or VMRESUME). Since VMXOFF, VMLAUNCH, and VMRESUME cause a CPL-based #GP inside an enclave, such an INIT remains blocked at least until the LP exits the enclave.

INIT received inside an enclave, while the logical processor is in VMX-non-root operation, causes an AEX. Subsequent to the AEX, the INIT is delivered to the VMM via appropriate VM exit with INSIDE_ENCLAVE bit in the VMCS.EXIT_REASON set.

A processor cannot be inside an enclave in WFS state. Consequently, a SIPI received while inside an enclave is lost.

If a processor is in WFS state outside VMX operation, receipt of SIPI vectors the processor to 000VV000H to run BIOS-initialization code. If a processor is in WFS state in VMX-non-root operation, receipt of SIPI causes the LP to deliver appropriate VM exit. A processor cannot be in WFS state in VMX-root operation. In either case, the behavior of the LP on SIPI while in WFS state does not change for Intel SGX.

INIT is considered a warm reset, which keeps all the cache state, RR state, and feature-configuration state unmodified. Consequently, subsequently to INIT, CPUID enumeration of Intel SGX feature remains intact.

The SGX-related processor states after INIT-SIPI-SIPI is as follows:

- EPCM: Unchanged.
- CPUID.LEAF_12H.*: Unchanged.
- ENCLAVE_MODE: 0 (LP exits enclave asynchronously).
- MEE state: Unchanged.

OSes that use INIT-SIPI-SIPI only during initial boot (i.e., only after reset) can blindly assume that the entire EPC is empty, and every entry in the EPCM is marked invalid. These OSes do not need to use EREMOVE after INIT-SIPI-SIPI.

OSes that use INIT-SIPI-SIPI for dynamic offlining of a processor should use software conventions for communicating the EPCM and other state with the processors that are offlined/onlined dynamically.

42.10 INTERACTIONS WITH DMA

DMA is not allowed to access any Processor Reserved Memory.

42.11 INTERACTIONS WITH MEMORY CONFIGURATION AND VARIOUS MEMORY RANGES

42.11.1 Interactions of Intel® SGX with APIC Access Address

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 38.3). Consequently, all implicit accesses are always targeted at a page inside an EPC.

For all other memory accesses, physical-address matches against the APIC-access address occur before checking for other range register matches.

An Enclave Access (a linear memory access which is either done by from within an enclave into it 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 (either SGX1 or SGX2) or by a logical processor inside an enclave that 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 results in abort-page semantics if these accesses eventually reach EPC. This applies to any physical accesses that are redirected to the virtual-APIC page.

While a logical processor inside an enclave, the checking of the instruction pointer's linear address against the enclave's linear-address range (ELRANGE) is done before checking the physical address to which the linear address translates against the APIC-access page. Thus, an attempt to execute an instruction outside ELRANGE, the instruction fetch results in a #GP(0), even if the linear address would translate to a physical address overlaps the APIC-access page.

42.12 INTERACTIONS WITH TXT

42.12.1 Enclaves Created Prior to Execution of GETSEC

Enclaves which have been created before the GETSEC[SENTER] instruction are available for execution after the successful completion of GETSEC[SENTER] and the corresponding SINIT ACM. Intel SGX will need to be re-enabled by the software launched by GETSEC[SENTER], in addition to any other actions a TXT launched environment performs when preparing to execute code which was running previously to GETSEC[SENTER].

42.12.2 Interaction of GETSEC with Intel® SGX

All leaf functions of the GETSEC instruction are illegal inside an enclave, and result in #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 threads executing 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.

Processors supporting Intel SGX also require that the ACM-verification key be located on die, and that such ACMs contain a new header field.

42.12.3 Interactions with Authenticated Code Modules (ACMs)

After execution of any non-faulting Intel SGX instructions, the Intel SGX architecture forbids the launching of ACMs with Intel SGX SVN that is lower than the expected Intel SGX SVN threshold that was specified by BIOS. The non-faulting Intel SGX instructions refer to Intel SGX instruction leaves that do not return error code and executed successfully without causing an exception. 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 42-7.

Table 42-7. 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. 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. 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. 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 INTERACTIONS WITH CACHING OF LINEAR-ADDRESS TRANSLATIONS

Entering and exiting an enclave causes the logical processor to flush all the global linear-address context as well as the linear-address context associated with the current VPID and PCID. The MONITOR FSM is also cleared.

42.14 INTERACTIONS WITH INTEL® TRANSACTIONAL SYNCHRONIZATION EXTENSIONS (INTEL® TSX)

1. ENCLU or ENCLS instructions inside an HLE region will cause the flow to be aborted and restarted non-speculatively. ENCLU or ENCLS instructions inside an RTM region will cause the flow to be aborted and transfer control to the fallback handler.

2. If XBEGIN is executed inside an enclave, the processor does NOT check whether the address of the fallback handler is within the enclave.

3. If an RTM transaction is executing inside an enclave and there is an attempt to fetch an instruction outside the enclave, the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

4. If an RTM transaction is executing inside an enclave and there is a data access to an address within the enclave that denied due to EPCM content (e.g., to a page belonging to a different enclave), the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

5. If an RTM transaction executing inside an enclave aborts and the address of the fallback handler is outside the enclave, a #GP is delivered after the abort (EIP reported is that of the fallback handler).

42.14.1 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 (any #BP is converted to #DB) - imply an Intel SGX AEX. 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).

42.15 INTEL® SGX INTERACTIONS WITH S STATES

Whenever an Intel SGX enabled processor leaves the S0 or S1 state for S2-S5 state, enclaves are destroyed. This is due to the EPC being destroyed when power down occurs.

42.16 INTEL® SGX INTERACTIONS WITH MACHINE CHECK ARCHITECTURE (MCA)

42.16.1 Interactions with MCA Events

All architecturally visible machine check events (#MC and CMCI) that are detected while inside an enclave cause an asynchronous enclave exit.

Any machine check exception (#MC) that occurs after Intel SGX is first enables causes Intel SGX to be disabled, (CPUID.SGX_Leaf.0:EAX[SGX1] == 0). It cannot be enabled until after the next reset.

42.16.2 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[SE1] to 0) until the next reset.

42.16.3 CR4.MCE

CR4.MCE can be set or cleared with no interactions with Intel SGX.

42.17 INTEL® SGX INTERACTIONS WITH PROTECTED MODE VIRTUAL INTERRUPTS

ENCLS[EENTER] modifies neither EFLAGS.VIP nor EFLAGS.VIF.

ENCLS[ERESUME] loads EFLAGS in a manner similar to that of an execution of IRET with CPL = 3. This means that ERESUME modifies neither EFLAGS.VIP nor EFLAGS.VIF regardless of the value of the EFLAGS image in the SSA frame.

AEX saves EFLAGS.VIP and EFLAGS.VIF unmodified into the EFLAGS image in the SSA frame. AEX modifies neither EFLAGS.VIP nor EFLAGS.VIF after saving EFLAGS.

If CR4.PVI = 1, CPL = 3, EFLAGS.VM = 0, IOPL < 3, EFLAGS.VIP = 1, and EFLAGS.VIF = 0, execution of STI causes a #GP fault. In this case, STI modifies neither EFLAGS.IF nor EFLAGS.VIF. This behavior applies without change within an enclave (where CPL is always 3). Note that, if IOPL = 3, STI always sets EFLAGS.IF without fault; CR4.PVI, EFLAGS.VIP, and EFLAGS.VIF are neither consulted nor modified in this case.

...

32. New Chapter 43, New Volume 3D

A new chapter, Chapter 43, has been added to the new volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:* System Programming Guide, Part 4.

...

CHAPTER 43
ENCLAVE CODE DEBUG AND PROFILING

Intel® SGX is architected to provide protection for production enclaves and permit enclave code developers to use an SGX-aware debugger to effectively debug a non-production enclave (debug enclave). Intel SGX also allows a non-SGX-aware debugger to debug non-enclave portions of the application without getting confused by enclave instructions.

43.1 CONFIGURATION AND CONTROLS

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 a special SIGSTRUCT to be generated for this matter.

The ATTRIBUTES field in the SECS is reported in the enclave's attestation, and is included in the key derivation for the 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, 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 SINGLE STEP DEBUG

43.2.1 Single Stepping Requirements

The following requirements are identified for the single-stepping architecture:

- The privileged Intel SGX instruction ENCLS must exhibit legacy single-stepping behavior.
- If a debugger is not debugging an enclave, then the enclave should appear as a “giant instruction” to the debugger.
- The architecture must allow an SGX-capable debugger and a debug enclave to single-step within an enclave that it wants to debug in a fashion that is consistent with the IA32/Intel 64 legacy prior to the introduction of Intel SGX.

43.2.2 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 on the instruction boundary immediately after the ENCLS instruction. Additionally, if the instruction is invoked from a VMX guest, and if the monitor trap flag is asserted at the time of the time of invocation, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

43.2.3 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 clear the RFLAGS.TF bit, and suppress the monitor trap flag. In such situations, an exit from the enclave, either via the EEXIT leaf function or via an AEX restores the RFLAGS.TF bit and effectiveness of the monitor trap flag. The details of this clearing/suppression and the exact pending of single stepping events across EENTER/ERESUME/EEXIT/AEX are covered in detail in Section 43.2.4.

If the RFLAGS.TF bit is set at the beginning of EREPORT or EGETKEY leafs, then a single-step debug exception is pending on the instruction boundary immediately after the ENCLU instruction. Additionally, if the instruction is invoked from a VMX guest, and if the monitor trap flag is asserted at the time of invocation, and if the monitor trap flag is not suppressed by the preceding enclave entry, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

Consistent with the IA32 and Intel® 64 architectures, a pending MTF VM exit takes priority over a pending debug exception. Additionally, if an SMI, an INIT, or an #MC is received on the same instruction boundary, then that event takes priority over both the pending MTF VM exit and the pending debug exception. In such a situation, the pending MTF VM exit and/or pending debug exception are handled in a manner consistent with the IA32 and Intel 64 architectures.

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 from a VMX guest, and if the VMM has asserted the monitor trap flag, then an MTF VM exit is pending after the delivery of the fault through the IDT (i.e., before the first instruction of the OS handler). If a VM exit occurs before reaching that boundary, then the MTF VM exit is lost.

43.2.4 Single-stepping Enclave Entry with Opt-out Entry

43.2.4.1 Single Stepping without AEX

Figure 43-1 shows the most common case for single-stepping after an opt-out entry.

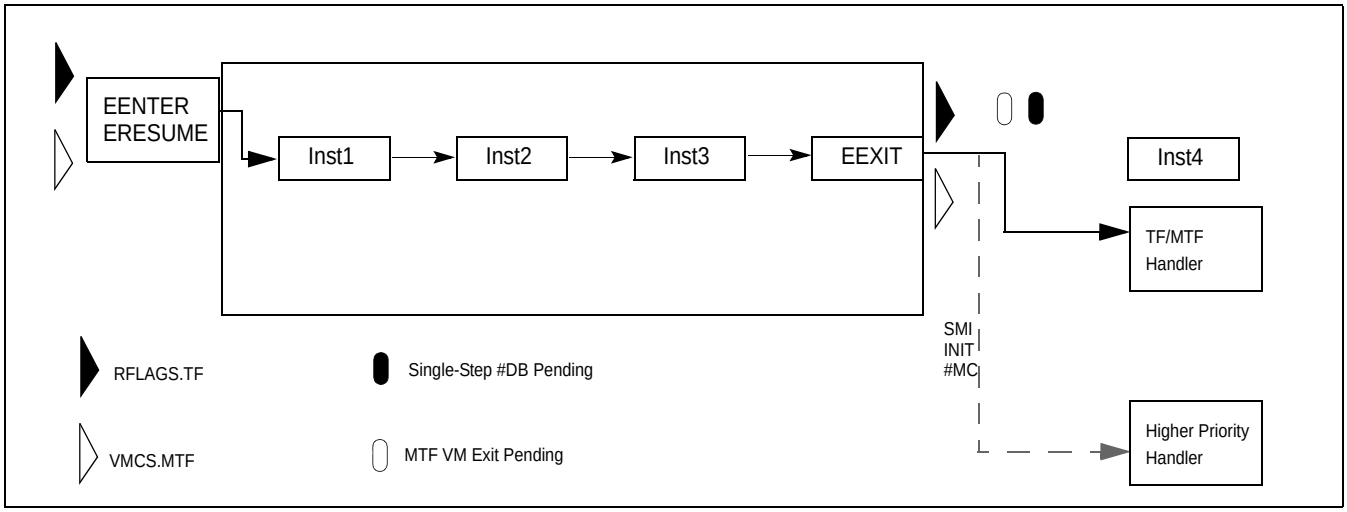


Figure 43-1. Single Stepping with Opt-out Entry - No AEX

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 a VMX guest, and if the monitor trap flag is asserted at the time of the enclave entry, then 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 same as the value of RFLAGS.TF at the time of the enclave entry. Similarly, if the enclave is executing inside a VMX guest, then the value of the monitor trap flag after EEXIT is same as the value of that control at the time of the enclave entry.

Consistent with the IA32 and Intel 64 architectures, MTF VM exit, if pending, takes priority over a pending debug exception. If an SMI, an INIT, or an MC# is received on the same instruction boundary, then that event takes priority over both the pending MTF VM exit and the pending debug exception. In such a situation, the pending MTF VM exit and/or pending debug exception are handled in a manner consistent with the IA32 and Intel 64 architecture.

43.2.4.2 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.

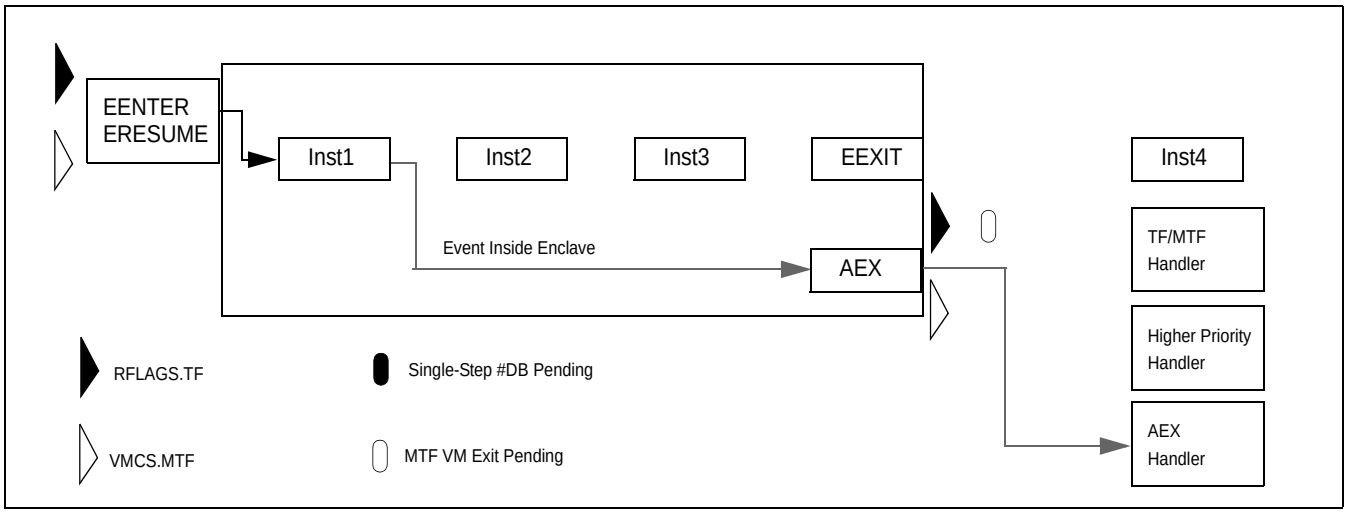


Figure 43-2. Single Stepping with Opt-out Entry -AEX Due to Non-SMI Event Before Single-Step Boundary

In this scenario, if the enclave is executing in a VMX guest, and if the monitor trap flag is asserted at the time of the enclave entry, then an MTF VM exit is pending on the instruction boundary after the delivery of the AEX. Consistent with the IA32 and Intel 64 architectures, if another VM exit happens before reaching that instruction boundary, the MTF VM exit is lost.

The value of the RFLAGS.TF bit at the end of AEX is same as the value of RFLAGS.TF at the time of the enclave entry. Also, if the enclave is executing inside a VMX guest, then the value of the monitor trap flag after AEX is the same as the value of that control at the time of the enclave entry.

43.2.5 RFLAGS.TF Treatment on AEX

When an opt-in enclave takes an AEX, RFLAGS.TF passes unmodified into synthetic state, and is saved as RFLAGS.TF=0 in the GPR portion of the SSA. For opt-out entry, the external value of TF is saved in CR_SAVE_TF, and TF is then cleared. For more detail see EENTER and ERESUME in Chapter 5.

43.2.6 Restriction on Setting of TF after an Opt-out Entry

From an opt-out EENTER or ERESUME until the next enclave exit, enclave is not allowed to set RFLAGS.TF. In such a situation, the POPF instruction forces RFLAGS.TF to 0 if the enclave was entered through TCS with DBGOPTIN=0.

43.2.7 Trampoline Code Considerations

Any AEX from the enclave which results in the RFLAGS.TF =1 on the reporting stack will result in a single-step #DB after the first instruction of the trampoline code if the trampoline is entered using the IRET instruction.

43.3 CODE AND DATA BREAKPOINTS

43.3.1 Breakpoint Suppression

On an opt-out entry into an enclave, all code and data breakpoints that overlap with the ELRANGE are suppressed. On any entry (either opt-in or opt-out) into an enclave, all code breakpoints that do not overlap with ELRANGE are also suppressed.

43.3.2 Breakpoint Match Reporting during Enclave Execution

The processor does not report any new 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.

Intel SGX architecture specifically forbids reporting of silent matches on any debug breakpoints that overlap with ELRANGE after an opt-out entry.

43.3.3 Reporting of Code Breakpoint on Next Instruction on a Debug Trap

If execution in an enclave encounters a single-step trap or an enabled data breakpoint, the logical processor performs an AEX. Following the AEX, the logical processor checks the new instruction pointer (the AEP address) against any code breakpoints programmed in DR0-DR3. Any matches are reported to software.

If execution in an enclave encounters an enabled code breakpoint, the logical processor checks the current instruction pointer (within the enclave) against any code breakpoints programmed in DR0-DR3. This checking for code breakpoints occurs before the AEX, the Intel SGX breakpoint-suppression architecture applies. Following this, the logical processor performs an AEX, after which any breakpoints matched earlier are reported to software.

43.3.4 RFLAGS.RF Treatment on AEX

RF is always set to 0 in synthetic state. This is because ERESUME after AEX is a new execution attempt.

RF value saved on SSA is the same as what would have been saved on stack in the non-SGX case. 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.

43.3.5 Breakpoint Matching in Intel® SGX Instruction Flows

None of the implicit accesses made by Intel SGX instructions to EPC regions generate 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 parameters do not fire any data breakpoints.

Explicit accesses made by the remaining Intel SGX instructions (ENCLU[EGETKEY] and ENCLU[EREPORT]), trigger precise data breakpoints for their EPC operands. It should also be noted that all Intel SGX instructions trigger precise data breakpoints for their non-EPC operands.

After an opt-out entry, ENCLU[EGETKEY] and ENCLU[EREPORT] do not fire any of the data breakpoints that were suppressed as a part of the enclave entry.

43.4 INT3 CONSIDERATION

43.4.1 Behavior of INT3 inside an Enclave

Inside an enclave, INT3 delivers a fault-class exception. However, the vector delivered as a result of executing the instruction depends on the manner in which the enclave was entered. Following opt-out entry, the instruction delivers #UD. Following opt-in entry, INT3 delivers #BP.

Since the event is a fault-class exception, the delivery flow of the exception does not check CPL against the DPL in the IDT gate. (Normally, delivery of INT3 generates a #GP if CPL is greater than the DPL field in IDT gate 3.) Additionally, the RIP saved in the SSA is always that of the INT3 instruction. The RIP saved on the stack/VMCS is that of the trampoline code as specified by the AEX architecture.

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 always fault-like inside an enclave. Consequently, the debugger must not decrement SSA.RIP for #BP coming from an enclave. INT3 will result in #UD, if the debugger is not attached to the enclave.

43.4.3 VMM Considerations

As described above, INT3 executed by enclave delivers #BP with "interruption type" of 3. This behavior will not cause any problems for VMMs that obtain VM-entry interruption information from appropriate VMCS field (as recommended in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*), and those VMMs will continue to work seamlessly.

VMMs that fabricate the VM-entry interruption information based on the interruption vector need additional enabling. Specifically, such VMMs should be modified to use injection type of 3 (instead of 6) when they see interruption vector 3 along with the VMCS "Enclave Interruption" bit set.

43.5 BRANCH TRACING

43.5.1 BTF Treatment

Any single-step traps pending after EENTER trigger BTF exception, as EENTER is considered a branch instruction. Additionally, any single-step traps pending after EEXIT trigger BTF exception, as EEXIT is also considered a branch instruction. ERESUME does not trigger BTF traps. An AEX does not trigger BTF or TF traps.

43.5.2 LBR Treatment

43.5.2.1 LBR Stack on Opt-in Entry

An opt-in enclave entry does not change the behavior of IA32_DEBUGCTL.LBR bit. Both enclave entry and enclave exit push a record on LBR stack. EENTER/ERESUME with TCS.FLAGS.DBGOPTIN=1, inserts a new LBR record on the LBR stack. The MSR_LASTBRANCH_n_FROM_IP of this record holds linear address of the EENTER/ERESUME

instruction, while MSR_LASTBRANCH_n_TO_IP of this record holds linear address of EENTER/ERESUME destination.

On EEXIT a new LBR record is pushed on the LBR stack. The MSR_LASTBRANCH_n_FROM_IP of this record holds linear address of the EEXIT instruction, while MSR_LASTBRANCH_n_TO_IP of this record holds the linear address of EEXIT destination.

On AEX a new LBR record is pushed on the LBR stack. The MSR_LASTBRANCH_n_FROM_IP of this record holds RIP saved in the SSA, while MSR_LASTBRANCH_n_TO_IP of this record holds RIP of the linear address of the AEP.

Additionally, for every branch inside the enclave, one record each is pushed on 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.

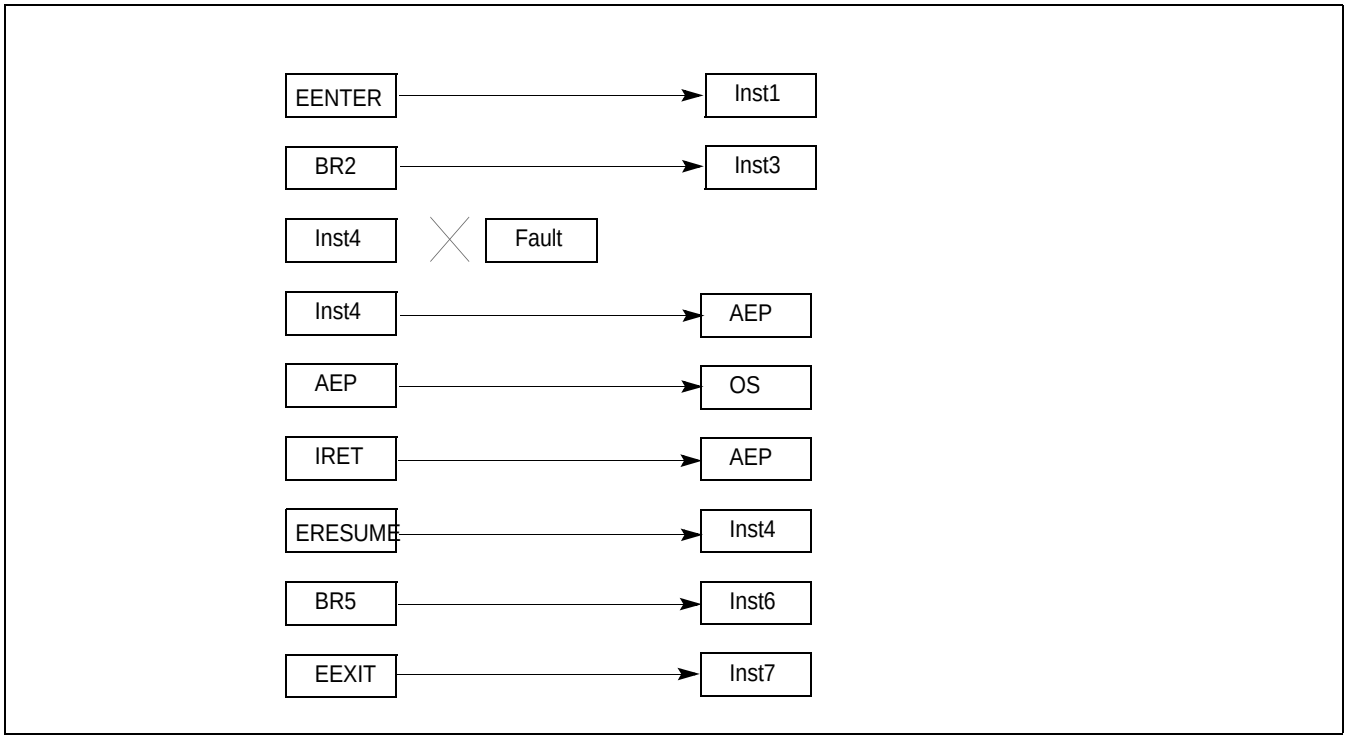


Figure 43-3. LBR Stack Interaction with Opt-in Entry

43.5.2.2 LBR Stack on Opt-out Entry

An opt-out entry into an enclave suppresses IA32_DEBUGCTL.LBR bit, and enclave exit after an opt-out entry unsuppresses the IA32_DEBUGCTL.LBR bit.

Opt-out entry into an enclave does not push any record on LBR stack.

If IA32_DEBUGCTL.LBR is set 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 that took the logical processor into the enclave, while the MSR_LASTBRANCH_n_TO_IP of such record holds linear address of the destination of EEXIT. Additionally, if IA32_DEBUGCTL.LBR is set 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. The MSR_LASTBRANCH_n_FROM_IP of the new record holds linear address of the instruction that took the LP into 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.

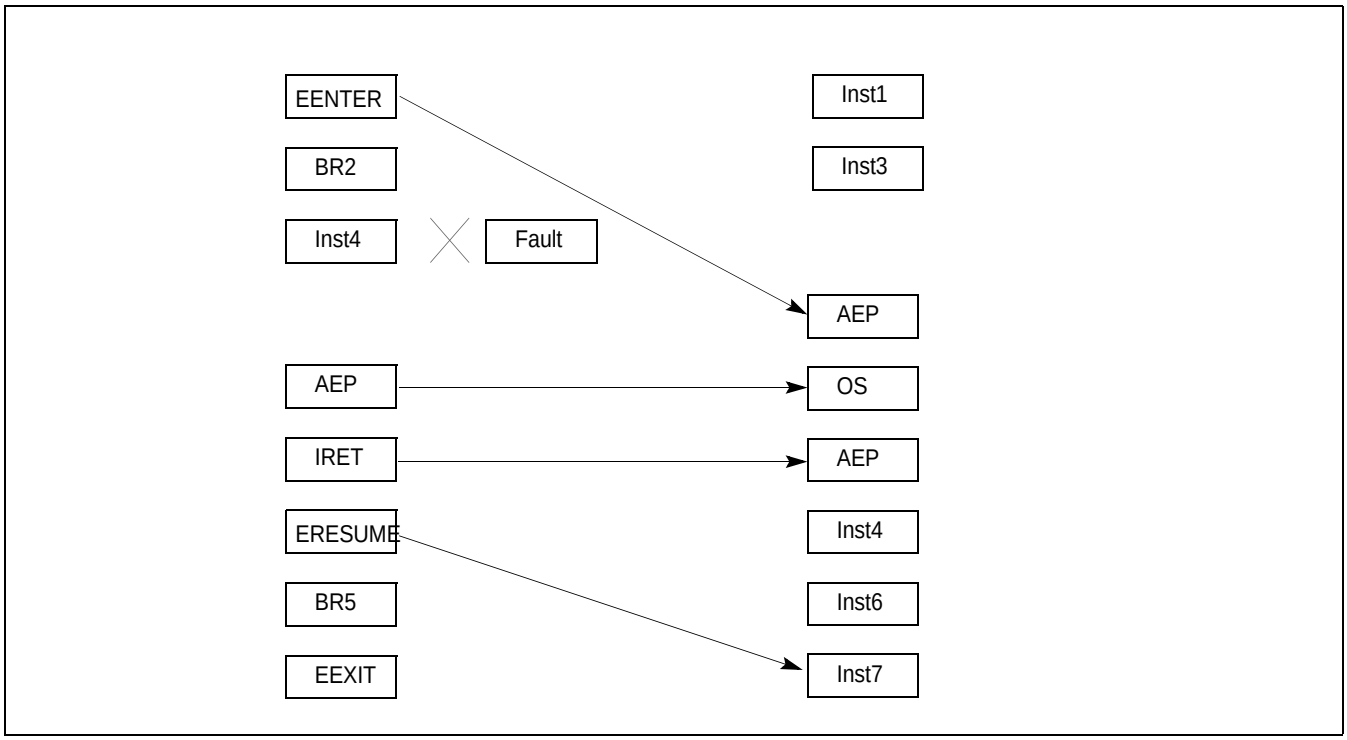


Figure 43-4. LBR Stack Interaction with Opt-out Entry

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.

For LBR filtering, 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 associated with the eventing logic. If the ASCI bit is set, the contents in various perfor-

mance 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 eventing 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 always cleared.

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.

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.

43.6.4 Enclave Exit and Performance Monitoring

When a logical processor exits an enclave, either via ENCLU[EEXIT] or via AEX, all performance monitoring activity (including PEBS) on that logical processor that was suppressed is unsuppressed.

Any counters that were demoted from AnyThread to MyThread on the sibling thread are promoted back to AnyThread.

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.

The behavior of EENTER and ERESUME leaf functions of the ENCLU instruction depends on whether these leaf functions are performing an opt-in entry or an opt-out entry. If these 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.

The behavior of the EEXIT leaf function is as follows. 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 (which is inside ELRANGE of the enclave just exited). 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 immediately 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 eventing LIP in the enclave. For opt-out entry, the record has the EVENTING_RIP set to EENTER/ERESUME LIP.

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.

It should be noted that a second PEBS event may be pended during the Enclave Exiting Event (EEE). If the PEBS event is taken at the end 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 OS/VMM is expected to pin the DS area in virtual memory. If the OS does not pin this area in memory, loads/stores to the PEBS or BTS buffer may incur faults (or other events such as APIC-access VM exit). Usually, such events are reported to the OS/VMM immediately, and generation of the PEBS/BTS record is skipped.

However, any events that are detected during PEBS/BTS record generation 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 is aborted (this may leave the buffers in a state where they have partial PEBS or BTS records), while trap-like events (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 branches, and any counters that are counting such branches are incremented by 1 as a part of execution of these instructions. All of these flows are also counted as instructions, and any counters configured appropriately are incremented by 1.

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.

EENTER, ERESUME, EEXIT and AEX are classified as far branches.

...

33. Updates to Appendix B, New Volume 3D

Change bars show changes to Appendix B in the new volume: *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
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
TSC offset (full)	000001000B	00002010H
TSC offset (high)		00002011H
Virtual-APIC address (full)[2]	000001001B	00002012H
Virtual-APIC address (high)[2]		00002013H
APIC-access address (full)[3]	000001010B	00002014H
APIC-access address (high)[3]		00002015H
Posted-interrupt descriptor address (full)[4]	000001011B	00002016H
Posted-interrupt descriptor address (high)[4]		00002017H
VM-function controls (full)[5]	000001100B	00002018H
VM-function controls (high)[5]		00002019H
EPT pointer (EPTP; full)[6]	000001101B	0000201AH
EPT pointer (EPTP; high)[6]		0000201BH
EOI-exit bitmap 0 (EOI_EXIT0; full)[7]	000001110B	0000201CH
EOI-exit bitmap 0 (EOI_EXIT0; high)[7]		0000201DH
EOI-exit bitmap 1 (EOI_EXIT1; full)[7]	000001111B	0000201EH
EOI-exit bitmap 1 (EOI_EXIT1; high)[7]		0000201FH
EOI-exit bitmap 2 (EOI_EXIT2; full)[7]	000010000B	00002020H
EOI-exit bitmap 2 (EOI_EXIT2; high)[7]		00002021H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
EOI-exit bitmap 3 (EOI_EXIT3; full)[7]	000010001B	00002022H
EOI-exit bitmap 3 (EOI_EXIT3; high)[7]		00002023H
EPTP-list address (full)[8]	000010010B	00002024H
EPTP-list address (high)[8]		00002025H
VMREAD-bitmap address (full)[9]	000010011B	00002026H
VMREAD-bitmap address (high)[9]		00002027H
VMWRITE-bitmap address (full)[9]	000010100B	00002028H
VMWRITE-bitmap address (high)[9]		00002029H
Virtualization-exception information address (full)[10]	000010101B	0000202AH
Virtualization-exception information address (high)[10]		0000202BH
XSS-exiting bitmap (full)[11]	000010110B	0000202CH
XSS-exiting bitmap (high)[11]		0000202DH
TSC multiplier (full)[12]	000011001B	00002032H
TSC multiplier (high)[12]		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 "use TPR shadow" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "virtualize APIC accesses" VM-execution control.
4. This field exists only on processors that support the 1-setting of the "process posted interrupts" VM-execution control.
5. This field exists only on processors that support the 1-setting of the "enable VM functions" VM-execution control.
6. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.
7. This field exists only on processors that support the 1-setting of the "virtual-interrupt delivery" VM-execution control.
8. This field exists only on processors that support the 1-setting of the "EPTP switching" VM-function control.
9. This field exists only on processors that support the 1-setting of the "VMCS shadowing" VM-execution control.
10. This field exists only on processors that support the 1-setting of the "EPT-violation #VE" VM-execution control.
11. This field exists only on processors that support the 1-setting of the "enable XSAVES/XRSTORS" VM-execution control.
12. This field exists only on processors that support the 1-setting of the "use TSC scaling" VM-execution control.

…